User Manual

DA16200 DA16600 PTIM Programmer Guide

UM-WI-019

Abstract

DA16200 (DA16600) is a highly integrated ultra-low power Wi-Fi system on a chip (SoC). It allows users to develop a Wi-Fi solution on a single chip. This SDK document guides developers in programming, using the DA16200 (DA16600) chipset. It describes the SDK API and peripheral device drivers and interfaces.

Contents

 © 2022 Renesas Electronics

Figures

Tables

© 2022 Renesas Electronics

1 Terms and Definitions

ACK	Acknowledgement
AP	Access Point
ARP	Address Resolution Protocol
ARP-REQ	ARP Request
ARP-REPLY	ARP Reply
AUTO-ARP	Auto Address Resolution Protocol
BC	Broadcast Data
BCN	Beacon
BCMC	Broadcast/Multicast Data
CLK	Clock
DEAUTH	Deauthentication
DEAUTH CHK	Deauthentication Check
DPM	Dynamic Power Management of Dialog
EXT-SIGNAL	External Signal
HW	Hardware
KA	Keep Alive
MAC	Media Access Control
MC	Multicast Data
NO ACK	No Acknowledgement
PERI	Peripheral
PSDK	PTIM SDK
PS-POLL	Power Save Poll
PTIM	Programable TIM Application
POR-BOOT	Power-On Reset Boot
POR/FULL	POR BOOT or FULL BOOT
RA	RAM Library
RTC	Real-Time Clock
RTM	Retention Memory
RTOS	Real-Time Operating System
RX	Receive
SFLASH	Serial Flash
STA	Station
TCP	Transmission Control Protocol
TIM	Traffic Information MAP
TIMP	PTIM Performance
TU	Time Unit. 1 TU = 1.024ms
UC	Unicast Data
UDPH	UDP Hole-punch
UDP	User Datagram Protocol
XTAL	Crystal

© 2022 Renesas Electronics

2 References

[1] DA16200, Datasheet, Dialog Semiconductor
[2] DA16200, EVK User Manual, User Manual, Dialog Semiconductor
[3] DA16200, ThreadX Example Application Manual, User Manual, Dialog Semiconductor
[4] DA16200 DA16600, SDK Programmer Guide, User Manual, Dialog Semiconductor

 © 2022 Renesas Electronics

3 Introduction

DA16200 (DA16600) is a highly integrated, ultra-low power Wi-Fi system on chip (SoC). It enables users to develop a Wi-Fi solution on a single chip. To support the ultra-low power of the DA16200 (DA16600), an ultralight application called PTIM is supported. You can implement PTIM using PSDK, which supports initialization, execution, going to sleep, going to full boot, the customized UDP data transmission, parsing RX data and function configurations. This document describes PSDK and shows examples of code.

Initialization

It initializes the PTIM application and the MAC HW.

Execution

It runs automatically PTIM functions to maintain the connection with AP. The PTIM functions to be executed can be set during PTIM initialization or in an application after POR/FULL-boot.

Going to sleep

All PTIM functions stop and PTIM goes to the sleep state and all HW powers off except RTC block, RTM and XTAL 32K.

Going to a full boot

If running a full-boot application is required to process some tasks that PTIM cannot handle, such as processing TCP data packet, and so forth, PTIM should stop itself and do a full-boot in order to run a full-boot application.

The customized UDP data transmission

If necessary, the user can transmit the user defined UDP data to the target peer.

Parsing RX data

If necessary, the user can parse the received data for verification.

3.1 DPM

DA16200 (DA16600) supports Sleep mode 1 using the maximum low power, the Sleep mode 2 waking up by ext-signal or RTC timer, and the Sleep mode 3 maintaining the connection with AP, in DPM. In particular, Sleep mode 3 can power-off DA16200 (DA16600), except for RTC and RTM, while maintaining the connection to AP. PSDK supports Sleep mode 3 operation. This document mainly explains Sleep mode 3.

3.2 DPM Boot

3.2.1 DPM Wake-Up Type

DA16200 (DA16600) has several sources to wake up from the SLEEP state. These are described as follows:

3.2.1.1 POR-BOOT

This occurs only once when DA16200 (DA16600) is powered on. The bootloader in DA16200 (DA16600) will launch the RTOS image, which is built from Dialog's SDK when it is compiled. The built RTOS image can be called as POR-BOOT or FULL-BOOT according to its boot source. It is called as POR-BOOT when it boots from POR (Power-On Reset) while called as FULL-BOOT when it boots from the sleep state. Both POR-Boot and FULL-Boot applications are same. The terms simply indicate the kind of source the RTOS image runs from. The terms, POR/FULL application is used in this document.

 © 2022 Renesas Electronics

3.2.1.2 EXT-SIGNAL

DA16200 (DA16600) can wake up from the SLEEP state when an EXT-SIGNAL occurs by the user or device. In this case, the boot loader launches FULL-BOOT application. An external signal can come from one of two dedicated pins, RTC_WAKEUP, RTC_WAKEUP2, and GPIOs in DA16200 (DA16600).

3.2.1.3 RTC TIMER WAKE-UP

DA16200 (DA16600) has an internal 32 kHz RTC clock that operates during the sleep state, and it can wake up from the SLEEP state at the scheduled time. An RTC scheduler in low level driver is designed to wake DA16200 (DA16600) up from sleep state after pre-defined time. RTC TIMER WAKEUP is also known as the counter wake-up.

3.2.2 DPM Application

There are two applications required for DPM in DA16200 (DA16600): the POR/FULL application as described before, and PTIM application.

3.2.2.1 PTIM Application

Programable TIM Application that supports limited features. After a POR/FULL application makes a new connection with an AP, it can sleep, but PTIM wakes up periodically to maintain the connection. Its main purpose is to process simple tasks and maintain the connection with the AP to which a POR/FULL application makes a connection. It can be implemented by PSDK, it is especially useful in Sleep mode 3.

3.3 RTM

PTIM image is stored in the RTM, which can retain the image while the power is off. Application data that must be kept in the POWER-OFF state must also be written to the RTM. Figure 1 shows the RTM allocation.

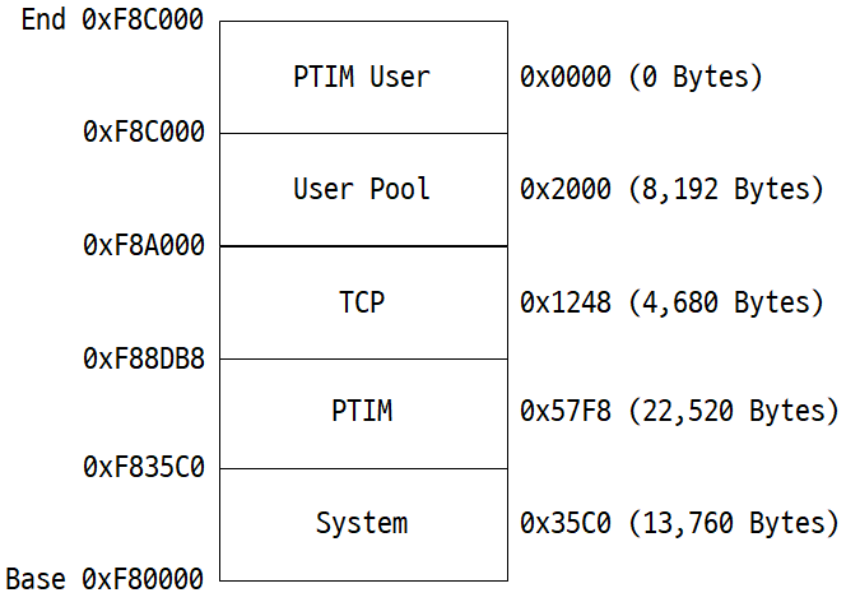


Figure 1: RTM Allocation

© 2022 Renesas Electronics

Table 1: RTM Allocation Table

Area	Offset	Size	Description
System	0xF80000	13,768 Bytes	This is reserved for the system. Do not change anything in this field.
PTIM	0xF835C0	22,520 Bytes	It is PTIM image region. Do not change the base address. The size of PTIM area can be changed by changing FC9K_RTM_PTIM_SIZE in da16x_system.h.
TCP	0xF88DB8	4,680 Bytes	It is the region for TCP session information.
User Pool	0xF8A000	8,192 Bytes	This area is used for applications in POR/FULL application. If applications in POR/FULL application does not need any retained data during power-off, this area can be removed.
PTIM User	0xF8C000	0 Bytes	This area is for PTIM application. Currently this area is not used. If you want some retained data which can be used in PTIM, you can use this area. The total size of User Pool and PTIM user area should not be over 8 kB. The default size is 0. If necessary, change the size by changing FC9K_RTM_PTIM_USER_SIZE in da16x_system.h.

3.4 DPM States

The following diagram illustrates the DPM state machine.

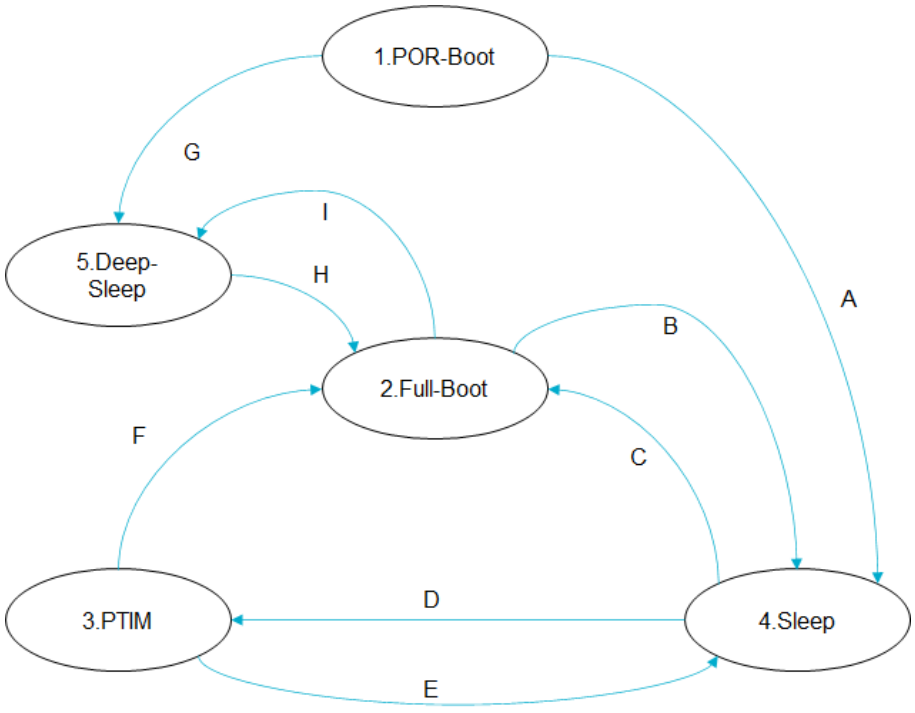


Figure 2: DPM State Machine

© 2022 Renesas Electronics

Table 2: Description of What Numbers Mean in the DPM State Machine Diagram

Number	Description
1 and 2	As said before, both POR-boot and full-boot are running POR/FULL application states. All features of DA16200 (DA16600) can be supported in these states. However, POR-BOOT means the running state when the station is powered on. On the other hand, FULL-BOOT means the running state when the station wakes up from its sleep state.
3	The PTIM is the lightweight application to support low power.
4	All HW blocks except for RTC block, RTM and 32 K XTAL are powered off. PTIM maintains the connection with the AP during the SLEEP state. This is called the Sleep mode 3 in DPM.
5	DEEP SLEEP is a long sleep due to abnormal communication with AP. This state can't maintain the connectivity with AP.

Table 3: Description of What Letters Mean in the DPM State Machine Diagram

Number	Description
A	The station is connected to AP.
B	The station remains connected to AP.
C	An ext-signal has been input or, A time set by RTC timer in POR/FULL application has elapsed.
D	A time set by RTC timer in POR/FULL application or PTIM has elapsed.
E	PTIM has completed its task and goes back to sleep.
F	PTIM should go to a full-boot due to one of the following reasons (F1 ~F6).
F1	PTIM received some data which should be handled after a FULL-BOOT.
F2	The content in beacon has changed.
F3	The station lost AP.
F4	AP does not give ACK.
F5	The AP dissociated the station.
F6	The function of PTIM had an error.
G	POR-boot application after POR-boot should go to DEEP-SLEEP due to one of the following reasons (G1 and G2).
G1	The station could not connect an AP.
G2	The station could not communicate with a server.
H	An ext-signal is input, or a time set by RTC timer in full-boot application has elapsed.
I	Full-boot application after a FULL-BOOT should go to DEEP-SLEEP due to one of the following reasons (I1 and I2).
I1	The station could not connect to an AP.
I2	The station could not communicate with a server.

3.5 PTIM

PTIM is a light-weight application whose main mission is to maintain the connection to an AP during POWER-OFF state of DA16200 (DA16600). It wakes up periodically and check the beacon from the AP and process some simple tasks which will be described in later sections. RTC and RTM should be used for PTIM operations so that PTIM runs in Sleep mode 3 of DA16200 (DA16600).

© 2022 Renesas Electronics

PTIM has the following features:

- PTIM keeps the connection with AP during sleep in Sleep mode 3
- PTIM keeps the server connection of MAC/UDP/TCP protocols
- PTIM's boot time is faster than a full boot because the image size is small and the image is loaded from RTM, not Sflash
- PTIM is designed to consume the power as less as possible
- PTIM is a simple application that users can rebuild
- If necessary, Peripherals such as GPIO, ADC, and UART and so on can be used in PTIM

3.5.1 PTIM Status

When the user runs PTIM by ptim_execute() API, it returns its result after running. PTIM goes to SLEEP or FULL-BOOT depending on this return value. PTIM goes to FULL_BOOT after getting the return value shown in Table 4 except for FAST_FROM for which PTIM goes to SLEEP. If necessary, the user can change the PTIM status.

Table 4: PTIM Number, Status, and Description Overview

Number	Status	Description
UC	0x00000001	There is UC data to be processed by FULL-BOOT.
BCMC	0x00000002	There is broadcast/multicast data to be processed by FULL-BOOT.
BCN CHANGED	0x00000004	The beacon has changed from previous beacon.
NO BCN	0x00000008	PTIM couldn't receive the beacon of the connected AP.
FROM FAST	0x00000010	The last state was PTIM.
NO ACK	0x00000020	PTIM transmits data, but No ACK for it from AP.
DEAUTH	0x00000800	The station was disassociated from AP.
FROM FULL	0x00002000	The last state was FULL-BOOT.

3.5.2 PTIM Functions

Supported functions by PTIM to keep the connection with an AP and process data from the AP are described in the following sections. Take note that most APs are verified for its functional compatibility with the PTIM Dialog designed. However, there may be some errant APs that may cause some functional problems with PTIM.

3.5.2.1 TIM

There is TIM field in the beacon frame indicting UC or BC/MC data buffered in AP. PTIM application receives a beacon frame and parses it to determine if there were UC or BC/MC data to receive. When there is UC data, PTIM sends a PS_POLL to receive it from AP.

3.5.2.2 BCMC

BCMC of PTIM is the function that receives BC/MC data buffered by the AP. The maximum number of BC/MC packets can be set by bcmc_max configuration. The maximum waiting time for receiving it can be set by bcmc_timeout_tu configuration.

3.5.2.3 PS-POLL

When an UC packet is buffered in AP (indicated by TIM field in a beacon frame already described), a PS_POLL packet is transmitted by PTIM to receive it.

© 2022 Renesas Electronics

3.5.2.4 AUTO ARP (Autonomous ARP)

Auto ARP (Autonomous ARP) of PTIM is a function that transmits autonomously ARP-REPLY to AP. Some APs transmit a broadcast packet called ARP request to the connected stations to check if they are still there; BC/MC data are transmitted at a fixed time called DTIM from AP. Normal stations in POWER SAVING state wakes up periodically at DTIM time to receive BC/MC data from AP. However, in DPM, DA16200 (DA16600) may not receive the BC/MC data at DTIM time because it could remain in the SLEEP state at that time. To emulate that DA16200 (DA16600) has received it, PTIM sends a ARP reply packet to AP to maintain the connection.

3.5.2.5 ARP-REQ

ARP-REQ of PTIM is a function that transmits autonomously ARP-REQUEST to AP. For some reason, AP transmits a deauthentication or disassociation frame to a connected station when AP wants to disconnect the stations. However, some APs does not send any packet even when they want to disconnect the stations. To confirm the connection, PTIM sends a frame called ARP request to the AP. When the AP is still there, it will respond to this frame with an ARP reply. In this way, PTIM can confirm if it is still connected to the AP.

3.5.2.6 ARP-REPLY

ARP-REPLY of PTIM is a function that transmits ARP-REPLY frame to a peer that has sent an ARP-REQUEST frame. PTIM does not need to wake up to FULL-BOOT to transmit the ARP-REPLY frame.

3.5.2.7 UDPH (UDP Hole-Punch)

UDPH of PTIM is a function that transmits autonomously a UDP packet called UDP hole punch (only UDP header without payload) or a user defined UDP packet to keep the UDP port in AP or router. If the UDP protocol is used instead of UDP hole-punch, some APs or routers may release the UDP port causing DA16200 (DA16600) to disconnect from the UDP server.

3.5.2.8 TCP ACK

TCP ACK of PTIM is a function that transmits autonomously TCP ACK frame to keep the TCP port. If the TCP protocol is used instead of TCP ACK, some APs or routers may release the TCP port causing DA16200 (DA16600) to disconnect from theTCP server.

3.5.2.9 DeauthChk (Deauthentication Check)

DeauthChk function is for waiting for the deauthentication or disassociation frame from AP. When an AP wants to disconnect the stations, it normally transmits a deauthentication or disassociation frame to the stations.
Deauthentication check is done in DA16200 (DA16600) as follows:

- Sends a KA packet to the AP
- Check if a deauthentication or disassociation packet is transmitted from the AP

DeauthChk waits for this deathentication or disassociation frame for up to the configured time duration called chk_timeout_tu. See ptim_set_deauthchk API section in this document for more details.

 © 2022 Renesas Electronics

4 Build

The user can implement applications using the DA16200 (DA16600) SDK. The IAR Embedded Workbench IDE of IAR Systems is used for the compile environment.

The DA16200 (DA16600) SDK has eight folders.

1. apps: There are 3 folders under apps folder for each project. The project includes IAR project files, applications and examples for customer.
2. core: source codes.
3. core_tim: source codes for PTIM.
4. doc: user documents (user guides, programmer guides, and so on).
5. library: to which the pre-compiled lib files (.a) are saved.
6. tools: Build scripts, temporary build artifacts, or environment files.
7. version: version files to include when Image created.
8. util: utilities for customer.

The DA16200 (DA16600) SDK may be supplied with different features for each customer or certain applications, so the source code configuration and the system libraries can differ according to the customer/application requirements. Dialog pre-compiles the system libraries with the relevant features enabled before the SDK is packaged.

NOTE
Not all features are freely enabled/disabled. This depends on the pre-compiled libraries included in the SDK package. Contact Dialog for more details.

A typical example of an IAR project for the DA16200 (DA16600) PTIM SDK is shown in Figure 3. The possibility exists to add new user application files to the existing `slib_tim` project. There is also a possibility to create one's own group.

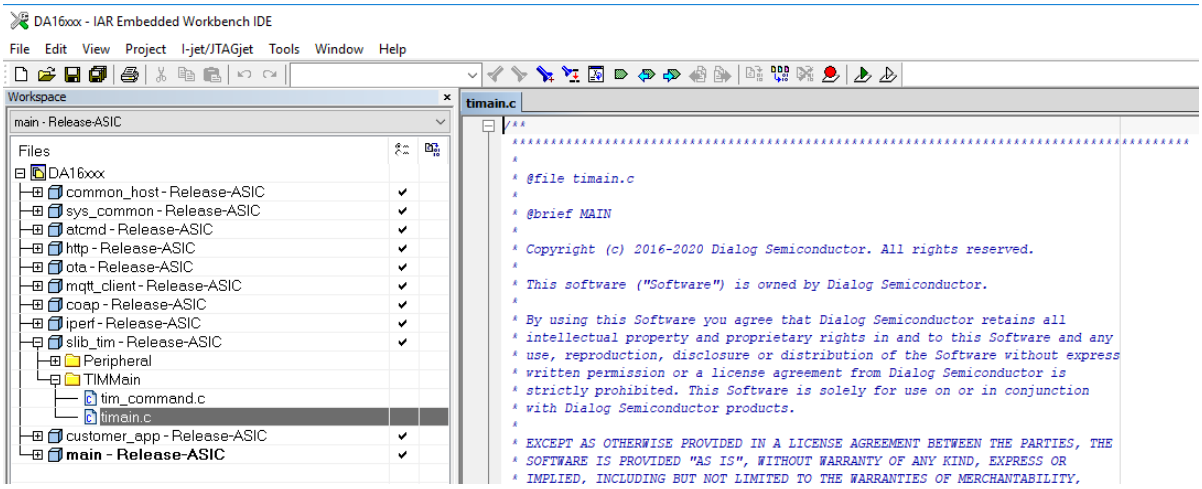


Figure 3: IAR Embedded Workbench Project Configuration

4.1 SDK Compilation

After an application is written, right-click on the project name in the IAR Embedded Workbench workspace and run command `Make` or `Rebuild All`. If you compile for the first time, then the advice is to run command `Clean` first (see Figure 4 and Figure 5).

© 2022 Renesas Electronics

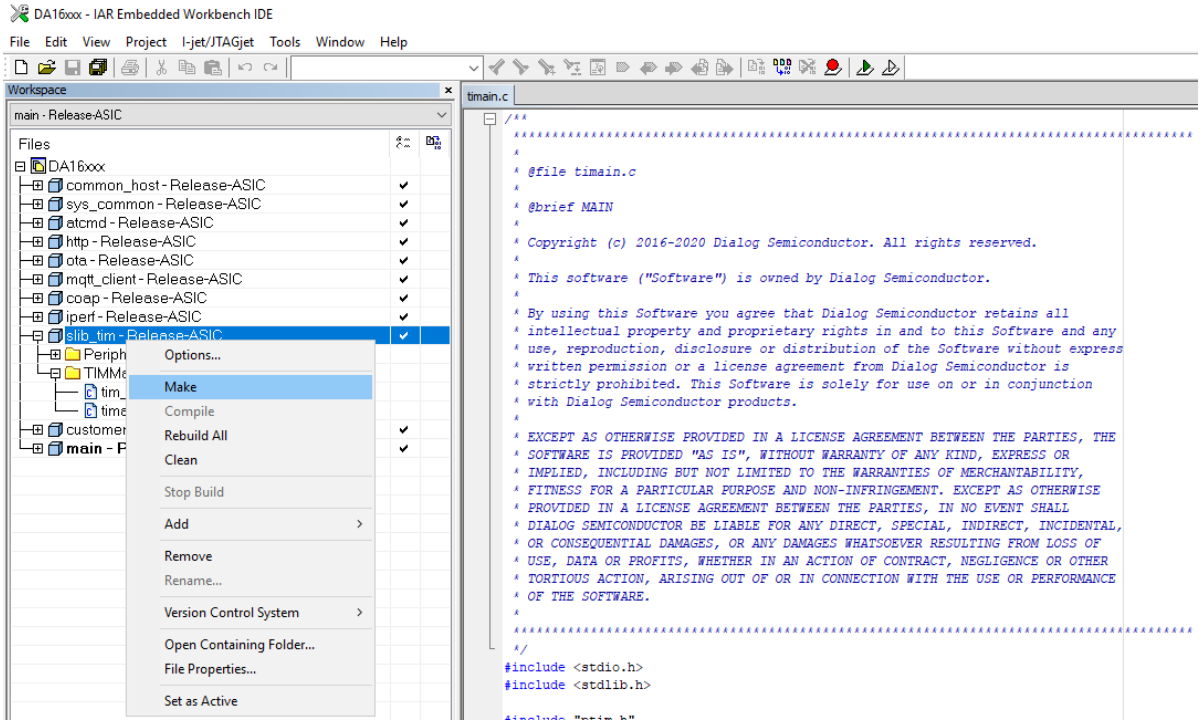


Figure 4: Run SDK on IAR Embedded Workbench

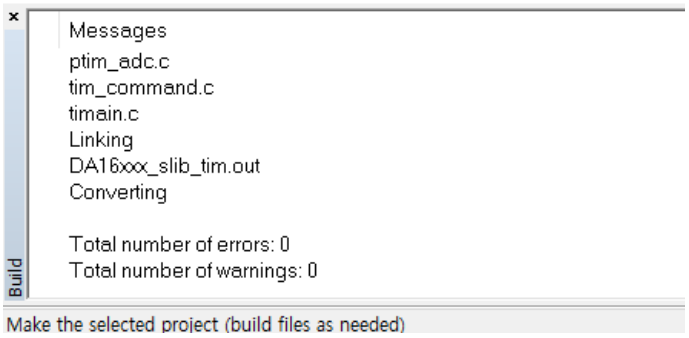


Figure 5: Successful Build on IAR Embedded Workbench

If the build is successful, a main image is created in folder "`\img`". The image file name may look like the following:

- `DA16200_RTOS_GEN01-01-XXXX-000000.img` (RTOS image [also known as main image])
- `DA16200_SLIB_GEN01-01-XXXX-000000.img` (System Library image [PTIM Image])
- `DA16200_BOOT_GEN01-01-XXXX-000000_IS25WP016D.img` (2nd bootloader image)
  - Download an image using sflash type. provided by Dialog

For more information about the firmware download, see document DA16200 (DA16600) EVK User Manual Ref. [2].

© 2022 Renesas Electronics

5 PSDK

PSDK is the PTIM software development kit. With PSDK, users can implement their own TIM application with their own requirements. This section describes APIs to implement PTIM and some example codes with it.

5.1 PSDK API

5.1.1 PSDK API Introduction

There are largely 3 kinds of APIs – control, configuration, and helper APIs. Control APIs are for initialization, execution, going to SLEEP, going to a FULL-BOOT, and data processing. Configuration APIs are for configurations for maximum timeout, counts and so on. Details are described in the below sections.

5.1.2 Application Programming Interface

5.1.2.1 PSDK DEFINITIONS

#define FC9K_RTM_USER_SIZE (0x2000)
This is the size of User Pool area as described in RTM section. The maximum size is 8 K. The size of (FC9K_RTM_USER_SIZE + FC9K_RTM_PTIM_USER_SIZE) should be less than 8 K.

#define FC9K_RTM_PTIM_USER_SIZE (0x0000)
This is the size of PTIM User area as described in RTM section. The maximum size is 8 K. The size of (FC9K_RTM_USER_SIZE + FC9K_RTM_PTIM_USER_SIZE) should be less than 8 K.

5.1.2.2 PSDK CONTROL API

int ptim_initialize(void)		
Parameter	void	Nothing
Return		If successful return DPM_SUCCESS, if failed return non-zero.
It initializes MAC HW and MAC SW related to PTIM.		

int ptim_execute(unsigned long *status)		
Parameter	status	Pointer to the PTIM status.
Return		If successful return DPM_SUCCESS, if failed return non-zero.
It executes PTIM functions. It contains receiving Beacon from AP and transmitting packet(s) based on the configuration in the RTM to which APIs for configuration set.		

int ptim_goto_fullboot(int force, unsigned long addr)		
Parameter	force	If force is 0, addr is ignored.
	addr	The user defined boot address. This is valid when force is non-zero.
Return		If successful return DPM_SUCCESS, if failed return non-zero.

© 2022 Renesas Electronics

This API makes PTIM stop and go to a FULL-BOOT to run full-boot application for further works.

int ptim_goto_sleep(int sleep_ms)		
Parameter	sleep_ms	The next sleep time in milliseconds. If sleep_ms is 0, it is calculated automatically based on the TIM configuration.
Return		If successful return DPM_SUCCESS, if failed return non-zero.
It makes PTIM stop and go to sleep state powering off all HW blocks in DA6200 except RTC and RTM. The next wake-up time is after sleep_ms milliseconds.		

int ptim_set_udp(unsigned char en, unsigned char *targetmac, unsigned short sport, unsigned short dport, unsigned long dip, unsigned char *data, int payload_len, unsigned char retry, unsigned long power, unsigned long rate)		
Parameter	en	Enables transmitting UDP data.
	targetmac	Target MAC address of UDP. If targetmac is NULL, MAC address in RTM set by POR/FULL application is used.
	sport	Source Port of UDP. If sport is 0, source port in RTM set by POR/FULL application is used.
	dport	Destination Port of UDP. If dport is 0, destination port in RTM set by POR/FULL application is used.
	dip	Destination IP of UDP. If dip is 0, destination IP in RTM set by POR/FULL application is used.
	data	pointer to user data of UDP
	payload_len	If payload_len is 0, PTIM transmits UDPH in which there is no payload. Otherwise PTIM transmits a user data over the UDP protocol.
	retry	Maximum MAC layer retry count when failed to transmit UDP packet.
	power	TX power of UDP. If power is 0, the TX power set by RTOS is used.
	rate	TX rate of UDP. If rate is 0, the TX rate set by RTOS is used. Refer to “Rate”.
Return		If successful return DPM_SUCCESS, if failed return non-zero.
This function configures a transmitting UDP frame. Rate 1 Mbps : 0x00 2 Mbps : 0x01 5.5 Mbps : 0x02 11 Mbps : 0x03 6 Mbps : 0x04 9 Mbps : 0x05 12 Mbps : 0x06 18 Mbps : 0x07 24 Mbps : 0x08 36 Mbps : 0x09 48 Mbps : 0x0a 54 Mbps : 0x0b		

© 2022 Renesas Electronics

int ptim_set_udp_txdone_callback(int (*txdone)(int txack))		
Parameter	txdone	Pointer to the callback function to call when finished a transmit.
Return		If successful return DPM_SUCCESS, if failed return non-zero.
Register a callback function when finished transmitting a UDP packet.		

int ptim_set_rx_handler(void (*rx_data_handler)(unsigned char *machdr, unsigned char *payload, int len))		
Parameter	rx_data_handler	Pointer to RX data handler.
Return		If successful return DPM_SUCCESS, if failed return non-zero.
Register a callback function when received data packet.		

5.1.2.3 PSDK CONFIGURATION API

unsigned long ptim_get_funcs(void)		
Parameter	void	
Return		It returns PTIM functions to run at this wake-up time. See the PTIM_FUNC definition in the below.
When the return value is 0, it means non-periodic function such as Beacon Loss will be processed at this wake-up time. When it is non-zero value, it means one of periodic functions of PTIM such as Keep Alive will be processed at this wake-up time. The following enum is the PTIM_FUNC definition: PTIM_FUNC_NO_FUNC(0x0): There is no a periodic PTIM function. PTIM_FUNC_TIM(0x1): TIM function to receive a beacon from the AP. PTIM_FUNC_BCMC(0x2): BCMC function to receive a BC/MC data from the AP. PTIM_FUNC_PS_POLL(0x4): PS_POLL function to transmit a PS-POLL frame to the AP. PTIM_FUNC_UC(0x8): UC function to receive UC data after the PS_POLL function. PTIM_FUNC_AUTO_ARP(0x10): AUTO_ARP function to transmit an ARP reply frame. PTIM_FUNC_ARPREQ(0x20): ARPREQ function to transmit an ARP request frame. PTIM_FUNC_UDP(0x40): UDP function to transmit a UDPH or a user defined UDP to a server. PTIM_FUNC_TCP_KA(0x80): TCP_KA function to transmit a TCP ACK frame to a server. PTIM_FUNC_KA(0x100): KA function to transmit a null function of MAC to the AP. PTIM_FUNC_DEAUTH_CHK(0x200): DEAUTH_CHK function to confirm the connection with AP.		

void ptim_set_funcs(unsigned long funcs)		
Parameter	pfunc	The new PTIM functions. See PTIM_FUNC.
Return		Nothing
It overrides PTIM_FUNC set in RTM. It can designate what kind of jobs PTIM process at this wake-up time with PTIM_FUNC definitions.		
void ptim_set_tim(unsigned char en, unsigned long period, unsigned char wait_bcn, unsigned char bcn_timeout_tu, unsigned char chk_cnt, unsigned long chk_step)		
Parameter	en	This enables/disables to receive beacon function of PTIM. 1 is for enabling.

© 2022 Renesas Electronics

Parameter	period	PTIM wakeup period. When it's 0, it keeps the configuration in RTM set by POR/FULL application. When it's 1, wake-up period will be beacon interval of the AP. Otherwise, PTIM will wake up after this period in unit of beacon interval. For example, when it's 2, PTIM will wake up every 2-beacon interval.
Parameter	wait_bcn	When this is 1, PTIM waits until it receives a beacon up to beacon interval.
Parameter	bcn_timeout_tu	PTIM will wait for a beacon for this time in unit of TU (1.024 ms)
Parameter	chk_cnt	Maximum count value to receive a beacon in unit of beacon interval. When PTIM failed to receive a beacon at the pre-defined wake up time, it can change its behavior with this value to try to receive a beacon. For example, PTIM wakes up at the pre-defined wake up time in RTM and it could not receive a beacon and go to sleep. But when this value is 2, PTIM will wake up after one beacon interval time (usually 100 ms), not after the pre-defined time in RTM. And it will try to receive a beacon again and when failed, it increases internal counter value by one and go to sleep again. It continues to wake up after a beacon interval to receive a beacon until the counter value reaches to chk_cnt value.
Parameter	chk_step	This determines how many times will the above operation related with chk_cnt be repeated. DA16200 (DA16600) finally determines BCN_LOSS if it couldn't receive a beacon even after this. PTIM will go to a FULL-BOOT after this.
Return		Nothing
Configuring PTIM for its operations.		

void ptim_set_ka(unsigned char en, unsigned long period_ms, unsigned char timeout_tu, unsigned char retry)		
Parameter	en	This enables/disables to transmit a Keep Alive function of PTIM. 1 is for enabling. Keep Alive packet is just a NULL frame.
Parameter	period_ms	KA function period in millisecond units.
Parameter	timeout_tu	TX-Done will be return from MAC layer when DA16200 (DA16600) sends a packet and it receives ACK from the peer correctly. When it could not receive ACK from peer for this timeout time, the return value from ptim_execute will be NO_ACK. PTIM should go to a FULL-BOOT for further process for this. Note that NO_ACK will be return from ptim_execute only for KA. Other transmitting packet such as UDPH or PS_POLL etc. do not affect to the return value of ptim_execute().
Parameter	retry	Maximum MAC layer retry count when failed to transmit KA packet.
Return		Nothing
Configuring KA function of PTIM.		

© 2022 Renesas Electronics

void ptim_set_bcmc(unsigned char en, unsigned char timeout_tu, unsigned char max)		
Parameter	en	This enables/disables to receive BC/MC function of PTIM. 1 is for enabling.
Parameter	timeout_tu	PTIM waits for receiving BC/MC packet from the AP for this time. This is in TU (1.024 ms) unit.
Parameter	max	Indicating how many BC/MC packets will be received in PTIM. The maximum count is 15.
Return		Nothing
Configuring BC/MC function of PTIM.		

void ptim_set_ps(unsigned char en, unsigned char timeout_tu, unsigned char retry)		
Parameter	en	This enables/disables to transmit a PS-POLL function of PTIM. 1 is for enabling.
Parameter	timeout_tu	TX-Done will be return from MAC layer when DA16200 (DA16600) receives ACK from the AP. Even if it could not receive ACK for this time, the return value from ptim_execute() will not be NO_ACK as said before. The NO_ACK from ptim_execute() will return only when DA16200 (DA16600) could not get ACK from the AP after transmitting KA.
Parameter	retry	Maximum MAC level layer count when failed to transmit PS-POLL packet.
Return		Nothing
Configuring PS-POLL function of PTIM.		

void ptim_set_uc(unsigned char en, unsigned char timeout_tu, unsigned char max)		
Parameter	en	This enables/disables to UC function of PTIM. 1 is for enabling. If disabled, PTIM will not wait for UC for duration of the below parameter, timeout_tu.
Parameter	timeout_tu	PTIM waits for receiving UC from the AP for this time. This is in TU (1.024 ms) unit.
Parameter	max	Indicating how many UC data will be received in PTIM. The maximum count is 15.
Return		Nothing
Configuring UC function of PTIM.		

void ptim_set_autoarp(unsigned char en, unsigned char timeout_tu, unsigned char retry, unsigned long period_ka)		
Parameter	en	This enables/disables to autonomous ARP function of PTIM. 1 is for enabling.
Parameter	timeout_tu	TX-Done will be return from MAC layer when DA16200 (DA16600) receives ACK from the AP. Even if it could not receive ACK for this time, the return value from ptim_execute() will not be NO_ACK as said before. The NO_ACK from ptim_execute() will return only when DA16200 (DA16600) could not get ACK from the AP after transmitting KA.
Parameter	retry	Maximum MAC layer retry count when failed to transmit ARP-REPLY packet.

© 2022 Renesas Electronics

Parameter	period_ka	AutoARP period in KA period units. For example, when this is 2, ARP-REPLY packet will be transmitted at every 2 KA transmitting time.
Return		Nothing
Configuring autonomous ARP function of PTIM.		

void ptim_set_udph(unsigned char en, unsigned char timeout_tu, unsigned char retry, unsigned long period_ka)		
Parameter	en	This enables/disables to UDPH function of PTIM. 1 is for enabling. UDPH packet consists of UDP header and no payload.
Parameter	timeout_tu	TX-Done will be return from MAC layer when DA16200 (DA16600) receives ACK from the AP. Even if it could not receive ACK for this time, the return value from ptim_execute() will not be NO_ACK as said before. The NO_ACK from ptim_execute() will return only when DA16200 (DA16600) could not get ACK from the AP after transmitting KA.
Parameter	retry	Maximum MAC layer retry count when failed to transmit UDPH packet.
Parameter	period_ka	UDPH period in KA period units. For example, when this is 2, UDPH packet will be transmitted at every 2 KA transmitting time.
Return		Nothing
Configuring autonomous UDPH function of PTIM.		

void ptim_set_arpreq(unsigned char en, unsigned char timeout_tu, unsigned char retry, unsigned long period_ka)		
Parameter	en	This enables/disables to ARPREQ function of PTIM. 1 is for enabling.
Parameter	timeout_tu	TX-Done will be return from MAC layer when DA16200 (DA16600) receives ACK from the AP. Even if it could not receive ACK for this time, the return value from ptim_execute() will not be NO_ACK as said before. The NO_ACK from ptim_execute() will return only when DA16200 (DA16600) could not get ACK from the AP after transmitting KA.
Parameter	retry	Maximum MAC layer retry count when failed to transmit ARPREQ packet.
Parameter	period_ka	ARPREQ period in KA period units. For example, when this is 2, ARPREQ packet will be transmitted at every 2 KA transmitting time.
Return		Nothing
Configuring ARPREQ function of PTIM.		

void ptim_set_deauthchk(unsigned char en, unsigned char timeout_tu, unsigned char retry, unsigned long period_ka, unsigned char chk_timeout_tu)		
Parameter	en	This enables/disables to deauthChk function of PTIM. 1 is for enabling.
Parameter	timeout_tu	TX-Done will be return from MAC layer when DA16200 (DA16600) receives ACK from the AP. When it could not receive ACK from peer for this timeout time, the return value from ptim_execute will be NO_ACK. PTIM should go to a full-boot for further process for this. Since KA packet is transmitted for DeauthChk function, ptim_execute() will return NO_ACK when no ACK is received from AP for this timeout period just like KA transmission case.
Parameter	retry	Maximum MAC layer retry count when failed to transmit KA packet.

© 2022 Renesas Electronics

Parameter	period_ka	DeauthChk function period in KA period units. For example, when this is 2, deauthChk will be performed at every 2 KA transmitting time.
Parameter	chk_timeout_tu	The time that PTIM waits for the deauthentication or disassociation frame.
Return		Nothing
Configuring deauthChk function of PTIM.		

void ptim_set_tcpka(unsigned char en, unsigned char timeout_tu, unsigned char retry, unsigned long period_ka)		
Parameter	en	This enables/disables to TCP ACK function of PTIM. 1 is for enabling.
Parameter	timeout_tu	TX-Done will be return from MAC layer when DA16200 (DA16600) receives ACK from the AP. Even if it could not receive ACK for this time, the return value from ptim_execute() will not be NO_ACK as said before. The NO_ACK from ptim_execute() will return only when DA16200 (DA16600) could not get ACK from the AP after transmitting KA.
Parameter	retry	Maximum MAC layer retry count when failed to transmit TCP ACK packet.
Parameter	period_ka	TCP ACK period in KA period units. For example, when this is 2, TCP ACK will be transmitted at every 2 KA transmitting time.
Return		Nothing
Configuring TCP ACK function of PTIM.		

void ptim_update_config(unsigned long renewal)		
Parameter	renewal	Flag to delete previous PTIM functions and to update new PTIM functions.
Return		Nothing
It updates the new configuration.		

5.1.2.4 PSDK HELPER API

void ptim_dbg_init(void)		
Parameter	void	Nothing
Return		Nothing
This function initializes UART and DBG module. This feature increases the current.		

© 2022 Renesas Electronics

void ptim_console_work(char *prompt)		
Parameter	prompt	A prompt label of console.
Return		Nothing
This enables console of PTIM.		

unsigned long ptim_get_p_user_rtm_base(void)		
Parameter	void	Nothing
Return		The base address of PTIM User area in RTM.
This gets the base address of PTIM User area in RTM.		
unsigned long ptim_get_p_user_rtm_size(void)		
Parameter	void	Nothing
Return		The size of RTM user.
This gets the size of PTIM User area in RTM.		

© 2022 Renesas Electronics

5.1.3 Example of Code

5.1.3.1 A General PTIM

The following example of code is a general form of PTIM. It receives a beacon, then goes into sleep state or goes to full-boot state, if needed.

```
int main(void)
{
	unsigned long status = DPM_ST_UNDEFINED;
	int result;

	////////////////////////////////////////////////////////////////////////////
	//
	// Write user initialization codes if needed
	//
	////////////////////////////////////////////////////////////////////////////

	// Initialize PTIM library
	result = ptim_initialize();

	////////////////////////////////////////////////////////////////////////////
	//
	// Write application codes if needed
	//
	////////////////////////////////////////////////////////////////////////////

	// Check the result to initialize PTIM library.
	if (DPM_SUCCESS == result)
		ptim_execute(&status);

	////////////////////////////////////////////////////////////////////////////
	//
	// Finalize PTIM library
	//
	////////////////////////////////////////////////////////////////////////////

	// Write finalizing codes if needed.

	////////////////////////////////////////////////////////////////////////////
	//
	// The end
	//
	////////////////////////////////////////////////////////////////////////////

	// Check the PTIM status
	if (status == DPM_ST_FROM_FAST) {
		// Go to sleep state.
		ptim_goto_sleep(0);
	} else {
		// Go to full-boot state.
		ptim_goto_fullboot(0, 0);
	}

	return 0;
}
```

 © 2022 Renesas Electronics

5.1.3.2 A General PTIM with DBG

This is an example with a debug feature.

```
int main(void)
{
	unsigned long status = DPM_ST_UNDEFINED;
	int result;

	////////////////////////////////////////////////////////////////////////////
	//
	// Write user initialization codes if needed
	//
	////////////////////////////////////////////////////////////////////////////

	// Initialize DBG module.
	tim_dbg_init();

	PRINTF("PTIM example with DBG feature\n");

	// Initialize PTIM library
	result = ptim_initialize();

	////////////////////////////////////////////////////////////////////////////
	//
	// Write application codes if needed
	//
	////////////////////////////////////////////////////////////////////////////

	// Check the result to initialize PTIM library.
	if (DPM_SUCCESS == result)
		ptim_execute(&status);

	////////////////////////////////////////////////////////////////////////////
	//
	// Finalize PTIM library
	//
	////////////////////////////////////////////////////////////////////////////

	// Write finalizing codes if needed.
	////////////////////////////////////////////////////////////////////////////
	//
	// The end
	//
	////////////////////////////////////////////////////////////////////////////

	// Check the PTIM status
	if (status == DPM_ST_FROM_FAST) {
		// Go to sleep state.
		ptim_goto_sleep(0);
	} else {
		// Go to full-boot state.
		ptim_goto_fullboot(0, 0);
	}

	return 0;
}
```

 © 2022 Renesas Electronics

5.1.3.3 A General PTIM with User's RTM

The following code is PTIM example with RTM. The FC9K_RTM_USER_SIZE and FC9K_RTM_PTIM_USER_SIZE place in da16x_system.h header. The sum of FC9K_RTM_USER_SIZE and FC9K_RTM_PTIM_USER_SIZE should be less than 8 K. This example needs to rebuild timlib.a.

```
// The size of user RTM for full-boot
#define       FC9K_RTM_USER_SIZE                        (0x1000)
// The size of RTM for PTIM
#define       FC9K_RTM_PTIM_USER_SIZE                   (0x1000)
```

```
int main(void)
{
       unsigned long status = DPM_ST_UNDEFINED;
       int result;
       unsigned long *p = (unsigned long *) ptim_get_p_user_rtm_base();

       ////////////////////////////////////////////////////////////////////////////
       //
       // Write user initialization codes if needed
       //
       ////////////////////////////////////////////////////////////////////////////

       // Initialize DBG module.
       tim_dbg_init();

       // User variable p is increased by 1 when PTIM is turned on.
       *p += 1;
       PRINTF("Base %08x Size %d: RTM Value %d\n", ptim_get_p_user_rtm_base(),
                     ptim_get_user_rtm_size(), *p);

       // Initialize PTIM library
       result = ptim_initialize();

       ////////////////////////////////////////////////////////////////////////////
       //
       // Write application codes if needed
       //
       ////////////////////////////////////////////////////////////////////////////

       // Check the result to initialize PTIM library.
       if (DPM_SUCCESS == result)
              ptim_execute(&status);

       ////////////////////////////////////////////////////////////////////////////
       //
       // Finalize PTIM library
       //
       ////////////////////////////////////////////////////////////////////////////

       // Write finalizing codes if needed.

       ////////////////////////////////////////////////////////////////////////////
       //
       // The end
       //
       ////////////////////////////////////////////////////////////////////////////
```

 © 2022 Renesas Electronics

```
        // Check the PTIM status
        if (status == DPM_ST_FROM_FAST) {
                // Go to sleep state.
                ptim_goto_sleep(0);
        } else {
                // Go to full-boot state.
                ptim_goto_fullboot(0, 0);
        }

        return 0;
}
```

```
Base 00f8b000 Size 4096: RTM Value 1
Base 00f8b000 Size 4096: RTM Value 2
Base 00f8b000 Size 4096: RTM Value 3
Base 00f8b000 Size 4096: RTM Value 4
Base 00f8b000 Size 4096: RTM Value 5
…
```

5.1.3.4 The Customized UDP Transmission

The following code is an example of the customized UDP transmission until *txack* is true or counter *n* is equal to 2.

```
int ptimcmd_udp_handler(int txack)
{
        // The count of transmitted UDP data
        static int n = 0;

        // It need to transmit more UDP data.
        int result = DPM_E_FAIL;

        // UDP default configurations
        unsigned char en = 1;
        unsigned char targetmac[6];
        unsigned short sport = 0;
        unsigned short dport = 0;
        unsigned long dip = 0;
        unsigned long data = 0;
        int payload_len = 0;
        unsigned char retry = 0;
        unsigned long power = 0;
        unsigned long rate = 0;

        // Get the UDP configuration.
        ptim_get_udp_default(&en, &targetmac[0], &sport, &dport, &dip, &data,
                        &payload_len, &retry, &power, &rate);

        // Check if txack is true.
        // If txack is false, we need to transmit more UDP data.
        if (txack) {
                // Complete UDP transmission.
                result = DPM_SUCCESS;
        } else {
                switch (n) {
```


© 2022 Renesas Electronics

```
                case 0:
                        // Configure 2nd UDP data.
                        // The last byte of MAC sets 0xff for UDP test.
                        targetmac[5] = 0xff;
                        ptim_set_udp(en, targetmac, sport, dport, dip,
                                        (unsigned char *) data, payload_len, retry,
                                        power, rate);
                        n++;
                        break;
                case 1:
                        // Configure 3nd UDP data.
                        // The last byte of MAC sets 0xff for UDP test.
                        targetmac[5] = 0xff;
                        ptim_set_udp(en, targetmac, sport, dport, dip,
                                        (unsigned char *) data, payload_len, retry,
                                        power, rate);
                        n++;
                        break;
                default:
                        // Complete UDP transmission.
                        result = DPM_SUCCESS;
                        break;
                }
        }

        return result;
}

int ptimcmd_udp_initialize()
{
        // Get the default UDP configuration.
        ptim_get_udp_default(&udp_en, &udp_targetmac[0], &udp_sport,
                        &udp_dport, &udp_dip, &udp_data,&udp_payload_len, &udp_retry,
                        &udp_power, &udp_rate);

        // Target MAC ff:ff:ff:ff:ff:ff
        memset(&udp_targetmac[0], 0xff, 6);

        PRINTF("%s::%d en:%d,power:%x,rate:%x,payload_len:%d,retry:%d\n",
                        __func__, __LINE__, udp_en, udp_power, udp_rate,
                        udp_payload_len, udp_retry);

        // Initialize the first udp data.
        ptim_set_udp(udp_en, (unsigned char *) &udp_targetmac, udp_sport,
                        udp_dport, udp_dip, (unsigned char *) udp_data,
                        udp_payload_len, udp_retry, udp_power, udp_rate);

        // Set a callback for udp test.
        ptim_set_udp_txdone_callback(ptimcmd_udp_handler);

        return DPM_SUCCESS;
}

int ptimcmd_udp_finalize()
{
        // Recover the UDP configuration.
        ptim_set_udp(udp_en, (unsigned char *) &udp_targetmac, udp_sport,
                        udp_dport, udp_dip, (unsigned char *) udp_data,
```


© 2022 Renesas Electronics

```
                        udp_payload_len, udp_retry, udp_power, udp_rate);

        return DPM_SUCCESS;
}
```

```
int main(void)
{
        unsigned long status = DPM_ST_UNDEFINED;
        int result;

        ////////////////////////////////////////////////////////////////////////////
        //
        // Write user initialization codes if needed
        //
        ////////////////////////////////////////////////////////////////////////////

        ptimcmd_udp_initialize();

        // Initialize PTIM library
        result = ptim_initialize();

        ////////////////////////////////////////////////////////////////////////////
        //
        // Write application codes if needed
        //
        ////////////////////////////////////////////////////////////////////////////

        // Check the result to initialize PTIM library.
        if (DPM_SUCCESS == result)
                ptim_execute(&status);

        ////////////////////////////////////////////////////////////////////////////
        //
        // Finalize PTIM library
        //
        ////////////////////////////////////////////////////////////////////////////

        ptimcmd_udp_finalize();

        // Check the PTIM status
        if (status == DPM_ST_FROM_FAST) {
                // Go to sleep mode.
                ptim_goto_sleep(0);
        } else {
                // Go to fullboot.
                ptim_goto_fullboot(0, 0);
        }

        return 0;
}
```

5.1.3.5 User RX Data Handler

The following code is an example of how the RX handler parses the received data.

```
void ptimcmd_data_handler(unsigned char *machdr, unsigned char *payload, int len)
```


© 2022 Renesas Electronics

```
{
        PRINTF("paylen: %d\n", len);
        PRINTF("MAC Header: %02x %02x %02x %02x %02x %02x %02x %02x "
                        "%02x %02x %02x %02x %02x %02x %02x %02x "
                        "%02x %02x %02x %02x %02x %02x %02x %02x\n",
                        machdr[0], machdr[1], machdr[2], machdr[3],
                        machdr[4], machdr[5], machdr[6], machdr[7],
                        machdr[8], machdr[9], machdr[10], machdr[11],
                        machdr[12], machdr[13], machdr[14], machdr[15],
                        machdr[16], machdr[17], machdr[18], machdr[19],
                        machdr[20], machdr[21], machdr[22], machdr[23]);
        PRINTF("Payload: %02x %02x %02x %02x %02x %02x %02x %02x "
                        "%02x %02x %02x %02x %02x %02x %02x %02x "
                        "%02x %02x %02x %02x %02x %02x %02x %02x\n",
                        payload[0], payload[1], payload[2], payload[3],
                        payload[4], payload[5], payload[6], payload[7],
                        payload[8], payload[9], payload[10], payload[11],
                        payload[12], payload[13], payload[14], payload[15],
                        payload[16], payload[17], payload[18], payload[19],
                        payload[20], payload[21], payload[22], payload[23]);
}
```

```
int main(void)
{
        unsigned long status = DPM_ST_UNDEFINED;
        int result;

        ////////////////////////////////////////////////////////////////////////////
        //
        // Write user initialization codes if needed
        //
        ////////////////////////////////////////////////////////////////////////////

        // Initialize DBG module.
        ptim_dbg_init();

        // Set RX handler to be parsed a received data.
        ptim_set_rx_handler(ptimcmd_data_handler);

        // Initialize PTIM library
        result = ptim_initialize();

        ////////////////////////////////////////////////////////////////////////////
        //
        // Write application codes if needed
        //
        ////////////////////////////////////////////////////////////////////////////

        // Check the result to initialize PTIM library.
        if (DPM_SUCCESS == result)
                ptim_execute(&status);

        ////////////////////////////////////////////////////////////////////////////
        //
        // Finalize PTIM library
        //
        ////////////////////////////////////////////////////////////////////////////

        // Check the PTIM status
```

 © 2022 Renesas Electronics

```
        if (status == DPM_ST_FROM_FAST) {
                // Go to sleep mode.
                ptim_goto_sleep(0);
        } else {
                // Go to fullboot.
                ptim_goto_fullboot(0, 0);
        }

        return 0;
}
```

```
paylen: 36
MAC Header: 08 02 00 00 ff ff ff ff ff ff 88 36 6c 38 11 32 d8 c4 97 c5 1c 13 10 81
Payload: aa aa 03 00 00 00 08 06 00 01 08 00 06 04 00 01 d8 c4 97 c5 1c 13 c0 a8
paylen: 36
MAC Header: 88 0a 6c 00 aa ff 22 02 01 06 88 36 6c 38 11 32 d8 c4 97 c5 1c 13 70 00
Payload: aa aa 03 00 00 00 08 06 00 01 08 00 06 04 00 01 d8 c4 97 c5 1c 13 c0 a8
paylen: 36
MAC Header: 88 2a 70 00 aa ff 22 02 01 06 88 36 6c 38 11 32 d8 c4 97 c5 1c 13 80 00
Payload: aa aa 03 00 00 00 08 06 00 01 08 00 06 04 00 01 d8 c4 97 c5 1c 13 c0 a8
paylen: 68
MAC Header: 88 2a 7c 00 aa ff 22 02 01 06 88 36 6c 38 11 32 d8 c4 97 c5 1c 13 90 00
Payload: aa aa 03 00 00 00 08 00 45 00 00 3c ae 5c 00 00 80 01 0a d7 c0 a8 00 1f
```

5.1.3.6 PTIM Functions Configurations: TIM & KA

The following code is an example of changes to TIM and KA's period. The `ptim_update_config` function is called only once.

```
void ptimcmd_change_config()
{
        // Check if it is periodic PTIM. The new configuration of aperiodic PTIM may
        // cause irregular PTIM.
        if (ptim_get_funcs() != 0) {
                // TIM is enabled.
                unsigned char tim_en = 1;
                // TIM's frequency is 32 times beacon.
                unsigned long tim_period = 32;
                // It waits for beacon during bcn_timeout_tu.
                unsigned char tim_wait_bcn = 0;
                // The beacon reception duration is 10 TUs.
                unsigned char tim_bcn_timeout_tu = 10;
                // The beacon reception retries are 2 times in the beacon interval.
                unsigned char tim_chk_cnt = 2;
                // The beacon reception retries are 4 times in the TIM period.
                unsigned char tim_chk_step = 4;
                // Set TIM configurations.
                ptim_set_tim(tim_en, tim_period, tim_wait_bcn, tim_bcn_timeout_tu,
                                                tim_chk_cnt, tim_chk_step);

                // KA is enabled.
                unsigned char ka_en = 1;
                // KA's frequency is 30 seconds.
                unsigned long ka_period_ms = 30000;
                // KA's timeout is 100 TUs.
                unsigned char ka_timeout_tu = 100;
                // The number of KA retries is 7 times.
```


© 2022 Renesas Electronics

```
                unsigned char ka_retry = 7;
                // Set KA configurations.
                ptim_set_ka(ka_en, ka_period_ms, ka_timeout_tu, ka_retry);

                // It is called only once.
                ptim_update_config(0);
        }
}
```

```
int main(void)
{
        unsigned long status = DPM_ST_UNDEFINED;
        int result;

        ////////////////////////////////////////////////////////////////////////////
        //
        // Write user initialization codes if needed
        //
        ////////////////////////////////////////////////////////////////////////////

        // Set new configuration.
        ptimcmd_change_config();

        // Initialize PTIM library
        result = ptim_initialize();

        ////////////////////////////////////////////////////////////////////////////
        //
        // Write application codes if needed
        //
        ////////////////////////////////////////////////////////////////////////////

        // Check the result to initialize PTIM library.
        if (DPM_SUCCESS == result)
                ptim_execute(&status);

        ////////////////////////////////////////////////////////////////////////////
        //
        // Finalize PTIM library
        //
        ////////////////////////////////////////////////////////////////////////////

        // Check the PTIM status
        if (status == DPM_ST_FROM_FAST) {
                // Go to sleep mode.
                ptim_goto_sleep(0);
        } else {
                // Go to fullboot.
                ptim_goto_fullboot(0, 0);
        }

        return 0;
}
```

5.1.3.7 PTIM Functions Configurations: TIM

The following code is an example of changes to TIM configurations.

```
void ptimcmd_change_config()
```

 © 2022 Renesas Electronics

```
{
        // Check if it is periodic PTIM. The new configuration of aperiodic PTIM may
        // cause irregular PTIM.
        if (ptim_get_funcs() != 0) {
                // TIM is enabled.
                unsigned char tim_en = 1;
                // TIM's frequency is 12 times beacon.
                unsigned long tim_period = 12;
                // It waits for beacon during bcn_timeout_tu.
                unsigned char tim_wait_bcn = 0;
                // The beacon reception duration is 100 TUs.
                unsigned char tim_bcn_timeout_tu = 100;
                // The beacon interval retries do not use.
                unsigned char tim_chk_cnt = 0;
                // The beacon reception retries are 10 times in the TIM period.
                unsigned char tim_chk_step = 10;
                // Set TIM configurations.
                ptim_set_tim(tim_en, tim_period, tim_wait_bcn, tim_bcn_timeout_tu,
                                                tim_chk_cnt, tim_chk_step);

                ptim_update_config(0);
        }
}
```

```
int main(void)
{
        unsigned long status = DPM_ST_UNDEFINED;
        int result;

        ////////////////////////////////////////////////////////////////////////////
        //
        // Write user initialization codes if needed
        //
        ////////////////////////////////////////////////////////////////////////////

        // Set new configuration.
        ptimcmd_change_config();

        // Initialize PTIM library
        result = ptim_initialize();

        ////////////////////////////////////////////////////////////////////////////
        //
        // Write application codes if needed
        //
        ////////////////////////////////////////////////////////////////////////////

        // Check the result to initialize PTIM library.
        if (DPM_SUCCESS == result)
                ptim_execute(&status);

        ////////////////////////////////////////////////////////////////////////////
        //
        // Finalize PTIM library
        //
        ////////////////////////////////////////////////////////////////////////////

        // Check the PTIM status
        if (status == DPM_ST_FROM_FAST) {
```

 © 2022 Renesas Electronics

```
                // Go to sleep mode.
                ptim_goto_sleep(0);
        } else {
                // Go to fullboot.
                ptim_goto_fullboot(0, 0);
        }

        return 0;
}
```

5.1.3.8 PTIM Functions Configurations: KA

The following code is an example of changes to KA configurations.

```
void ptimcmd_change_config()
{
        // Check if it is periodic PTIM. The new configuration of aperiodic PTIM may
        // cause irregular PTIM.
        if (ptim_get_funcs() != 0) {
                // KA is enabled.
                unsigned char ka_en = 1;
                // KA's frequency is 10 seconds.
                unsigned long ka_period_ms = 10000;
                // KA's timeout is 100 TUs.
                unsigned char ka_timeout_tu = 100;
                // The number of KA retries is 7 times.
                unsigned char ka_retry = 7;
                // Set KA configurations.
                ptim_set_tim(tim_en, tim_period, tim_wait_bcn, tim_bcn_timeout_tu,
                                        tim_chk_cnt, tim_chk_step);

                ptim_update_config(0);
        }
}
```

```
int main(void)
{
        unsigned long status = DPM_ST_UNDEFINED;
        int result;

        ////////////////////////////////////////////////////////////////////////////
        //
        // Write user initialization codes if needed
        //
        ////////////////////////////////////////////////////////////////////////////

        // Set new configuration.
        ptimcmd_change_config();

        // Initialize PTIM library
        result = ptim_initialize();

        ////////////////////////////////////////////////////////////////////////////
        //
        // Write application codes if needed
        //
        ////////////////////////////////////////////////////////////////////////////
```

 © 2022 Renesas Electronics

```
	// Check the result to initialize PTIM library.
	if (DPM_SUCCESS == result)
		ptim_execute(&status);

	////////////////////////////////////////////////////////////////////////
	//
	// Finalize PTIM library
	//
	////////////////////////////////////////////////////////////////////////

	// Check the PTIM status
	if (status == DPM_ST_FROM_FAST) {
		// Go to sleep mode.
		ptim_goto_sleep(0);
	} else {
		// Go to fullboot.
		ptim_goto_fullboot(0, 0);
	}

	return 0;
}
```

5.1.3.9 PTIM Functions Configurations: UC & BCMC

The following code is an example of changes to UC & BC/MC configurations.

This example is:

- UC data are received for up to 5 and 20 TUs.
- BC/MC data are received for up to 5 and 20 TUs.
- There is no need to call ptim_update_config as the frequency does not change.

```
void ptimcmd_change_config()
{
	// Check if it is periodic PTIM. The new configuration of aperiodic PTIM may
	// cause irregular PTIM.
	if (ptim_get_funcs() != 0) {
		// UC is enabled.
		unsigned char uc_en = 1;
		// It waits for UC data during 20 TUs.
		unsigned char uc_timeout_tu = 20;
		// UC data receives up to 5.
		unsigned char uc_max = 5;
		// Set UC configurations.
		ptim_set_uc(uc_en, uc_timeout_tu, uc_max);
		// BCMC is enabled.
		unsigned char bcmc_en = 1;
		// It waits for BC/MC data during 20 TUs.
		unsigned char bcmc_timeout_tu = 20;
		// BC/MC data receives up to 5.
		unsigned char bcmc_max = 5;
		// Set BCMC configurations.
		ptim_set_bcmc(bcmc_en, bcmc_timeout_tu, bcmc_max);
	}
}
```


© 2022 Renesas Electronics

```
int main(void)
{
	unsigned long status = DPM_ST_UNDEFINED;
	int result;

	////////////////////////////////////////////////////////////////////////////
	//
	// Write user initialization codes if needed
	//
	////////////////////////////////////////////////////////////////////////////

	// Set new configuration.
	ptimcmd_change_config();

	// Initialize PTIM library
	result = ptim_initialize();

	////////////////////////////////////////////////////////////////////////////
	//
	// Write application codes if needed
	//
	////////////////////////////////////////////////////////////////////////////

	// Check the result to initialize PTIM library.
	if (DPM_SUCCESS == result)
		ptim_execute(&status);

	////////////////////////////////////////////////////////////////////////////
	//
	// Finalize PTIM library
	////////////////////////////////////////////////////////////////////////////
	// Check the PTIM status
	if (status == DPM_ST_FROM_FAST) {
		// Go to sleep mode.
		ptim_goto_sleep(0);
	} else {
		// Go to fullboot.
		ptim_goto_fullboot(0, 0);
	}
	return 0;
}
```

5.2 PTIM ADC

5.2.1 Introduction

The DA16200 (DA16600) has Analog-to-Digital Converters (ADC); namely, a four-channel single-end ADC of 12-bit resolution. Analog input is measured by means of 4 pins from GPIO0 to GPIO3, and the pin selection is changed through the register setting. See Figure 6 and Table 5.

© 2022 Renesas Electronics

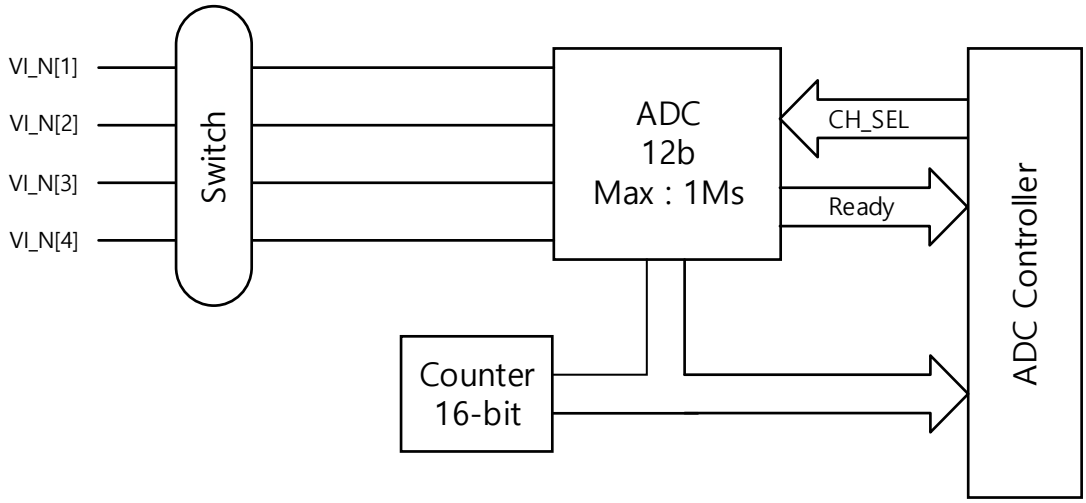


Figure 6: ADC Control Block Diagram

Table 5: AUX ADC Pin Configuration

Pin Name	Pin Number		I/O	Function Name
	QFN	fcCSP		
GPIOA3	36	D4	A	Analog signal
GPIOA2	37	B2	A	Analog signal
GPIOA1	38	C3	A	Analog signal
GPIOA0	39	A3	A	Analog signal

For more details, see DA16200 (DA16600) Datasheet in Ref. [1].

© 2022 Renesas Electronics

5.2.2 PTIM ADC Interface API

Table 6: PTIM ADC Interface API Elements

HANDLE DRV_PADC_CREATE(UINT32 dev_id)		
Parameter	dev_id	Device number to create handle
Return		If successful return handle for such device, if failed return NULL
Description		Function create handle with parameter `dev_id` designated

int DRV_PADC_INIT(unsigned int use_timestamp)		
Parameter	use_timestamp	Device handle to initialize
Return		If successful return TRUE, if failed return FALSE
Description		ADC Initialization command

int DRV_PADC_START(UINT32 divider12, UINT32 dummy)		
Parameter	divider12	Fs = 1 MHz / (div12 +1)
Return		If successful return TRUE, if failed return FALSE
Description		ADC start command

int DRV_PADC_STOP(void)	
Return	If successful return TRUE, if failed return FALSE
Description	ADC stop command

Int DRV_PADC_CLOSE(void)	
Return	If successful return TRUE, if failed return FALSE
Description	ADC driver close

int DRV_PADC_READ(UINT32 channel, UINT32 *data, UINT32 dummy)		
Parameter	channel	Channel number to read instant ADC value
	*data	Buffer to read
Return		If successful return TRUE, if failed return FALSE
Description		ADC read command, instant value

int DRV_PADC_READ_FIFO(UINT32 channel, UINT16 *p_data, UINT32 dummy)		
Parameter	channel	Channel number to read
	*p_data	Buffer block to read
Return		If successful return TRUE, if failed return FALSE

© 2022 Renesas Electronics

int DRV_PADC_READ_FIFO(UINT32 channel, UINT16 *p_data, UINT32 dummy)	
Description	ADC read command, FIFO ADC value

int DRV_PADC_ENABLE_CHANNEL(UINT32 channel, unsigned int sel_adc, UINT32 dummy)		
Parameter	channel	Channel number to set ADC devices
	sel_adc	12: SMI 12B ADC, 0: disable
Return		If successful return TRUE, if failed return FALSE
Description		ADC channel enable command

int DRV_PADC_WAIT_FIFO_HALF (UINT32 channel);		
Parameter	channel	Channel number to wait FIFO Half Interrupt
Return		If receive ADC FIFO Half Interrupt Occurred
Description		ADC FIFO Half interrupt wait command

int DRV_PADC_ENABLE_FIFO_HALF_INTERRUPT (UINT32 channel);		
Parameter	channel	Channel number to enable FIFO Half Interrupt
Return		If successful return TRUE, if failed return FALSE
Description		ADC FIFO Half interrupt wait command

© 2022 Renesas Electronics

5.3 DPM Power Profile

With Keysight, a current consumption measuring tester, the user can check how DPM works in Connected communication. DPM allows the system to stay in Sleep mode most of the time and only wake up (and stay active for only a small amount of time to get the job done) when needed.

In the Keysight snapshot below, DA16200 (DA16600) was in Sleep mode until it woke up to post a periodic status message to the broker. Once DA16200 (DA16600) received the response, it enters and stays in Sleep mode until the next Status Message Tx time (the interval depends on application).

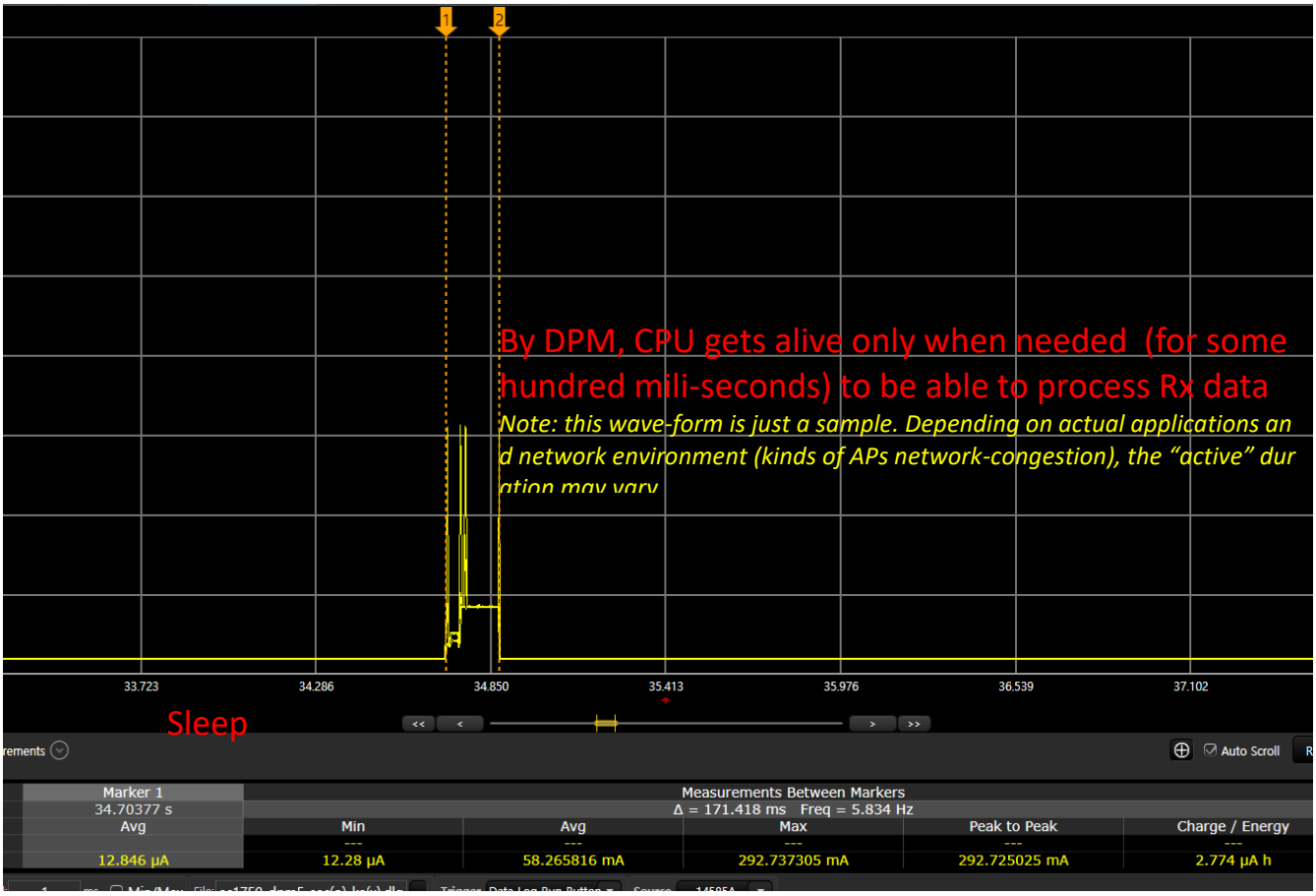


Figure 7: DPM Sleep Mode Current

© 2022 Renesas Electronics

6 Example Application

6.1 PADC + GPIO + UDP

This section shows how to use the I2C interface.

6.1.1 How to Run

1. Open workspace for the PTIM ADC Sample.
   a. Path: .\apps\common\examples\PTIM\pTim_ADC\build\DA16200_sample.eww
      The sample application code is written in the following source file.
      .\apps\common\examples\PTIM\pTim_ADC\src\ptim_adc_sample.c

6.1.2 Operation

1. Hardware setup.

Table 7: PADC Example Pin Description

GPIO	MUX	REMARK
GPIOA0	CH0	Connect Power Supply
GPIOA1	CH1	Connect Power Supply
GPIOA2	GPIO	GPIO Control for Supplying Power to External Sensor
GPIOA3	GPIO	GPIO Control for Supplying Power to External Sensor

2. Set PAD Mux.

```
_fc9k_io_pinmux(PIN_AMUX, AMUX_AD12);
_fc9k_io_pinmux(PIN_BMUX, BMUX_GPIO);
```

3. Prepare GPIO To Read ADC Data.

```
void prepare_gpio_to_adc_read(void)
{
        HANDLE dpm_gpio_cntrl_0 ;
        unsigned short data_mask;

        FC9K_CLOCK_SCGATE->Off_DAPB_GPIO0 = 0;

        dpm_gpio_cntrl_0 = GPIO_CREATE(GPIO_UNIT_A);
        GPIO_INIT(dpm_gpio_cntrl_0);

        data_mask = SS_MUX_READ_PREPH_TEMP|SS_MUX_READ_PREPH_BAT;
        GPIO_IOCTL(dpm_gpio_cntrl_0, GPIO_SET_OUTPUT, (void *)&data_mask);

        data_mask = SS_MUX_READ_PREPH_TEMP;
        GPIO_WRITE(dpm_gpio_cntrl_0, SS_MUX_READ_PREPH_TEMP, &data_mask, 2);

        data_mask = SS_MUX_READ_PREPH_BAT;
        GPIO_WRITE(dpm_gpio_cntrl_0, SS_MUX_READ_PREPH_BAT, &data_mask, 2);

        dpm_wait_clk(6); // wait 1 / 32768 x 6 second = 183us
}
```

4. Get ADC Data.

```
void get_adc_avg_data(UINT32 *adc0, UINT32 *adc1)
```


© 2022 Renesas Electronics

```
{
        int i, j;

        UINT32  adc_data0[16], adc_data1[16], sum0 = 0, sum1 = 0;

        FC9K_CLOCK_SCGATE->Off_DAPB_AuxA = 0;
        FC9K_CLOCK_SCGATE->Off_DAPB_APBS = 0;

        DRV_PADC_CREATE(FC9050_ADC_DEVICE_ID);
        DRV_PADC_INIT(FC9050_ADC_NO_TIMESTAMP);

        // Start. Set Sampling Frequency. 12B ADC Set to 200KHz
        DRV_PADC_START(FC9050_ADC_DIVIDER_12, 0);

        // Set ADC_0 to 12Bit ADC, ADC_1 to 12Bit ADC
        DRV_PADC_ENABLE_CHANNEL(FC9050_ADC_CH_0, FC9050_ADC_SEL_ADC_12, 0);
        DRV_PADC_ENABLE_CHANNEL(FC9050_ADC_CH_1, FC9050_ADC_SEL_ADC_12, 0);

        DRV_PADC_ENABLE_FIFO_HALF_INTERRUPT(FC9050_ADC_CH_0);
        DRV_PADC_ENABLE_FIFO_HALF_INTERRUPT(FC9050_ADC_CH_1);

        //Flushing FIFO
        for (i = 0; i < 8; i++) {
                DRV_PADC_READ_FIFO(FC9050_ADC_CH_0, adc_data0, 0);
                DRV_PADC_READ_FIFO(FC9050_ADC_CH_1, adc_data1, 0);
        }

        for (i = 0; i < 4; i++){
                DRV_PADC_WAIT_FIFO_HALF(FC9050_ADC_CH_0);
                DRV_PADC_WAIT_FIFO_HALF(FC9050_ADC_CH_1);

                for (j = 0; j < 4; j++) {
                        DRV_PADC_READ_FIFO(FC9050_ADC_CH_0,
                                                &adc_data0[(i * 4) + j], 0);
                        DRV_PADC_READ_FIFO(FC9050_ADC_CH_1,
                                                &adc_data1[(i * 4) + j], 0);
                }
        }

        for (i = 0; I < 16; i++) {
                sum0 += adc_data0[i] & 0xffff;
                sum1 += adc_data1[i] & 0xffff;
        }

        *adc0 = (sum0 >> 4) / 16;
        *adc1 = (sum1 >> 4) / 16;
}
```

5. After ADC Read, Release GPIO to Input Mode.

```
void release_gpio(void)
{
        HANDLE dpm_gpio_cntrl_0;
        unsigned short data_mask;

        dpm_gpio_cntrl_0 = GPIO_CREATE(GPIO_UNIT_A);
        GPIO_INIT(dpm_gpio_cntrl_0);
```


© 2022 Renesas Electronics

```
        data_mask = SS_MUX_READ_PREPH_TEMP|SS_MUX_READ_PREPH_BAT;
        GPIO_IOCTL(dpm_gpio_cntrl_0, GPIO_SET_INPUT, (void *) &data_mask);
}
```

6. Prepare UDP Packet to Send.

```
void prepare_udp_packet(UINT8 *data, UINT32 data_len)
{
        // Get the default UDPH configuration.
        ptim_get_udp_default(&udp_en, &udp_targetmac[0], &udp_sport,
                &udp_dport, &udp_dip, &udp_data,&udp_payload_len, &udp_retry,
                &udp_power, &udp_rate);

        // Target MAC ff:ff:ff:ff:ff:ff
        memset(&udp_targetmac[0], 0xff, 6);
        udp_data = (UINT32) tx_data;
        udp_payload_len = data_len;
        udp_sport = 5000;
        udp_dport = 5001;
        udp_retry = 3;
        udp_power = 0;
        udp_rate = 0x10000404; //11g 6m
        udp_dip = 0xff00a8c0;  //192.168.0.255

        udp_en = 1;

        PRINTF("%s::%d en:%d,power:%x,rate:%x,payload_len:%d,retry:%d\n",
                __func__, __LINE__, udp_en, udp_power, udp_rate,
                udp_payload_len, udp_retry);

        // Initialize the first UDPH data.
        ptim_set_udp(udp_en, (unsigned char *) &udp_targetmac, udp_sport,
                udp_dport, udp_dip, (unsigned char *) udp_data, udp_payload_len,
                udp_retry, udp_power, udp_rate);

        // Set a callback for UDPH test.
        ptim_set_udp_txdone_callback(udp_tx_done_callback_func);
}
```

 © 2022 Renesas Electronics

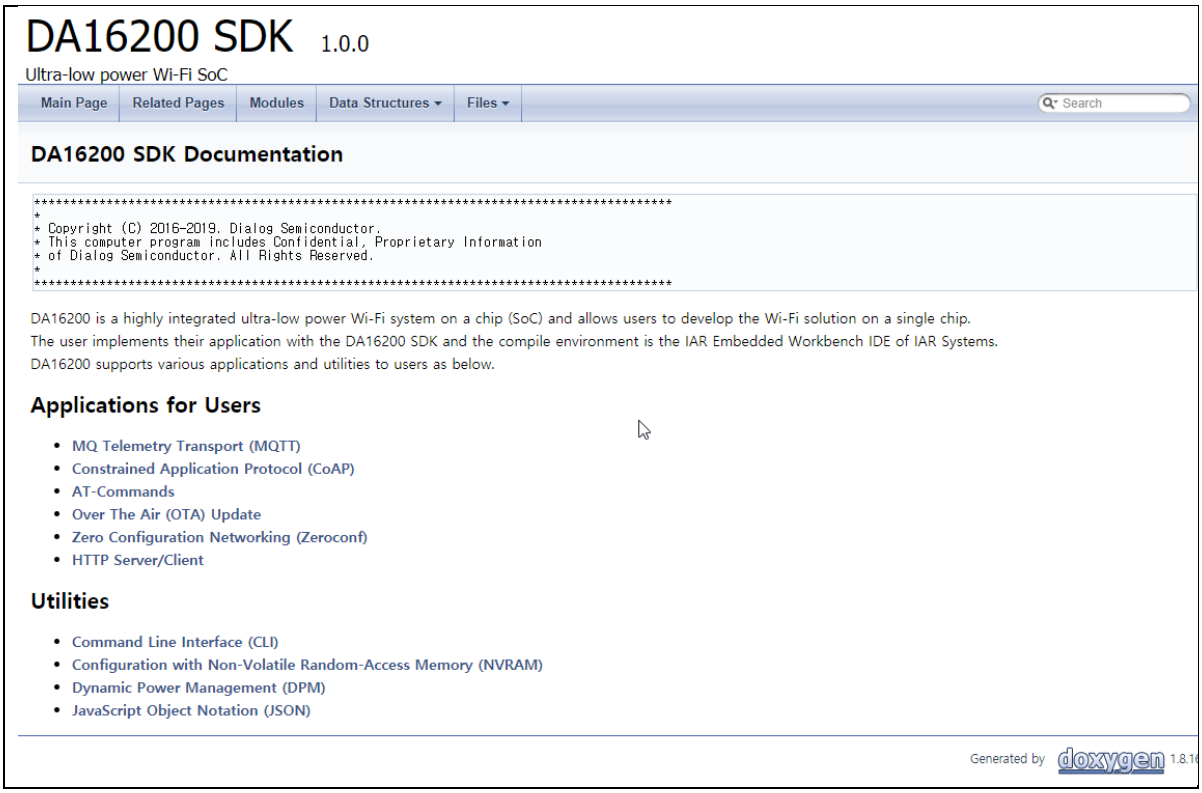
Appendix A Doxygen Documents

DA16200 (DA16600) SDK supports online documents which is a follow-up Doxygen document format.

To start the doxygen document page, open index.html in the Web-browser.

NOTE
Compatible with all web browsers.

~/DA16200_SDK/doc/html/index.html



 © 2022 Renesas Electronics

Revision History

Revision	Date	Description
1.4	28-Mar-2022	Update logo, disclaimer, copyright.
1.3	29-Nov-2021	Title was changed.
1.2	23-Aug-2021	Applied changes to SDK folder hierarchy.
1.1	10-Apr-2020	Editorial and template application
1.0	28-Feb-2020	Preliminary DRAFT Release

© 2022 Renesas Electronics

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

Dialog Semiconductor's 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.

© 2022 Renesas Electronics

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

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.