User Manual

DA16200 DA16600 DPM User Manual

UM-WI-030

Abstract

This document describes how to use DPM features in DA16200 and DA16600.

Contents

Figures

Tables

1 Terms and Definitions

AP	Access Point
API	Application Programming Interface
BUFP	Buffering Probe
BSS	Basic Service Set
DDPS	DPM Dynamic Period Setting
DPM	Dynamic Power Management
MCU	Micro Controller Unit
POR	Power on Reset
RTOS	Real Time Operating System
SSID	Service Set Identifier
TCP	Transmission Control Protocol

TIM	Traffic Indicator Module
UC	Unicast Packet
UDP	User Datagram Protocol

2 References

[1] UM-WI-056, FreeRTOS Getting Started Guide, Renesas Electronics
[2] UM-WI-003, Host Interfaces and AT Command Manual, Renesas Electronics
[3] UM-WI-006, Hardware Design Guide, Renesas Electronics

3 DPM

VirtualZero™ is a synthesis of breakthrough ultra-low-power technologies which enables extremely low power operation in the DA16200 (DA16600) SoC. VirtualZero™ shuts down every microelement of the chip that is not in use, which allows a near-zero level of power consumption when the DA16200 (DA16600) SoC does not actively transmit or receive data. Such low power operation can deliver a year or more of battery life depending on the application. VirtualZero™ also enables ultra-low-power operation to transmit and receive data when the SoC needs to be awake to exchange information with other devices. Advanced algorithms enable one to stay asleep until the exact moment required to wake up to transmit or receive. DA16200's (DA16600) DPM (Dynamic Power Management) APIs are used to realize VirtualZero™ in customer's application.

3.1 DPM Introduction

DA16200 (DA16600) supports DPM service on station only. And it has two modes which are Non-DPM and DPM modes. In Non-DPM mode, there is not much difference with other SDK, which is based on FreeRTOS and lwIP. DA16200 (DA16600) operates DPM service to support DPM service. It's working in DPM mode DA16200 (DA16600). To reduce power consumption, DPM service manages two states, which are DPM sleep and wakeup states in DPM mode. This session describes how DA16200 (DA16600) manages DPM modes and DPM states.

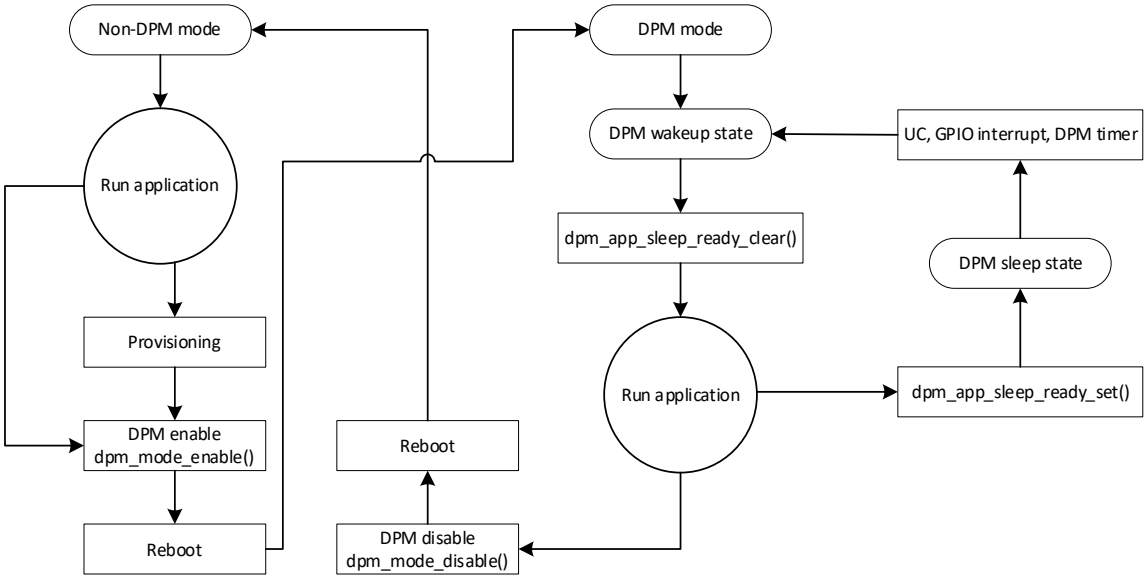

Figure 1: Overview of DA16200 (DA16600) Modes

The DA16200 (DA16600) SoC has two modes: Non-DPM and DPM modes.

- Non-DPM mode is the default mode of the DA16200 (DA16600) system. Regardless of applications state (running/idle/waiting on event / …) DA16200 (DA16600) is on all the time and ready to run a task.
- DPM mode can be enabled by variable ways like AT-Command and calling by dpm_mode_enable() API. Basically, provisioning to configure STA profile should be required. As mentioned before, DPM service is working on DA16200 (DA16600) station. If there is no STA profile in DPM mode, DA16200 (DA16600) is not able to connect AP (Access Point) and executes abnormal operation. It will work difference with what you expect. To enable DPM mode, system reboot is required. If the DA16200 (DA16600) is in DPM mode, it dynamically changes its power state either to sleep state (DPM Sleep State) or non-sleep state (DPM Wakeup State) depending on the states of applications who have registered to DPM service. It works like the following.

- DPM Sleep State: RTOS of DA16200 (DA16600) is inactive.
- DPM Wakeup State: RTOS of DA16200 (DA16600) is active.

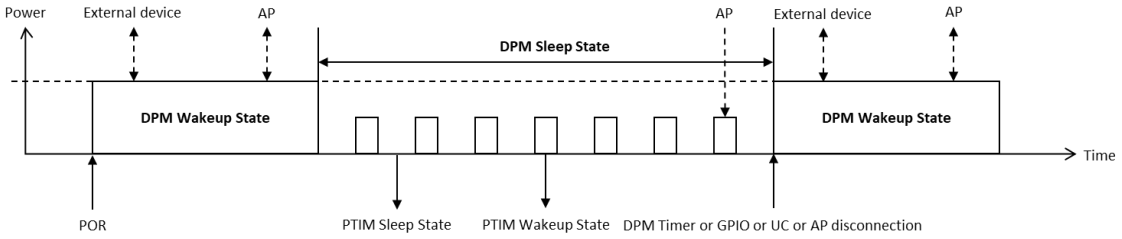

Figure 2: Flow of DPM mode

Applications can be registered to DPM service which will monitor them and change system's DPM state accordingly. For instance, if application (who registered to DPM service) becomes IDLE (there's no packets to handle) and notifies DPM service of it, DPM service automatically makes the system enter the sleep state if all applications registered to DPM service has declared "nothing to do currently / job done".

3.1.1 How DPM Works

To keep ultra-low-power consumption, DA16200 (DA16600) internally manages DPM states like to decide entering DPM sleep state or keeping DPM wakeup state. This session describes how DPM service works briefly.

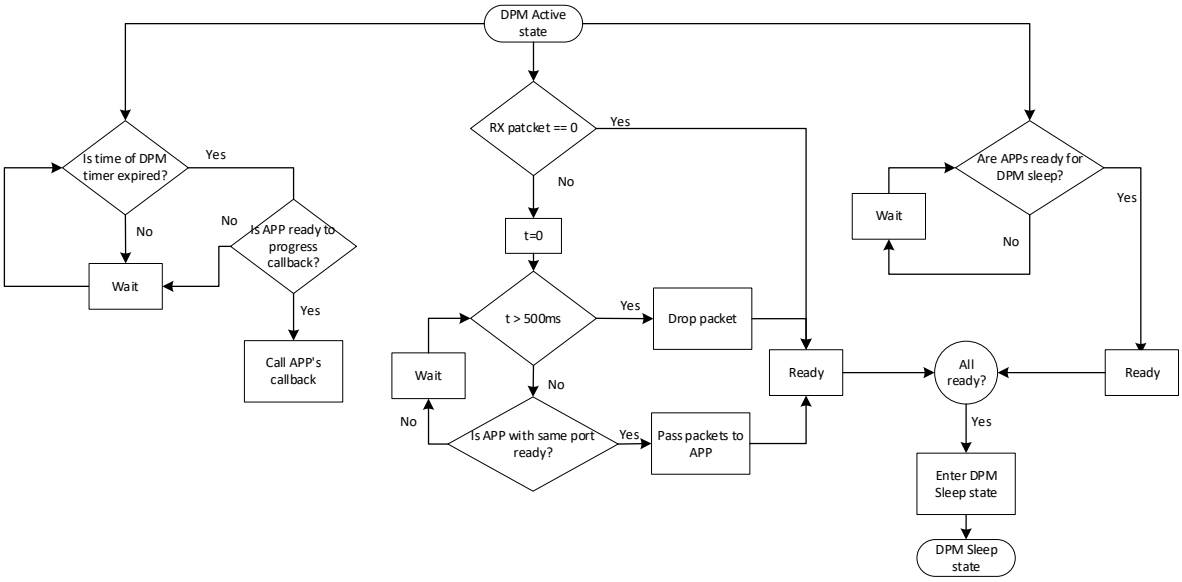

Figure 3: Flow of DPM Service

Normally, DPM service checks three conditions which are DPM timer, received packet, and state of registered applications. The conditions are parallelly running in each task. DA16200 (DA16600) enters DPM sleep state if there is no more operation.

The three conditions are related with application which is registered to DPM service. In DPM mode, registered application should notify the state to DPM service for working as expected. Therefore,

application should be registered. There are two ways to register application to DPM service. First, applicant can be registered by calling dpm_app_register() API. Second, application can be registered by user app table. DA16200 (DA16600) provides user app table to register application. The registered application will be automatically called after booting RTOS, also can be registered to DPM service.

Registered application can decide to enter DPM sleep state or not. The state can be changed by calling dpm_app_sleep_ready_set() and dpm_app_sleep_ready_clear() APIs.

If the application is related with network communication, it can notify to receive packet by calling dpm_app_data_rcv_ready_set() API. The application should be registered with the port number. If there is no port number, the received packet is possible to be dropped.

If it uses DPM timer, the application notifies the state to execute the registered call-back function by calling dpm_app_wakeup_done() API.

3.2 Application Programming Interface

The APIs listed in this session are for management of DPM mode.

Table 1: Management API List

`void dpm_mode_enable(void)`	
Description	Enable DPM mode

`void dpm_mode_disable(void)`	
Description	Disable DPM mode

`int dpm_mode_is_enabled(void)`	
Return	1 (pdTRUE): DPM mode is enabled 0 (pdFALSE): DPM mode is disabled
Description	Return current DPM mode is enabled or not

`int dpm_mode_is_wakeup(void)`	
Return	1 (pdTRUE): When DA16200 (DA16600) wakes up from DPM sleep 0 (pdFALSE): When DA16200 (DA16600) wakes up by other cases (POR) not DPM sleep
Description	Get whether DA16200 (DA16600) is waken-up by DPM timer or Receiving packet or External wake-up signal.

`int dpm_mode_get_wakeup_source(void)`	
Return	0x00 (WAKEUP_RESET): Internal reset add 0x01 (WAKEUP_SOURCE_EXT_SIGNAL): Boot from extern wake-up signal 0x02 (WAKEUP_SOURCE_WAKEUP_COUNTER): Boot from wake-up counter 0x03 (WAKEUP_EXT_SIG_WAKEUP_COUNTER): Boot from wakeup counter or external wake-up signal 0x04 (WAKEUP_SOURCE_POR): Boot from power on reset 0x08 (WAKEUP_WATCHDOG): Boot from watch dog add Others: Declared in the enumeration WAKEUP_SOURCE
Description	Get source of DPM wakeup state

int dpm_mode_get_wakeup_type(void)	
Return	DPM_RTCTIME_WAKEUP: Boot from DPM timer DPM_PACKET_WAKEUP: Boot from received packet Others: Declared in the enumeration DPM_WAKEUP_TYPE
Description	Get type of DPM wakeup state

int dpm_sleep_is_started(void)	
Return	0 (WAIT_DPM_SLEEP): Wait to enter DPM sleep 1 (RUN_DPM_SLEEP): Run to enter DPM sleep 2 (DONE_DPM_SLEEP): Done to enter DPM sleep
Description	Get state of what DA16200 (DA16600) enters DPM sleep

int dpm_sleep_start_mode_2(unsigned long long usec, unsigned char retention)		
Return		0: Succeed
Parameter	usec	Wake-up time; how long DA16200 is in DPM sleep mode 2. If 0, DA16200 wakes up only by external wake-up signal.
	retention	Power on/off RTM in DPM sleep mode 2
Description		Enter DPM sleep mode 2. DA16200 wakes up only by external wake-up signal or DPM timer.

3.2.1 Application Registration/Status Notification

The APIs listed in this session are for registering application for DPM service which will control the DA16200 (DA16600) SoC's power state based on applications' status. Once registered, applications are responsible to correctly notify their state to DPM service.

Table 2: Application API List

int dpm_app_register(char *mod_name, unsigned int port_number)		
Return		0 (DPM_REG_OK): Succeeded 9999 (DPM_REG_DUP_NAME): Failed because there is duplicated name of application in DPM service. Other: Failed due to other causes
Parameter	mod_name	Name of application to be registered to DPM service. The name must be less than 19 characters.
	port_number	Port number of application. If not required, the value can be 0.
Description		Register name of application. DPM service identifies a DPM enabled application with the name. DA16200 (DA16600) can be waken-up by some reason such as received packet, DPM timer, and GPIO. If the application is related with network application, it may require initialization time much more. DPM service holds the received packet which includes the port number before application is done to be initialization. After application notifies to be done of initialization, DPM service pass the received packet.

void dpm_app_unregister(char *mod_name)		
Parameter	mod_name	Name of registered application
Description		Deregister name of application

int dpm_app_is_register(char *mod_name)		
Return		9999 (DPM_REG_DUP_NAME): Registered application Other: Failed due to other causes
Parameter	mod_name	Name of registered application
Description		Check whether the application is registered or not

char *dpm_app_is_register_port(unsigned int port)		
Return		Pointer of name of application if the port number is registered
Parameter	mod_name	Port number of registered application
Description		Check what the registered name of application is with the port number

int dpm_app_sleep_ready_set(char *mod_name)		
Return		0 (DPM_SET_OK): Succeeded Other: Failed due to other causes
Parameter	mod_name	Name of registered application
Description		Set the application is ready to go to DPM sleep state. DPM service won't let the whole system enter DPM sleep unless all registered application successfully has invoked this function.

int dpm_app_is_sleep_ready_set(char *mod_name)		
Return		1: Application is ready to enter DPM sleep 0: Application is not ready to enter DPM sleep
Parameter	mod_name	Name of registered application
Description		The application is ready to go to DPM sleep.

int dpm_app_sleep_ready_clear(char *mod_name)		
Return		0 (DPM_SET_OK): Succeeded Other: Failed due to other causes
Parameter	mod_name	Name of registered application
Description		Set the application is not ready to go to DPM sleep state. DPM service won't enter DPM sleep state.

int dpm_app_data_rcv_ready_set(char *mod_name)		
Return		0 (DPM_SET_OK): Succeeded Other: Failed due to other cause

int dpm_app_data_rcv_ready_set(char *mod_name)		
Parameter	mod_name	Name of registered application
Description		Set the application is ready for data transmission. If not set or the port number is not registered, DPM service drops the received data. It's the same operation with dpm_app_data_rcv_ready_set_by_port() function. In order to set the application is ready for data transmission, the function can be invoked instead of this function.

int dpm_app_data_rcv_ready_set_by_port(unsigned int port)		
Return		0 (DPM_SET_OK): Succeeded Other: Failed due to other causes
Parameter	port	Port number of registered application.
Description		Set the application is ready for data transmission. If not set or the port number is not registered, DPM service drops the received data. It's the same operation with dpm_app_data_rcv_ready_set () function. In order to set the application is ready for data transmission, the function can be invoked instead of this function.

int dpm_app_wakeup_done(char *mod_name)		
Return		0 (DPM_SET_OK): Succeeded Other: Failed due to other causes
Parameter	mod_name	Name of registered application
Description		Set the DPM service to ready to get DPM timer callback function. If not set, DPM service won't be invoked DPM timer callback function even if the DPM timer has been expired.

bool dpm_app_is_wakeup_done(char *mod_name)		
Return		1: If the DPM service is ready to get DPM timer callback function
Parameter	mod_name	Name of registered application
Description		Check whether the DPM service is ready to get DPM timer callback function or not. It will be internally waiting for 1 second if it's not ready.

3.2.2 User Data Backup and Restore in DPM Memory

The APIs listed in this session are to use DPM memory. Application may require keeping their data in DPM sleep state. DPM memory helps to keep them. The data can be restored after DPM wakeup state.

Table 3: DPM memory API List

unsigned int dpm_user_mem_alloc(char *name, void **memory_ptr, unsigned long memory_size, unsigned long wait_option)		
Return		0: Succeeded
Parameter	name	Specified name of allocated memory
	memory_ptr	Pointer to place allocated bytes
	memory_size	Number of bytes to allocate
	wait_option	Suspension option. Deprecated in FreeRTOS

unsigned int dpm_user_mem_alloc(char *name, void **memory_ptr, unsigned long memory_size, unsigned long wait_option)	
Description	Allocate bytes from user DPM memory

unsigned int dpm_user_mem_free(char *name)		
Return		0: Succeeded
Parameter	name	Specified name of allocated memory
Description		Release previously allocated memory to DPM memory

unsigned int dpm_user_mem_get(char *name, unsigned char **data)		
Return		Length of allocated bytes
Parameter	name	Specified name of allocated memory
	memory_ptr	Pointer of allocated bytes
Description		Get allocated memory from DPM memory

int dpm_user_mem_init_check(void)	
Return	1 (pdTRUE): DPM memory is ready to be accessed
Description	Get state of DPM memory can be accessed or not

3.2.3 Using DPM Timer

The APIs listed in this session are to use DPM Timer (RTC Timer). Timer who is provided by RTOS won't be available during DPM sleep state. SDK provides DPM Timer to periodically or one-time wake-up during DPM sleep state.

Table 4: DPM Timer APIs

int dpm_timer_create(char *task_name, char *timer_name, void (* callback_func)(char *timer_name), unsigned int secs, unsigned int reschedule_secs)		
Return		5 ~ 15: Assigned Timer ID.
Parameter	task_name	Name of the process that wants to use the DPM timer. It must be the same as the name registered using the dpm_app_register() function.
	timer_name	A string of 7 characters or less as a unique character to distinguish timer.
	callback_func	Function pointer to be called when timeout occurs. If there is no function you want to call, enter NULL.
	secs	Timeout occurrence time
	reschedule_secs	Periodic timeout occurrence time. When set to 0, timeout occurs only once with the time set as the value of secs,
Description		In DPM sleep state, the timer count is also activated, and when the timeout event occurs, the DA16200 (DA16600) wakes up and calls the registered callback function.

int dpm_timer_delete(char *task_name, char *timer_name)	
Return	5 ~ 15: Assigned Timer ID.

int dpm_timer_delete(char *task_name, char *timer_name)		
Parameter	task_name	Name of the process that wants to use the DPM timer. It must be the same as the name registered using the dpm_app_register() function.
	timer_name	Name of the timer to be deleted. It is the same name used when registering timer using dpm_timer_create() function.
Description		Delete the registered timer.

int dpm_timer_change(char *task_name, char *timer_name, unsigned int secs)		
Return		5 ~ 15: Assigned Timer ID.
Parameter	task_name	Name of the process that wants to use the DPM timer. It must be the same as the name registered using the dpm_app_register() function.
	timer_name	Name of the timer to be deleted. It is the same name used when registering timer using dpm_timer_create() function.
	secs	Time value to change.
Description		Change the timeout occurrence time of an already running timer.

int dpm_timer_remaining_sec_get(char *thread_name, char *timer_name)		
Return		Number of seconds remaining until timeout is returned.
Parameter	task_name	Name of the process that wants to use the DPM timer. It must be the same as the name registered using the dpm_app_register() function.
	timer_name	Name of the timer to be deleted. It is the same name used when registering timer using dpm_timer_create() function.
Description		Get the remaining timeout occurrence time of an already running timer.

3.2.4 DPM Filtering

The APIs listed in this session are to filter specific port number of TCP/UDP, or IP multicast address. In order to provide ultra-low-power consumption, the DA16200 (DA16600) wakes up by responding only to the registered TCP/UDP port number, or IP multicast address in DPM sleep state.

Table 5: DPM filtering API List

void dpm_udp_filter_enable(unsigned char en_flag)		
Parameter	en_flag	Flag to enable UDP filter functionality
Description		Enable/Disable UDP filter functionality. If enabled, DA16200(or DA16600) allows to receive UDP packet of registered port in DPM sleep state.

void dpm_udp_port_filter_set(unsigned short d_port)		
Parameter	d_port	Port number of UDP. The maximum is DPM_MAX_UDP_FILTER(8)
Description		Set port number of UDP to allow to receive UDP packet in DPM sleep

void dpm_udp_port_filter_delete(unsigned short d_port)		
Parameter	d_port	Port number of UDP
Description		Delete port number of UDP. It's set by dpm_udp_port_filter_set() function

void dpm_tcp_filter_enable(unsigned char en_flag)		
Parameter	en_flag	Flag to enable TCP filter functionality
Description		Enable TCP filter functionality. If enabled, DA16200(or DA16600) allows to receive TCP packet of registered port in DPM sleep state.

void dpm_tcp_port_filter_set(unsigned short d_port)		
Parameter	d_port	Port number of TCP. The maximum is DPM_MAX_TCP_FILTER(8).
Description		Set port number of TCP to allow to receive UDP packet in DPM sleep state.

void dpm_tcp_port_filter_delete(unsigned short d_port)		
Parameter	d_port	Port number of TCP
Description		Delete port number of TCP. It's set by dpm_tcp_port_filter_set() function.

void dpm_mc_filter_set(unsigned long mc_addr)		
Parameter	mc_addr	IP multicast address
Description		Set IP multicast address to allow receiving packet in DPM sleep state

3.3 Limitations

For those applications which are registered to DPM service, there are some restrictions.

- A “Unique” application name should be given in registering for DPM service.
- The application name should not exceed 19 bytes in dpm_app_register().
- The number of application modules which calls dpm_app_register() cannot exceed 11.
- TCP/UDP port numbers cannot be duplicated among TCP/UDP applications.

3.4 Sample Code

```
void start_user_application()
{
    ...
/* Register your application to DPM service */
    if (dpm_mode_is_enabled()) {
        status = dpm_app_register(USER_UNIQUE_NAME, 0);
    }
    ...
}
```

3.4.1 Declaring DPM Sleep Ready State

```
void user_application_1()
{
      ...
      if (dpm_mode_is_enabled()) {
                dpm_retry_cnt = 0;
dpm_set_retry :
                if (dpm_retry_cnt++ < 10) {
                        dpm_sts = dpm_app_sleep_ready_set(USER_UNIQUE_NAME);
```

```
                        switch (dpm_sts) {
                        case DPM_SET_ERR :
                                goto dpm_set_retry;
                                break;
;
                        case DPM_SET_OK :
                                break;
                        }
                }
        }

        ...

        if (dpm_mode_is_enabled()) {
                dpm_retry_cnt = 0;

dpm_clr_retry :
                if (dpm_retry_cnt++ < 10) {
                        dpm_sts = dpm_app_sleep_ready_clear(USER_UNIQUE_NAME);

                        switch (dpm_sts) {
                        case DPM_SET_ERR :
                                goto dpm_clr_retry;
                                break;

                        case DPM_SET_OK :
                                break;
                        }
                }
        }
        ...
}
```

3.5 Abnormal DPM Operation

While DA16200 (DA16600) operates in DPM sleep, it executes an abnormal DPM mode if DA16200 (DA16600) is disconnected from the home AP. If DA16200 (DA16600) wakes up via an abnormal DPM mode, DA16200 (DA16600) tries to search the home AP within a predefined period and sleeps again for a predefined time. The DA16200 (DA16600) library provides a predefined value by default, but users can modify the related parameters based on their application.

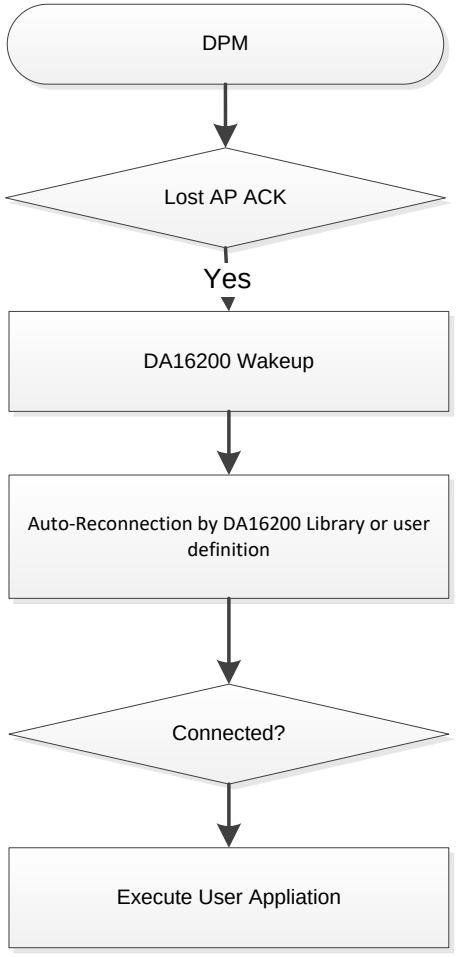

Figure 4: Abnormal DPM

3.5.1 Implementation

Table 6 gives an example enabling `__USER_DPM_ABNORM_WU_INTERVAL__` in `apps/da16200/get_started/inc/sys_common_features.h`.

Table 6: Enable the Abnormal DPM

```
apps/da16200/get_started/inc/sys_common_features.h

#define __USER_DPM_ABNORM_WU_INTERVAL__
```

Table 7 shows how to set the wake-up interval in `src/customer/main_user.c`.

Table 7: Set Wake-Up Interval

```
apps/da16200/get_started/src/user_system_feature.c

#ifdef  __USER_DPM_ABNORM_WU_INTERVAL__
/*
 * Format of dpm abnormal wakeup interval
 *      unsigned long long dpm_abnorm_wakeup_interval[10];
 *      {
 *              -1,                     // Initial value : -1
 *              10, 10, 10, 10, 10,
 *              60,
 *              3600,
 *              3600,
 *              3600 * 24
```

```
 *      }
 */

unsigned long long _user_defined_wakeup_interval[DPM_MON_RETRY_CNT] =
{
       -1,                  // Initial value : -1
       60,           // 1st Wakeup
       60,           // 2nd Wakeup : 0xdeadbeaf is no wakeup
       60,           // 3rd Wakeup : 0xdeadbeaf is no wakeup
       60 * 30,      // 4th Wakeup : 0xdeadbeaf is no wakeup
       60 * 30,      // 5th Wakeup : 0xdeadbeaf is no wakeup
       60 * 30,      // 6th Wakeup : 0xdeadbeaf is no wakeup
       60 * 60,      // 7th Wakeup : 0xdeadbeaf is no wakeup
       60 * 60,      // 8th Wakeup : 0xdeadbeaf is no wakeup
       0xDEADBEAF    // 9th Wakeup : 0xdeadbeaf is no wakeup
};

static void set_dpm_abnorm_user_wakeup_interval(void)
{
    extern unsigned long long *dpm_abnorm_user_wakeup_interval;

    dpm_abnorm_user_wakeup_interval =
                     (unsigned long long *)_user_defined_wakeup_interval;
}
#endif  /* __USER_DPM_ABNORM_WU_INTERVAL__ */
```

- The user can modify `_user_defined_wakeup_interval[10]` with the millisecond unit defined in `user_system_feature.c`
- If the parameter setting value is `0xdeafbeaf`, DA16200 (DA16600) executes the Power-off mode to not do the upcoming wake-up
- If the compile option is not defined, DA16200 (DA16600) operates based on the default setting (library). See Table 8.

Table 8: Default Value of Library

```
unsigned long long _user_defined_wakeup_interval[DPM_MON_RETRY_CNT] =
{
        -1,                     // Initial value : -1
        60,             // 1st Wakeup
        60,             // 2nd Wakeup : 0xdeadbeaf is no wakeup
        60,             // 3rd Wakeup : 0xdeadbeaf is no wakeup
        60 * 30,        // 4th Wakeup : 0xdeadbeaf is no wakeup
        60 * 30,        // 5th Wakeup : 0xdeadbeaf is no wakeup
        60 * 30,        // 6th Wakeup : 0xdeadbeaf is no wakeup
        60 * 60,        // 7th Wakeup : 0xdeadbeaf is no wakeup
        60 * 60,        // 8th Wakeup : 0xdeadbeaf is no wakeup
        0xDEADBEAF      // 9th Wakeup : 0xdeadbeaf is no wakeup
};
```

3.6 DPM Procedure in Host Interface

When initialization of the DA16200 (DA16600) and MCU is complete and then the MCU configures the DPM enabled using AT-CMD (AT+DPM=1), the DA16200 (DA16600) operates its DPM module.

If the DPM mode of DA16200 (DA16600) is enabled, the DA16200 (DA16600) is ready to enter DPM Sleep mode.

In this case, the MCU of the customer's product has two options.

- MCU power: Always ON
- MCU power: Power OFF

NOTE
This document does not describe how to use AT-CMDs and DPM setting over AT-CMD. See Ref. [1].

3.6.1 MCU Power - Always ON

In this case, the DPM operation of the DA16200 (DA16600) is simple. Since the power of MCU is always turned ON, MCU can operate in a pre-determined sequence to control the DPM operation of the DA16200 (DA16600). No additional separate operation is required.

The DPM operation of the DA16200 (DA16600) is independent of the MCU working. So, if MCU has something to do before the DA16200 (DA16600) enters DPM Sleep state, MCU must control the DPM operation (CPU Power-OFF) of the DA16200 (DA16600).

There are two instances as follows:

- Operation/Data communication: MCU to DA16200 (DA16600)
- Operation/Data communication: DA16200 (DA16600) to MCU

NOTE
In this case, a physical HW connection is required for MCU to drive the DA16200 (DA16600), which is in DPM Sleep mode. **It is necessary to hardwire a connection between MCU GPIO and the DA16200 (DA16600) External Wakeup PIN.** See Ref. [1] and [3].

3.6.1.1 MCU to DA16200 (DA16600)

The proper sequences are required for the DPM of the DA16200 (DA16600) to operate normally on a customer product, which consists of MCU and the DA16200 (DA16600).

After doing any operation, MCU has to determine the correct time for DA16200 (DA16600) to enter DPM Sleep mode (CPU Power-Off). If MCU does not send a defined AT-CMD for DPM operation of the DA16200 (DA16600), the DA16200 (DA16600) enters DPM Sleep mode without delay. If MCU has something to do itself, MCU must notify to enter the DPM Sleep mode to the DA16200 (DA16600).

```
      …
Do operation for MCU itself
      …
Drive the DA16200 External Wakeup PIN   ← Wakeup the DA16200 from DPM Sleep
Detect the DA16200 wakeup event from DPM Sleep mode: +INIT:WAKEUP,UC

AT+CLRDPMSLPEXT  ← Send AT-CMD to hold the DA16200 from entering DPM mode
      …
Operates or sends data from MCU to the DA16200
      …
```

AT+SETDPMSLPEXT ← Send AT-CMD to resume the DA16200 DPM module ...

3.6.1.2 DA16200 (DA16600) to MCU

The DA16200 (DA16600) state is in DPM Sleep mode (CPU Power-Off), but the MCU power is always ON, and the DA16200 (DA16600) operates according to the DPM standard.

There are three instances as follows:

- In case of wakeup from receiving network data
- In case of wakeup from RTC timer event
- In case of wakeup from Wi-Fi disconnect

I. Wakeup from Receiving Network Data

In this case, the received data is transferred to the MCU for data processing from the DA16200 (DA16600) over AT-CMD protocol.

When the MCU detects data reception from the DA16200 (DA16600), it must process as the following order to stably receive the data from the DA16200 (DA16600), and then let the DA16200 (DA16600) enter DPM Sleep again.

```
      …
Waiting any event from the DA16200
      …
Detect the DA16200 wakeup event from DPM Sleep mode : +INIT:WAKEUP,UC
AT+CLRDPMSLPEXT  ← Send AT-CMD to hold the DA16200 from entering DPM mode
      …
Receive the data from the DA16200 and run the operation MCU itself
      …
AT+SETDPMSLPEXT  ← Send AT-CMD to resume the DA16200 DPM module
      …
```

II. Wakeup from RTC Timer Event

In this case, the DA16200 (DA16600) automatically wakes up by the timer registered in DPM Sleep state and it sends the wakeup event to MCU. If this timer event was registered by MCU, MCU has to process the event and then proceed to the DPM Sleep procedure again.

```
      …
Waiting any action from the DA16200
      …
Detect the DA16200 wakeup event from DPM Sleep mode : +INIT:WAKEUP,RTC
AT+CLRDPMSLPEXT  ← Send AT-CMD to hold the DA16200 from entering DPM mode
      …
Process any action for this Timer event
      …
AT+SETDPMSLPEXT  ← Send AT-CMD to resume the DA16200 DPM module
      …
```

III. Wakeup from Wi-Fi Disconnect Event

In this case, the DA16200 (DA16600) wakes up by the Wi-Fi disconnection event in the DPM Sleep state and it sends the wakeup event to MCU. If any actions are needed in MCU about Wi-Fi disconnection, MCU does not need any DPM process, except this event action.

The DA16200 (DA16600) enters the DPM abnormal case handling procedure.

```
      …
Waiting any action from the DA16200
      …
Detect the DA16200 wakeup event from DPM Sleep mode : +INIT:WAKEUP,[NOBCN / DEAUTH]
AT+CLRDPMSLPEXT  ← Send AT-CMD to hold the DA16200 from entering DPM mode
      …
To do something which defined for this timer event
      …
AT+SETDPMSLPEXT  ← Send AT-CMD to resume the DA16200 DPM module
      …
```

3.6.2 MCU Power - Power OFF for Low-Power Mode

In this case, the DPM operation for the DA16200 (DA16600) is needed for additional operation to the MCU (see Section 3.6.1). Since MCU power also turned off in low-power mode, it is necessary to wake up MCU before any action.

There are two instances as follows:

- Operation/Data communication: MCU → DA16200 (DA16600)
- Operation/Data communication: DA16200 (DA16600) → MCU

NOTE
In this case, **MCU or the DA16200 (DA16600) needs to hardwire the connection to wakeup each one.** One is the DA16200 (DA16600) Wakeup PIN and the other is MCU wakeup PIN. See Ref. [1] and [3]

3.6.2.1 MCU to DA16200 (DA16600)

In the low-power standby state (In DPM Sleep of the DA16200 (DA16600) and MCU Power OFF), if MCU needs to perform any operation with the DA16200 (DA16600) after waking up first, MCU must drive the "External Wake up PIN" of the DA16200 (DA16600) and should perform any operation after detecting the DA16200 (DA16600) wakeup message.

```
      …
MCU start/wakeup by any events
Drive the DA16200 External Wakeup PIN
      …
Detect the DA16200 wakeup event from DPM Sleep mode : +INIT:WAKEUP,EXT
AT+CLRDPMSLPEXT  ← Send AT-CMD to hold the DA16200 from entering DPM mode
      …
Process any action to the DA16200
      …
AT+SETDPMSLPEXT  ← Send AT-CMD to resume the DA16200 DPM module
      …
```

3.6.2.2 DA16200 (DA16600) to MCU

In the low-power standby state (In DPM Sleep of the DA16200 (DA16600) and MCU Power OFF), if MCU needs to do some operation with the DA16200 (DA16600) after waking up first, MCU must drive the "External Wake up PIN" of the DA16200 (DA16600), then has to run the next operation after detecting the DA16200 (DA16600) wakeup message.
For descriptions on data communication, timer, and Wi-Fi disconnect, see Sections I and III.

```
      …
[DA16200] Wakeup from DPM Sleep by any events
[DA16200] Drives the "Wakeup PIN" of MCU
      …
MCU have to send the defined AT-CMD to notice "MCU Ready" : AT+MCUWUDONE
AT+CLRDPMSLPEXT  ← Send AT-CMD to hold the DA16200 from entering DPM mode
      …
Process any action for this Timer event
      …
AT+SETDPMSLPEXT  ← Send AT-CMD to resume the DA16200 DPM module
      …
```

4 DPM Manager

4.1 DPM Manager Introduction

VirtualZero™ is a synthesis of breakthrough ultra-low-power technologies, which enables extremely low power operation in the DA16200 (DA16600) SoC. VirtualZero™ shuts down every microelement of the chip that is not in use, which allows a near-zero level of power consumption when the DA16200 (DA16600) SoC does not actively transmit or receive data. Such low power operation can deliver a year or more of battery life depending on the application. VirtualZero™ also enables ultra-low-power operation to transmit and receive data when the SoC needs to be awake to exchange information with other devices. Advanced algorithms enable one to stay asleep until the exact moment required to wake up to transmit or receive. DPM (Dynamic Power Management) Manager APIs make it easy to develop a DPM application. DPM Manager is developed for users to easily develop DPM applications. It has a simple interface; all that users need to do is to write the necessary callback functions and register them to the DPM Manager, which then takes care of all the DPM relevant jobs internally.

4.2 DPM Manager Features

- Provides a callback interface for application initialization
- Supports Timer registration interface
  - Periodic timer supported. Possibility to register up to four Timer callbacks
- Session (TCP/UDP) management
  - Possibility to register up to four TCP/UDP sessions (either server/client); up to four sessions for a TCP client. One TCP Server is assigned two sessions, so the use of two TCP servers is possible
  - After registration, the User Application can transmit and receive data in a session using a callback (Rx) and a Send API
  - When a “Connect/Accept” event happens, registered callbacks are invoked (TCP only)
- Non-volatile memory space (aka DPM Memory/Retention Memory) is supported for applications in-between DPM Sleep
  - The User Application lets DPM Manager know what address and size of DPM Memory to use, and then DPM Manager takes care of saving and loading the non-volatile memory when the system enters sleep or wakes up.
- Callback on an external signal is supported
- DPM Manager control APIs for User Applications
- All User DPM Manager configuration information is written in a header file

4.3 How DPM Manager Works

- Define and create a DPM Manager configuration header: `xxx_dpm_config.h` (see Section 4.6.4)
- In user main, invoke the configuration callback and start DPM Manager
- The user implements callbacks that the DPM Manager invokes when registered
- Once the user’s job in a callback is finished, the User Application invokes the 'job done' API, and then the DPM Manager makes the system enter DPM Sleep
- Once callbacks are registered successfully on any event (timer/external wake-up/and so forth) during DPM Sleep, the registered user callbacks should be invoked when needed and the DPM Manager takes care of changing the power state of the system

4.4 DPM Manager Operation

4.4.1 DPM Manager Operation

Figure 5 shows the DPM Manager architecture and operation. DPM manager offers easy control of the DPM function between User Applications and the Sleep Daemon. To enter DPM Sleep mode, the User Application should finish all active threads. After the User Application requests DPM Sleep mode, the DPM manager saves the data of the User Application that need to be maintained during DPM Sleep mode in retention memory and controls the callback function timer of the User Application, and manages the TCP/UDP session that processes the transmission and receipt of a packet.

To repeatedly process Sleep and Wake-up mode, the DPM manager offers to maintain a Wi-Fi connection without the reconnection process. When the User Application requests DPM Sleep mode, the DPM manager commands the Sleep Daemon to enter Sleep mode. Sleep Daemon is a system thread that operates the actual DPM sleep function. When the Register and Set bits of all threads are set to 1, the System is in DPM Sleep mode.

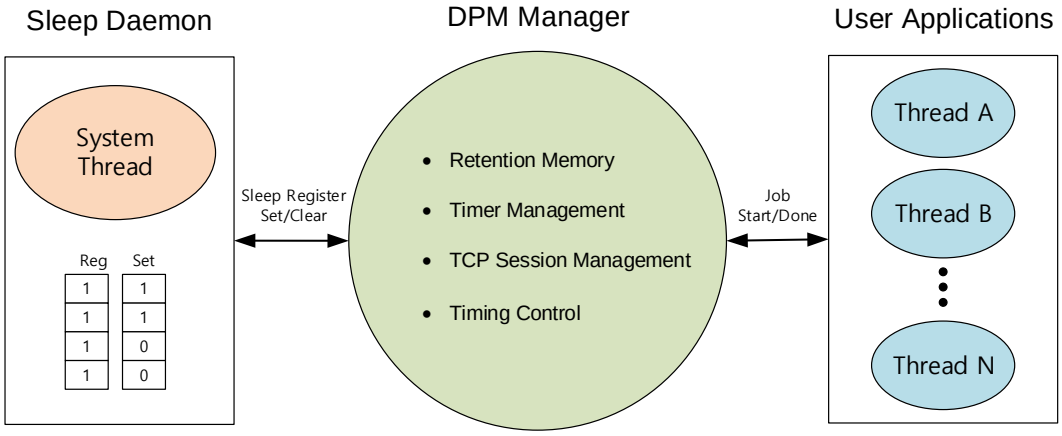

Figure 5: DPM Manager Structure

4.4.2 DPM Manager Workflow

DPM Manager works with the workflow mentioned below between System and User Application.

4.4.2.1 POR Boot

The System executes a Power on Reset (POR) boot.

4.4.2.2 DPM Configuration Request

DPM Manager requests DPM configuration information from the User Application:

- Boot initial callback function information
- Wake up initial callback function information
- Timer information, Retention Memory information
- TCP/UDP session information such as IP/port number/Callback function information
- External Wake-up callback function information

4.4.2.3 DPM Configuration Response

The User Application responds with DPM configuration information to the DPM Manager.

4.4.2.4 Retention Memory Allocation

Retention memory is allocated to save DPM configuration information.

4.4.2.5 Call Boot Initial Callback Function

DPM Manager calls the Boot Initial Callback function to boot a device for the first time.

4.4.2.6 Timer Registration and Start TCP/UDP Session Thread

DPM Manager registers Timer information and starts the TCP/UDP Session Thread.

4.4.2.7 Processing Request Event

DPM Manager calls the Callback function of the target Thread to process the request Event.

4.4.2.8 Response Job Done of Request Event

The User Application's response signal for Job done of the processing request Event in the target Thread.

4.4.2.9 Save DPM Configuration

DPM Manager saves DPM configuration information of the device in Retention memory. Go to DPM Sleep mode.

4.4.2.10 Enter DPM Sleep Mode

The system enters DPM Sleep mode.

4.4.2.11 Wake Up Device

The system wakes up a device by Timer, Unicast signal, and external wake-up sources such as the HW button.

4.4.2.12 Restore DPM Configuration

DPM Manager restores the DPM configuration from Retention memory.

4.4.2.13 Call Wake Up Initial Callback Function

DPM Manager calls the Wake-up Initial Callback function for a quick system reboot.

4.4.2.14 Start TCP/UDP Session Thread

DPM Manager starts a TCP/UDP Session Thread to operate a Network Layer.

4.4.2.15 Processing Request Event

DPM Manager calls the Callback function of the target Thread to process the request Event.

4.4.2.16 Response Job Done of Request Event

User application responds by signaling “Job Done” in the callback function on the request Event.

4.4.2.17 Save DPM Configuration

DPM Manager saves the DPM configuration of the device in Retention memory. Go to DPM Sleep mode.

4.4.2.18 Enter DPM Sleep Mode

The system enters the DPM Sleep mode.

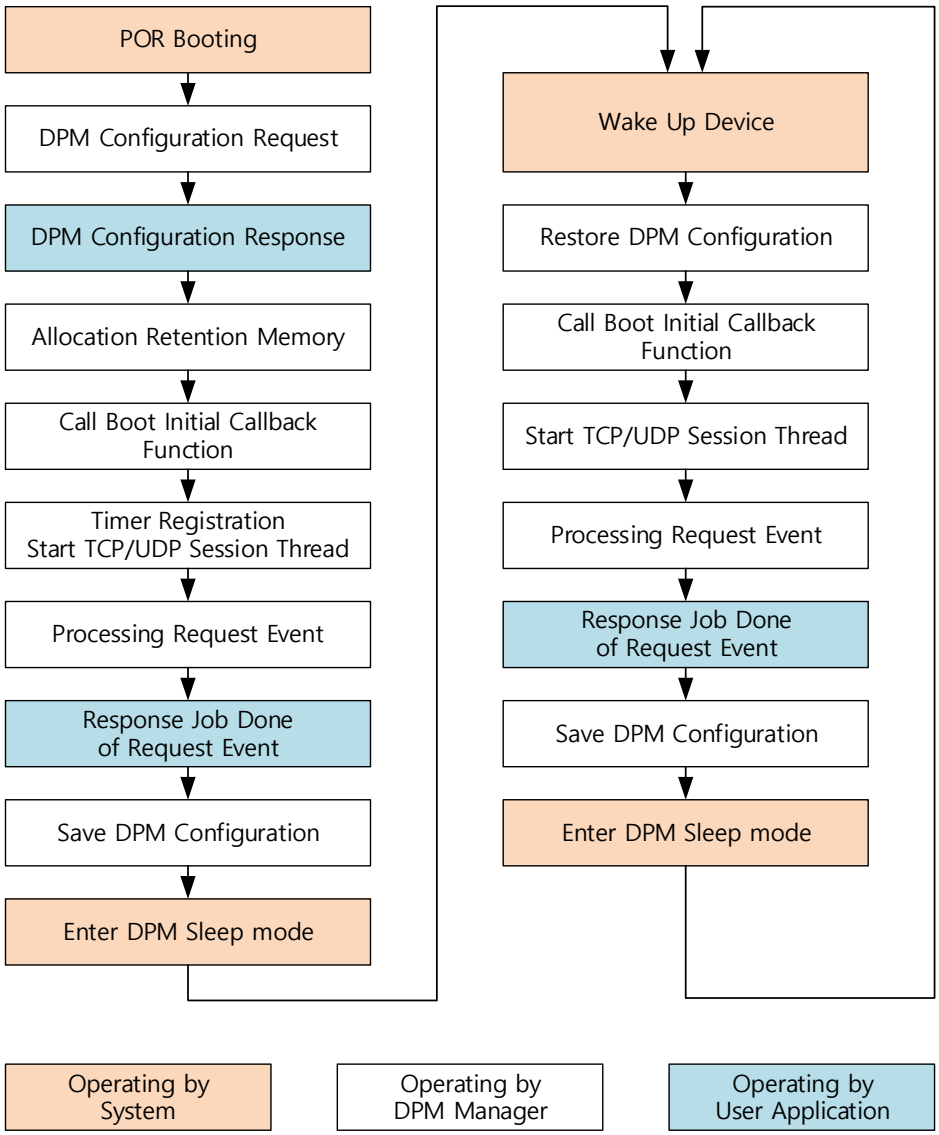

Figure 6: DPM Manager Workflow

4.4.3 The Threads of DPM Manager

DPM Manager creates three threads to process DPM Sleep mode as mentioned below.

4.4.3.1 dpmControlManager Thread

The `dpmControlManager` thread processes the following tasks:

- Get User DPM Configuration
- Allocate, restore, and save DPM configuration in Retention Memory
- Call initial function for Boot and Wake up device
- Register the Timer function and create `dpmEventManager` thread
- Create TCP/UDP Session thread
- Manage external wake up sources
- Control Timing

4.4.3.2 dpmEventManager

- Call Timer callback function
- Call external wake-up function

4.4.3.3 dpmSessionManager

The `dpmSessionManager` has four managers depending on the Network session:

- `tcpSvrSessionManager` – TCP server session manager
- `tcpCliSessionManager` – TCP client session manager
- `udpSvrSessionManager` – UDP server session manager
- `udpCliSessionManager` – UDP client session manager

The `dpmSessionManager` process has the following functions:

- Manage initial session resources such as Type, IP address, Port Number, and Socket Pool
- Call Callback function for connection and disconnection
- Call Receive Callback function
- Execute Send function

4.5 Develop an Application

4.5.1 Initialization

The user should register a callback function to initialize the configuration for the DPM manager in the User Application through the `dpm_mng_regist_config_cb()` function. See Table 10.

Table 9: Initialization

```
dpm_mng_regist_config_cb(initDpmUserConfig);
```

4.5.2 DPM Configuration

DPM Manager can define the DPM configuration of four threads.

Table 10 shows an example of the DPM configuration for a User Application.

Table 10: Configuration of DPM Manager for User Applications

```
void init_DPM_user_config (dpm_user_config_t *dpmUserConf)
```

```
{
    /* Define Boot Initial Callback and Wakeup Callback */
    dpmUserConf->bootInitCallback = BOOT_INIT_FUNC;
    dpmUserConf->wakeupInitCallback = WAKEUP_INIT_FUNC;

    /* Define Timer Registration for Thread 0 of User Application */
    dpmUserConf->timerConfig[0].timerType = TIMER1_TYPE;
    dpmUserConf->timerConfig[0].timerInterval = TIMER1_INTERVAL;
    dpmUserConf->timerConfig[0].timerCallback = TIMER1_FUNC;

    /* Define Timer Registration for Thread 1 of User Application */
    dpmUserConf->timerConfig[1].timerType = TIMER2_TYPE;
    dpmUserConf->timerConfig[1].timerInterval = TIMER2_INTERVAL;
    dpmUserConf->timerConfig[1].timerCallback = TIMER2_FUNC;

    /* Define Timer Registration for Thread 2 of User Application */
    dpmUserConf->timerConfig[2].timerType = TIMER3_TYPE;
    dpmUserConf->timerConfig[2].timerInterval = TIMER3_INTERVAL;
    dpmUserConf->timerConfig[2].timerCallback = TIMER3_FUNC;

    /* Define Timer Registration for Thread 2 of User Application */
    dpmUserConf->timerConfig[3].timerType = TIMER4_TYPE;
    dpmUserConf->timerConfig[3].timerInterval = TIMER4_INTERVAL;
    dpmUserConf->timerConfig[3].timerCallback = TIMER4_FUNC;

    /* Define Network Session Configuration for Thread 0 of User Application */
    dpmUserConf->sessionConfig[0].session_type = REGIST_SESSION_TYPE1;
    dpmUserConf->sessionConfig[0].session_myPort = REGIST_MY_PORT_1;
    memcpy(dpmUserConf->sessionConfig[0].session_serverIp, REGIST_SERVER_IP_1,
           sizeof(REGIST_SERVER_IP_1));
    dpmUserConf->sessionConfig[0].session_serverPort = REGIST_SERVER_PORT_1;
    dpmUserConf->sessionConfig[0].session_ka_interval = SESSION1_KA_INTERVAL;
    dpmUserConf->sessionConfig[0].session_connectCallback = SESSION1_CONN_FUNC;
    dpmUserConf->sessionConfig[0].session_recvCallback = SESSION1_RECV_FUNC;
    dpmUserConf->sessionConfig[0].supportSecure = SESSION1_SECURE_SETUP;
    dpmUserConf->sessionConfig[0].session_setupSecureCallback =
                                            SESSION1_SECURE_SETUP_FUNC;

    /* Define Network Session Configuration for Thread 1 of User Application */
    dpmUserConf->sessionConfig[1].session_type = REGIST_SESSION_TYPE2;
    dpmUserConf->sessionConfig[1].session_myPort = REGIST_MY_PORT_2;
    memcpy(dpmUserConf->sessionConfig[1].session_serverIp, REGIST_SERVER_IP_2,
           sizeof(REGIST_SERVER_IP_2));
    dpmUserConf->sessionConfig[1].session_serverPort = REGIST_SERVER_PORT_2;
    dpmUserConf->sessionConfig[1].session_ka_interval = SESSION2_KA_INTERVAL;
    dpmUserConf->sessionConfig[1].session_connectCallback = SESSION2_CONN_FUNC;
    dpmUserConf->sessionConfig[1].session_recvCallback = SESSION2_RECV_FUNC;
    dpmUserConf->sessionConfig[1].session_conn_retry_cnt =
                            SESSION2_CONNECT_RETRY_COUNT; /* Only TCP Client */
    dpmUserConf->sessionConfig[1].session_conn_wait_time =
                            SESSION2_CONNECT_WAIT_TIME;   /* Only TCP Client */
    dpmUserConf->sessionConfig[1].session_auto_reconn =
                            SESSION2_AUTO_RECONNECT;      /* Only TCP Client */
    dpmUserConf->sessionConfig[1].supportSecure = SESSION2_SECURE_SETUP;
    dpmUserConf->sessionConfig[1].session_setupSecureCallback =
                            SESSION2_SECURE_SETUP_FUNC;

    /* Define Network Session Configuration for Thread 2 of User Application */
    dpmUserConf->sessionConfig[2].session_type = REGIST_SESSION_TYPE3;
    dpmUserConf->sessionConfig[2].session_myPort = REGIST_MY_PORT_3;
```

```
    memcpy(dpmUserConf->sessionConfig[2].session_serverIp, REGIST_SERVER_IP_3,
           sizeof(REGIST_SERVER_IP_3));
    dpmUserConf->sessionConfig[2].session_serverPort = REGIST_SERVER_PORT_3;
    dpmUserConf->sessionConfig[2].session_ka_interval = SESSION3_KA_INTERVAL;
    dpmUserConf->sessionConfig[2].session_connectCallback = SESSION3_CONN_FUNC;
    dpmUserConf->sessionConfig[2].session_recvCallback = SESSION3_RECV_FUNC;
    dpmUserConf->sessionConfig[2].supportSecure = SESSION3_SECURE_SETUP;
    dpmUserConf->sessionConfig[2].session_setupSecureCallback =
                            SESSION3_SECURE_SETUP_FUNC;

    /* Define Network Session Configuration for Thread 3 of User Application */
    dpmUserConf->sessionConfig[3].session_type = REGIST_SESSION_TYPE4;
    dpmUserConf->sessionConfig[3].session_myPort = REGIST_MY_PORT_4;
    memcpy(dpmUserConf->sessionConfig[3].session_serverIp, REGIST_SERVER_IP_4,
           sizeof(REGIST_SERVER_IP_4));
    dpmUserConf->sessionConfig[3].session_serverPort = REGIST_SERVER_PORT_4;
    dpmUserConf->sessionConfig[3].session_ka_interval = SESSION4_KA_INTERVAL;
    dpmUserConf->sessionConfig[3].session_connectCallback = SESSION4_CONN_FUNC;
    dpmUserConf->sessionConfig[3].session_recvCallback = SESSION4_RECV_FUNC;
    dpmUserConf->sessionConfig[3].supportSecure = SESSION4_SECURE_SETUP;
    dpmUserConf->sessionConfig[3].session_setupSecureCallback =
                            SESSION4_SECURE_SETUP_FUNC;

    /* Allocation Retenstion Memory for saving and restoring DPM Configuration */
    dpmUserConf->ptrDataFromRetentionMemory = NON_VOLITALE_MEM_ADDR;
    dpmUserConf->sizeOfRetentionMemory = NON_VOLITALE_MEM_SIZE;

    /* Define Error state of Callback function */
    dpmUserConf->externWakeupCallback = EXTERN_WU_FUNCTION;
    dpmUserConf->errorCallback = ERROR_FUNCTION;
}
```

4.5.3 Callback Functions

Table 11 shows the callback function to register a timer.

Table 11: Callback Functions

```
void timer1_callback()
{
        extern  long iptolong(char *ip);
        char    txBuf[100];
        ULONG   ip;
        UINT    size;
        memset(txBuf, 0, 100);
        strcpy(txBuf, "Hellow");
        ip = (ULONG)iptolong(REGIST_SERVER_IP_3);
        size = strlen(txBuf);
        /* tcp client */
        sendToSession(SESSION3, ip, REGIST_SERVER_PORT_3, txBuf, size);
        PRINTF(" [%s] Called by timer1...\n", __func__);
}
```

4.5.4 DPM Manager API List

DPM Manager provides the APIs mentioned in Table 12 to get a callback. All functions should return 0 if done successfully.

Table 12: DPM Manager API List

API	Description
`int dpm_mng_regist_config_cb(` `  void (*regConfigFunction)())`	Ask the DPM Manager to start after a register callback function

`int dpm_mng_send_to_session(`
`  UINT sessionNo,`
`  ULONG ip,`
`  ULONG port,`
`  char *buf,`
`  UINT size)`
	Ask packet transmission

`int dpm_mng_set_session_info_my_port_no(`
`  UINT sessionNo,`
`  ULONG port)`
	Register own port number for this session (only for server)

`int dpm_mng_set_session_info_peer_port_no(`
`  UINT sessionNo,`
`  ULONG port)`
	Register peer's port number for this session (only for Server)

`int dpm_mng_set_session_info_peer_ip_addr(`
`  UINT sessionNo,`
`  char *ip)`
	Register the peer's IP address for this session (only for Server)

`int dpm_mng_set_session_info_server_ip_addr(`
`  UINT sessionNo,`
`  char *ip)`
	Register the server's IP address for this session (only for Client)

`int dpm_mng_set_session_info_server_port_no(`
`  UINT sessionNo,`
`  ULONG port)`
	Register the server's port number for this session (only for Client)

`int dpm_mng_set_session_info_local_port(`
`  UINT sessionNo,`
`  ULONG port)`
	Register own port number for this session (only for Client)

`int dpm_mng_set_session_info(`
`  UINT sessionNo,`
`  ULONG type,`
`  ULONG myPort,`
`  char *peerIp,`
`  ULONG peerPort,`
`  ULONG kaInterval,`
`  void (*connCb)(),`
`  void (*recvCb)())`
	Type: Set all the config info in one go to the session specified type
	1: TCP Server
	2: TCP Client
	3: UDP Server
	4: UDP Client
	kaInterval: in seconds

`int dpm_mng_set_DPM_timer(`
`  UINT timerId,`
`  UINT timerType,`
`  UINT interval,`
`  void (*timerCallback)())`
	timerId: from 1 to 4, up to four timers can be registered in total
	timerType: 1(periodic), 2(one-shot)
	interval: in seconds
	timerCallback: invoked when the timer is expired

`int dpm_mng_unset_DPM_timer(UINT timerId)`	timerId: from 1 to 4, up to 4 timers can be unregistered in total
`int dpm_mng_start_session(UINT sessionNo)`	Start the session
`int dpm_mng_stop_session(UINT sessionNo)`	Stop the session

`int dpm_mng_set_session_info_window_size(`
`  UINT sessionNo,`
`  UINT windowSize)`
	Register Window size to the session (only for a TCP session. Session restart (stop/start) is needed to take effect

API	Description
int dpm_mng_set_session_info_conn_retry_count(UINT sessionNo, UINT connRetryCount)	Set connection retry count for the session (only for TCP Client session)
int dpm_mng_set_Session_info_conn_wait_time(UINT sessionNo, UINT connWaitTime)	Set connection wait time for the session (only for TCP Client session) connWaitTime: in seconds
int dpm_mng_set_Session_info_auto_reconnect(UINT sessionNo, UINT autoReconnect)	You can specify the auto reconnect behavior of your session when the session is disconnected for some reason. If autoReconnect is 1, this function is activated. Only for TCP Client
int dpm_mng_save_to_RTM()	Store data in non-volatile memory. (It is also stored by the dpm_mng_job_done function.)
int dpm_mng_init_done()	Informs whether the initialization process of DPM Manager has been completed. return value (1: Done 0: Incomplete)
int dpm_mng_job_done()	A function to invoke when a job is done and to ask the DPM Manager for sleep
int dpm_mng_job_start()	A function to invoke when a job starts. When the DPM Manager gets this request, it keeps the system from entering the sleep mode. In case of a callback, this function is invoked by the DPM Manager before a callback is invoked, so the User Application does not need to call this function inside a callback

4.6 Example Code

4.6.1 Enable DPM Manager

In the SDK, enable the DPM Manager as shown in Table 13.

Table 13: Enable DPM Manager

```
[\src\application\inc\sample_features.h]

#undef   __ALL_USED_DPM_MANAGER_SAMPLE__          /* definition */
#define  __ALL_USED_LIGHT_DPM_MANAGER_SAMPLE__    /* Define to use DPM Manager */
```

Note that the recommendation is to choose the `__ALL_USED_DPM_MANAGER_SAMPLE__` or `__ALL_USED_LIGHT_DPM_MANAGER_SAMPLE__` feature depending on application requirement.

4.6.2 Start DPM Manager

The callback registration functions, `dpm_mng_regist_config_cb()` and `dpm_mng_start()` can be found in `all_used_dpm_manager_sample.c` and `all_used_light_dpm_manager_sample.c`.It will be executed depending on the definition that you enable. Table 14 gives an example of how to register the callback function and starts the DPM manager.

Table 14: Start DPM Manager

```
int user_main(void)
{
    …
    /* Initialize WLAN interface */
```

```
    wlaninit();
#ifdef  __SUPPORT_DPM_MANAGER__
dpm_mng_regist_config_cb(initDpmUserConfig);
dpm_mng_start();
#endif  /* __SUPPORT_DPM_MANAGER__ */
    …
    start_user_apps();
    return TRUE;
}
```

4.6.3 Define DPM Configuration Callback Function

Table 15: Configuration for Callback

```
void initDpmUserConfig (dpm_user_config_t *dpmUserConf)
{
    dpmUserConf->bootInitCallback = BOOT_INIT_FUNC;
    dpmUserConf->wakeupInitCallback = WAKEUP_INIT_FUNC;
    dpmUserConf->timerConfig[0].timerType = TIMER1_TYPE;
    dpmUserConf->timerConfig[0].timerInterval = TIMER1_INTERVAL;
    dpmUserConf->timerConfig[0].timerCallback = TIMER1_FUNC;
    dpmUserConf->timerConfig[1].timerType = TIMER2_TYPE;
    dpmUserConf->timerConfig[1].timerInterval = TIMER2_INTERVAL;
    dpmUserConf->timerConfig[1].timerCallback = TIMER2_FUNC;
    dpmUserConf->timerConfig[2].timerType = TIMER3_TYPE;
    dpmUserConf->timerConfig[2].timerInterval = TIMER3_INTERVAL;
    dpmUserConf->timerConfig[2].timerCallback = TIMER3_FUNC;
    dpmUserConf->timerConfig[3].timerType = TIMER4_TYPE;
    dpmUserConf->timerConfig[3].timerInterval = TIMER4_INTERVAL;
    dpmUserConf->timerConfig[3].timerCallback = TIMER4_FUNC;

    dpmUserConf->sessionConfig[0].session_type = REGIST_SESSION_TYPE1;
    dpmUserConf->sessionConfig[0].session_myPort = REGIST_MY_PORT_1;
    memcpy(dpmUserConf->sessionConfig[0].session_serverIp,
           REGIST_SERVER_IP_1, sizeof(REGIST_SERVER_IP_1));
    dpmUserConf->sessionConfig[0].session_serverPort = REGIST_SERVER_PORT_1;
    dpmUserConf->sessionConfig[0].session_ka_interval = SESSION1_KA_INTERVAL;
    dpmUserConf->sessionConfig[0].session_connectCallback = SESSION1_CONN_FUNC;
    dpmUserConf->sessionConfig[0].session_recvCallback = SESSION1_RECV_FUNC;
    dpmUserConf->sessionConfig[0].supportSecure = SESSION1_SECURE_SETUP;
    dpmUserConf->sessionConfig[0].session_setupSecureCallback =
                               SESSION1_SECURE_SETUP_FUNC;

    dpmUserConf->sessionConfig[1].session_type = REGIST_SESSION_TYPE2;
    dpmUserConf->sessionConfig[1].session_myPort = REGIST_MY_PORT_2;
    memcpy(dpmUserConf->sessionConfig[1].session_serverIp,
           REGIST_SERVER_IP_2, sizeof(REGIST_SERVER_IP_2));
    dpmUserConf->sessionConfig[1].session_serverPort = REGIST_SERVER_PORT_2;
    dpmUserConf->sessionConfig[1].session_ka_interval = SESSION2_KA_INTERVAL;
    dpmUserConf->sessionConfig[1].session_connectCallback = SESSION2_CONN_FUNC;
    dpmUserConf->sessionConfig[1].session_recvCallback = SESSION2_RECV_FUNC;
    dpmUserConf->sessionConfig[1].session_conn_retry_cnt =
                               SESSION2_CONNECT_RETRY_COUNT; /* Only TCP Client */
    dpmUserConf->sessionConfig[1].session_conn_wait_time =
                               SESSION2_CONNECT_WAIT_TIME;   /* Only TCP Client */
    dpmUserConf->sessionConfig[1].session_auto_reconn =
                               SESSION2_AUTO_RECONNECT;      /* Only TCP Client */
    dpmUserConf->sessionConfig[1].supportSecure = SESSION2_SECURE_SETUP;
    dpmUserConf->sessionConfig[1].session_setupSecureCallback =
                               SESSION2_SECURE_SETUP_FUNC;

    dpmUserConf->sessionConfig[2].session_type = REGIST_SESSION_TYPE3;
    dpmUserConf->sessionConfig[2].session_myPort = REGIST_MY_PORT_3;
    memcpy(dpmUserConf->sessionConfig[2].session_serverIp,
           REGIST_SERVER_IP_3, sizeof(REGIST_SERVER_IP_3));
    dpmUserConf->sessionConfig[2].session_serverPort = REGIST_SERVER_PORT_3;
    dpmUserConf->sessionConfig[2].session_ka_interval = SESSION3_KA_INTERVAL;
    dpmUserConf->sessionConfig[2].session_connectCallback = SESSION3_CONN_FUNC;
    dpmUserConf->sessionConfig[2].session_recvCallback = SESSION3_RECV_FUNC;
    dpmUserConf->sessionConfig[2].supportSecure = SESSION3_SECURE_SETUP;
    dpmUserConf->sessionConfig[2].session_setupSecureCallback =
```

```
                                SESSION3_SECURE_SETUP_FUNC;

    dpmUserConf->sessionConfig[3].session_type = REGIST_SESSION_TYPE4;
    dpmUserConf->sessionConfig[3].session_myPort = REGIST_MY_PORT_4;
    memcpy(dpmUserConf->sessionConfig[3].session_serverIp,
           REGIST_SERVER_IP_4, sizeof(REGIST_SERVER_IP_4));
    dpmUserConf->sessionConfig[3].session_serverPort = REGIST_SERVER_PORT_4;
    dpmUserConf->sessionConfig[3].session_ka_interval = SESSION4_KA_INTERVAL;
    dpmUserConf->sessionConfig[3].session_connectCallback = SESSION4_CONN_FUNC;
    dpmUserConf->sessionConfig[3].session_recvCallback = SESSION4_RECV_FUNC;
    dpmUserConf->sessionConfig[3].supportSecure = SESSION4_SECURE_SETUP;
    dpmUserConf->sessionConfig[3].session_setupSecureCallback =
                                SESSION4_SECURE_SETUP_FUNC;

    dpmUserConf->ptrDataFromRetentionMemory = NON_VOLITALE_MEM_ADDR;
    dpmUserConf->sizeOfRetentionMemory = NON_VOLITALE_MEM_SIZE;

    dpmUserConf->externWakeupCallback = EXTERN_WU_FUNCTION;
    dpmUserConf->errorCallback = ERROR_FUNCTION;
}
```

4.6.4 Option Definition of DPM Configuration

All these configurations are defined in the `all_used_dpm_manager_sample.c` or `all_used_light_dpm_manager_sample.c file as an example.` So, users can define these definitions in a header file named `xxx_dpm_config.h`.

Table 16: Option Definition

```
#define TIMER_TYPE_NONE                0
#define TIMER_TYPE_PERIODIC            1
#define TIMER_TYPE_ONETIME             2
#define REG_TYPE_NONE                  0
#define REG_TYPE_TCP_SERVER            1
#define REG_TYPE_TCP_CLIENT            2
#define REG_TYPE_UDP_SERVER            3
#define REG_TYPE_UDP_CLIENT            4
#define DISABLE                        0
#define ENABLE                         1

/* Boot initial callback function */
#define BOOT_INIT_FUNC                 initConfigSampleByBoot
/* Wakeup initial callback function */
#define WAKEUP_INIT_FUNC               initConfigSampleByWakeup
/* timer1 type */
#define TIMER1_TYPE                    TIMER_TYPE_PERIODIC
/* timer1 interval */
#define TIMER1_INTERVAL                10
/* timer1 callback function */
#define TIMER1_FUNC                    timer1_callback
/* timer2 type */
#define TIMER2_TYPE                    TIMER_TYPE_PERIODIC
/* timer2 interval */
#define TIMER2_INTERVAL                15
/* timer2 callback function */
#define TIMER2_FUNC                    timer2_callback
/* timer3 type */
#define TIMER3_TYPE                    TIMER_TYPE_PERIODIC
/* timer3 interval */
#define TIMER3_INTERVAL                10
```

```
/* timer3 callback function */
#define TIMER3_FUNC                     timer3_callback
/* timer4 type */
#define TIMER4_TYPE                     TIMER_TYPE_PERIODIC
/* timer4 interval */
#define TIMER4_INTERVAL                 5
/* timer4 callback function */
#define TIMER4_FUNC                     timer4_callback

/* Session Type (TCP Server) */
#define REGIST_SESSION_TYPE1            REG_TYPE_TCP_SERVER
/* My port no */
#define REGIST_MY_PORT_1                10197
/* Server ip : Client only */
#define REGIST_SERVER_IP_1              "0.0.0.0"
/* Server port : Client only */
#define REGIST_SERVER_PORT_1            0
/* Keep alive interval:TCP only, Sec */
#define SESSION1_KA_INTERVAL            0
/* Connect callback function */
#define SESSION1_CONN_FUNC              connect_callback_1
/* Receive callback function */
#define SESSION1_RECV_FUNC              recvPacket_callback_1
/* TLS enable/disable */
#define SESSION1_SECURE_SETUP           ENABLE
/* setup tls function */
#define SESSION1_SECURE_SETUP_FUNC      setup_secure_callback_1

/* Session Type (TCP Client) */
#define REGIST_SESSION_TYPE2            REG_TYPE_TCP_CLIENT
/* My port no */
#define REGIST_MY_PORT_2                0
/* Server ip : Client only */
#define REGIST_SERVER_IP_2              "192.168.0.24"
/* Server port : Client only */
#define REGIST_SERVER_PORT_2            10196
/* Keep alive interval:TCP only, Sec */
#define SESSION2_KA_INTERVAL            0
/* Connect callback function */
#define SESSION2_CONN_FUNC              connect_callback_2
/* Receive callback function */
#define SESSION2_RECV_FUNC              recvPacket_callback_2
/* connect wait time(SEC): TCP Cli Only */
#define SESSION2_CONNECT_WAIT_TIME      4
/* connect retry count : TCP Client Only */
#define SESSION2_CONNECT_RETRY_COUNT  3
/* auto reconnect : TCP Client Only */
#define SESSION2_AUTO_RECONNECT         ENABLE
/* TLS enable/disable */
#define SESSION2_SECURE_SETUP           ENABLE
/* setup tls function */
#define SESSION2_SECURE_SETUP_FUNC      setup_secure_callback_2

/* Session Type (UDP Server) */
#define REGIST_SESSION_TYPE3            REG_TYPE_UDP_SERVER
/* My port no */
#define REGIST_MY_PORT_3                10197
/* Server ip : Client only */
#define REGIST_SERVER_IP_3              "0.0.0.0"
/* Keep alive interval:TCP only, Sec */
```

```
#define REGIST_SERVER_PORT_3            0
/* Keep alive interval : TCP only */
#define SESSION3_KA_INTERVAL            0
/* Connect callback function */
#define SESSION3_CONN_FUNC              connect_callback_3
/* Receive callback function */
#define SESSION3_RECV_FUNC              recvPacket_callback_3
/* DTLS enable/disable */
#define SESSION3_SECURE_SETUP           ENABLE
/* setup tls function */
#define SESSION3_SECURE_SETUP_FUNC      setup_secure_callback_3

/* Session Type (UDP Client) */
#define REGIST_SESSION_TYPE4            REG_TYPE_UDP_CLIENT
/* My port no */
#define REGIST_MY_PORT_4                0
/* Server ip : Client only */
#define REGIST_SERVER_IP_4              "192.168.0.24"
/* Server port : Client only */
#define REGIST_SERVER_PORT_4            10196
/* Keep alive interval:TCP only, Sec */
#define SESSION4_KA_INTERVAL            0
/* Connect callback function */
#define SESSION4_CONN_FUNC              connect_callback_4
/* Receive callback function */
#define SESSION4_RECV_FUNC              recvPacket_callback_4
/* DTLS enable/disable */
#define SESSION4_SECURE_SETUP           ENABLE
/* setup tls function */
#define SESSION4_SECURE_SETUP_FUNC      setup_secure_callback_4
```

4.6.5 Define Callback Function Type

Table 17: Callback Declaration

```
void initConfigEn673ByBoot();
void initConfigEn673ByWakeup();
void timer1_callback();
void timer2_callback();
void timer3_callback();
void timer4_callback();
void connect_callback_1(void *sock, UINT conn_status);
void recvPacket_callback_1(void *sock, UCHAR *rx_buf,
                           UINT rx_len, ULONG rx_ip, ULONG rx_port);
void connect_callback_2(void *sock, UINT conn_status);
void recvPacket_callback_2(void *sock, UCHAR *rx_buf,
                           UINT rx_len, ULONG rx_ip, ULONG rx_port);
void connect_callback_3(void *sock, UINT conn_status);
void recvPacket_callback_3(void *sock, UCHAR *rx_buf, UINT rx_len,
                           ULONG rx_ip, ULONG rx_port);
void connect_callback_4(void *sock, UINT conn_status);
void recvPacket_callback_4(void *sock, UCHAR *rx_buf, UINT rx_len,
                           ULONG rx_ip, ULONG rx_port);
void connect_callback_5(void *sock, UINT conn_status);
void recvPacket_callback_5(void *sock, UCHAR *rx_buf, UINT rx_len,
                           ULONG rx_ip, ULONG rx_port);
void connect_callback_6(void *sock, UINT conn_status);
void recvPacket_callback_6(void *sock, UCHAR *rx_buf, UINT rx_len,
                           ULONG rx_ip, ULONG rx_port);
void external_wu_callback();
```

```
void error_callback(UINT error_code, char *comment);
```

5 DDPS

5.1 DDPS Introduction

DA16200 (DA16600) has a DPM Dynamic Period Setting (DDPS) function.

Access Points (AP) with Wi-Fi BSS (Basic Server Set) provides a method to configure the power saving options of each connected station device. To support a station's power saving feature, the AP must maintain the packets for that station when the station is in a power saving state. The DDPS algorithm checks the buffering time required for the DPM operation of the AP that the DA16200 (DA16600) station is connected to and calculates the optimal TIM wake up period (interval) for a DPM operation based on the APs buffer size and buffering time.

5.2 Operation Scenario

The DDPS configuration and operation have the following sequence:

1. During provisioning, DDPS is enabled when configuring the DPM mode.
2. Once DDPS is enabled, the DDPS will start triggering in DPM state and enter DPM when connecting to the AP.
3. When DDPS execution is completed the inspection cycle value is compared to the rx maximum number of beacons and to the number of inspection Probe conditions to determine if the DDPS check condition is satisfied.
    - If the DDPS check condition is satisfied, then the TIM Wakeup Interval is set to 3 seconds.
    - If the DDPS check condition is not satisfied, then the DPM interval set to 1 second
4. DDPS, DPM, and related operational services will be terminated.

5.3 Enable DDPS

When using the `setup` console command, DDPS can be enabled during the DPM configuration as shown below:

```
Dialog DPM (Dynamic Power Management) ? [Yes/No/Quit] : y

DPM factors : Defaults ? [Yes/No/Quit] : n

DDPS Enable : Default ? [No/Yes/Quit] : y

DPM Keep Alive Time(0~600000 ms) ? [Quit] (Default 30000 ms) :

DPM User Wakeup Time(0~86400 Sec.) ? [Quit] (Default 0 Sec.) :

DPM TIM Wakeup Count(1~65535 dtim) ? [Quit] (Default 10) :
==========================================
DPM MODE           : Enable
Dynamic Period Set : Enable
Keep Alive Time    : 30000 ms
User Wakeup Time   : 0 sec.
TIM Wakeup Count   : 10 dtim
==========================================
DPM CONFIGURATION CONFIRM ? [Yes/No/Quit] : y
```

For more information on the `setup` console command, see Getting Started Guide [1].

5.3.1 DPM API

The following API is called after the above setup command to save the configuration to NVRAM.

User application can call API to change DPM parameters.

unsigned char setup_apply_dpm(unsigned char dpm_mode, unsigned char dpm_Dynamic_Period_Set, int dpm_KeepAlive_time, int dpm_User_Wakeup_time, int dpm_TIM_wakeup_count)		
Parameter	dpm_mode	Enable/Disable DPM MODE
	dpm_Dynamic_Period_Set	Enable/Disable Dynamic Period Set
	dpm_KeepAlive_time	Keep Alive time (0~600000 ms) default: 30000ms Time to wake up periodically to sync with the AP
	dpm_User_Wakeup_time	User Wakeup Time (0~86400 Sec) default : 0sec This is used when the user needs to wakeup periodically.
	dpm_TIM_wakeup_count	TIM Wakeup Count (1~65535 dtim) default : 10dtim This is the interval to check the AP's beacon. It is recommended to use 30 when using DDPS.
Return		E_ERROR(254): Error
		Others (E_CONTINUE): Success

```
void easy_setup(void)
{
  ...
  /* DPM MODE */
  ret = setup_apply_dpm(e_dpm_mode,
    #ifdef __SUPPORT_DPM_DYNAMIC_PERIOD_SET__
                  e_dpm_Dynamic_Period_Set,
    #endif // __SUPPORT_DPM_DYNAMIC_PERIOD_SET__
                  e_dpm_KeepAlive_time,
                  e_dpm_User_Wakeup_time,
                  e_dpm_TIM_wakeup_count);

     switch (ret)
     {
        case E_ERROR:
          goto CMD_ERROR;
        case E_CONTINUE:
        default:
          break;
     }
    ...
     /* reboot */
     reboot_func(SYS_REBOOT);
     return;
    ...
}
```

5.4 BUFP

The buffering probe (BUFP) state starts when DDPS is enabled and RTOS goes into a sleep state. Figure 7 shows the state change during BUFP.

5.4.1 BUFP State Diagram

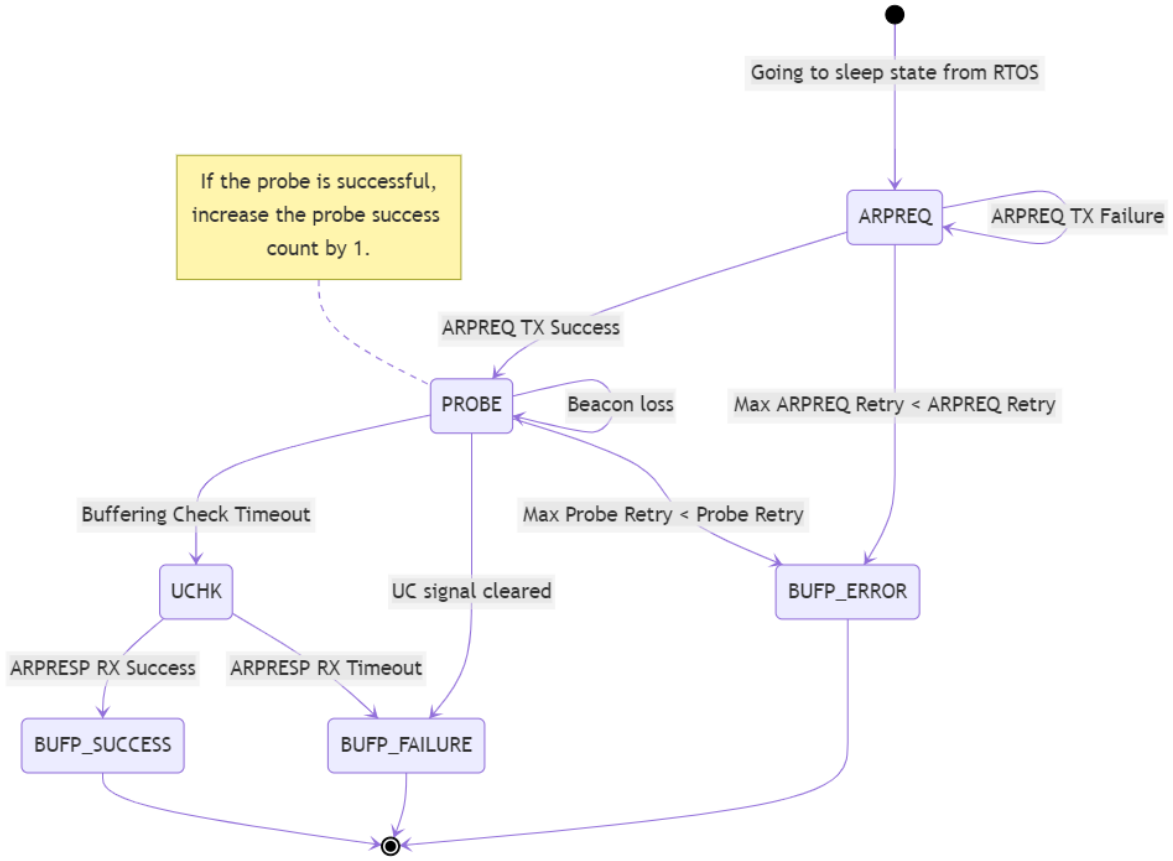

Figure 7: DDPS BUFP Block Diagram

Table 18: BUFP Main States

State	Description
ARPREQ	• State where the station transmits ARP request data to the AP • Success when ACK is received from the AP • If there is no ACK from the AP, retry ARPREQ data transmission
PROBE	• State where BUFP measures the AP UC buffering time • Success when the AP's UC signal is maintained until the probe times out • If the AP's UC signal is cleared during the probe state, the probe fails • If the beacon is not continuously received from the AP, then retry the probe again
UCHK	• State where the station waits for ARP REPLY data from the AP • Success when ARP reply data is received from the AP until the UCHK timeout
BUFP_SUCCESS	• State where BUFP was successful
BUFP_FAILURE	• State where BUFP failed • The probe failed or ARPRESP data was not received
BUFP_ERROR	• BUFP error state • When the ARPREQ retry count has reached the maximum ARPREQ frame transmission count • When the probe retry count has reached the maximum probe retry count

5.4.2 When DDPS Changes the Sleep Time to 1 Second

When the BUFP fails 4 out of 5 times, DDPS estimates that the AP's UC buffering time is less than 3 seconds and changes the sleep time to 1 second.

5.5 AP Test Report for DDPS

Table 19 shows the test results for each AP model that the DDPS functionality was tested on. Each AP was tested 100 times.

Table 19: DDPS Result

AP Model	DDPS 1s	PROBE MAX	PROBE MIN
360 F5C		5	5
360 F5S		5	4
360 P1		5	4
360 P4		5	5
360 V5S		5	5
AMPED ALLY-0091K		5	5
ANTIBANG A3		5	4
ASUS ACRH13		5	5
ASUS RT-AC1200GU	1s	0	0
ASUS RT-AC1750		5	5
ASUS RT-AC3200		5	5
ASUS RT-AC51UPLUS	1s	0	0
ASUS RT-AC5300		5	5
ASUS RT-AC58U		5	5
ASUS RT-AC66U		5	5
ASUS RT-AC87U		5	5
ASUS RT-AC88U		5	5
ASUS RT-N14UHP		5	5
ASUS TM-AC1900		5	5
BELKIN F7D6301		5	4
BELKIN F9K1002		5	3
BUFFALO WHR-300HP2D		4	2
BUFFALO WSR-1166DHP3		5	5
BUFFALO WSR-2533DHPL		5	4
CISCO RV110W-ECN		5	5
DLINK 605L		5	5
DLINK 616		5	5
DLINK 619L		5	5
DLINK 822		5	5

AP Model	DDPS 1s	PROBE MAX	PROBE MIN
DLINK DIR-806A		5	5
DLINK DIR-820L		5	4
DLINK DIR-822P		5	5
DLINK DIR-823PRO		5	4
DLINK DIR-828		5	5
DLINK DIR-842		5	5
DLINK DIR850LW		5	4
DLINK DIR-880L		5	5
DLINK DIR-890L		5	5
ELECOM WRC-1167GEBKS		5	5
EZVIZ CS-X3C-8E		5	5
FASTCOM FAC1200R		5	4
FASTCOM FAC2100R	1s	0	0
FASTCOM FW313R		5	4
FASTCOM FW450R		5	5
FASTCOM FWR200		5	4
H3CMAGIC R100		5	5
H3CMAGIC R300		5	5
HIWIFI E30		5	4
HIWIFI HC5861B		5	4
HUAWEI GLORY-ROUTINGPRO		5	5
HUAWEI HONOR-X2		5	5
HUAWEI WS5100		5	5
HUAWEI WS5102		5	5
HUAWEI WS5200		5	5
HUAWEI WS550		5	4
HUAWEI WS832		5	5
HUAWEI WS851		5	5
HUMAX QUANTUM-T3Av2		5	5
HUMAX T10X		5	4
IODATA WNAC583R	1s	0	0
IODATA WNAC733GR	1s	0	0
IODATA WNAX1167	1s	0	0
IODATA WNPR2600G		5	5
IPTIME A1004	1s	0	0
IPTIME A2004NSR		5	4
IPTIME A300NS-BCM		5	5
IPTIME A7004M		5	4
IPTIME A3004NS-BCM		5	5

AP Model	DDPS 1s	PROBE MAX	PROBE MIN
IPTIME A3004NS-BCM		5	5
IPTIME A8004ITL		5	4
IPTIME A804NS		5	4
IPTIME N604		5	4
IPTIME A604R		5	5
IPTIME N702BCM		5	5
IPTIME N704BCM		5	4
IPTIME N804V		5	5
LBLINK BL-AC1200D		5	4
LBLINK WR9000		5	4
LBLINK WR4000		5	4
LINKSYS E1200		5	4
LINKSYS EA6900		5	5
LINKSYS EA7500		5	5
LINKSYS EA8300		5	5
LINKSYS WRT1900AC		5	5
LINKSYS WRT300N		5	5
LINKSYS WRT3200ACM		5	4
LINKSYS WRT54GL		5	5
MERCURY C12G	1s	0	0
MERCURY D196G		5	5
MERCURY D19G		5	4
MERCURY D26GPro		5	5
MERCURY MW300R		5	4
MERCURY MW313R		5	4
MERCURY MW316R		5	5
MIKROTIK RB751U-2H	1s	2	0
MOTOROLA MR1900		5	1
MERCURY RUSH-1537N		5	5
NETCORE 360_P2		5	5
NETGEAR JWNR2000v2		5	5
NETGEAR ORBI		5	5
NETGEAR R6120		5	4
NETGEAR R6220		5	3
NETGEAR R7000		5	4
NETGEAR R8000		5	4
NETGEAR RAX120		5	5
NETGEAR RAX40		5	4
NETGEAR RAX80		5	5

AP Model	DDPS 1s	PROBE MAX	PROBE MIN
NETGEAR WNDR3400v3		5	1
NETGEAR X10		5	5
NETIS M3200N		5	5
NETIS MF1200AC		5	5
NETIS WF2770	1s	0	0
NETIS WF2785		5	5
NETIS WF302		5	4
NEXT 504N		5	5
NEXT 7004N		5	5
NEXT 8004N		5	4
PHICOMM PSG1218	1s	0	0
PIXLINK WR07		5	4
SAMSUNG SWW3100BG		5	3
SAMSUNG SWW-3400RW		5	5
SAMSUNG ET-WV525		5	5
SEMA SAP-H310SR	1s	1	0
SYNOLOGY MR2200AC		5	5
SYNOLOGY RT2600AC		5	5
TENDA AC15		5	5
TENDA FH304		5	5
TENDA N318		5	3
TOTOLINK A2500R		5	5
TOTOLINK A3100R		5	5
TOTOLINK A780R		5	4
TOTOLINK A800R		5	5
TOTOLINK A850R		5	5
TOTOLINK N350RP		5	5
TOTOLINK N600R		5	5
TPLINK AD7200		5	4
TPLINK ARCHER-AX10		5	4
TPLINK ARCHER-C2600		5	5
TPLINK TL-WAR1200L		5	5
TPLINK TL-WDR8610		5	5
TPLINK TL-WDR8690		5	5
TPLINK WDR5600		5	4
TPLINK WDR5660		5	5
TPLINK WDR6500		5	5
TPLINK WDR7660		5	4
TPLINK WR2041		5	5

AP Model	DDPS 1s	PROBE MAX	PROBE MIN
TPLINK WR842N		5	4
TPLINK WR880N		5	5
TPLINK WR940N		5	5
TRENDNET TEW-812DRU		5	1
TRENDNET TEW-827DRU		5	4
UNICORN AW		5	4
UTT A310	1s	0	0
UTT A655W	1s	0	0
UTT A755W		5	4
VOLANS G1		5	5
WAVLINK A33		0	0
WAVLINK N300		5	4
WAVLINK WN521N2A		0	0
WEVO 11AC-NASROUTER		5	4
WEVO HI1200AC		5	3
XIAOMI DVB4218CN		5	4
XIAOMI MIWIFI3	1s	1	0
XIAOMI MIWIFIPRO		5	5
XIAOMI R1CM	1s	0	0
XIAOMI R3AC		5	5
ZIO 2520N		5	5
ZIO 5500AC		5	5
ZIO FREEZIO		5	5

Revision History

Revision	Date	Description
1.4	12-Jan-2023	Merged files listed below and added descriptions about DPM APIs ● UM-WI-005_DA16200_DA16600_DPM_Manager ● UM-WI-034_DA16200_DA16600_DPM_Over_AT-CMD
1.3	27-Sep-2022	Update DPM API
1.2	28-Mar-2022	Update logo, disclaimer, copyright.
1.1	25-Nov-2021	Title was changed.
1.0	29-Oct-2020	Initial version.

Status Definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

RoHS Compliance

Renesas Electronics' suppliers certify that its products are in compliance with the requirements of Directive 2011/65/EU of the European Parliament on the restriction of the use of certain hazardous substances in electrical and electronic equipment. RoHS certificates from our suppliers are available on request.

Important Notice and Disclaimer

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers skilled in the art designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only for development of an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising out of your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu

Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:

https://www.renesas.com/contact/

Trademarks