Realtek Ameba-Z FreeRTOS Tickless

Table of Contents

1 FreeRTOS Low Power Feature 3
1.1 FreeRTOS tickless design 3
1.2 Wakelock Feature 5
1.3 Use Ameba sleep in tickless 7
1.4 Wakelock AT command 8
2 Put WLAN into tickless design 9
2.1 IEEE 802.11 power management 9
2.2 Ameba LPS 9
2.3 Ameba IPS 10
3 Measure Power Consumption 12
3.1 Hardware preparation 12
3.2 Build SDK example 14

1 FreeRTOS Low Power Feature

1.1 FreeRTOS tickless design

FreeRTOS support a low power feature called tickless. It is implemented in idle task which has lowest priority. It means it is invoked when there is no other task under running. Its idea is place the microcontroller into a low power state if it expects there will be no event in nearly future. It calculates expected idle time by looking timer task list. Then it performs suspend action by using ARM instruction “WFI” (Wait For Interrupt) which makes the processor suspend execution (Clock is stopped) until interrupt happened. The interrupts include timer which set by FreeRTOS with value equal to expected idle time. So every time idle task is invoked, it calculates the expected idle time, setup timer and then put process into suspend.

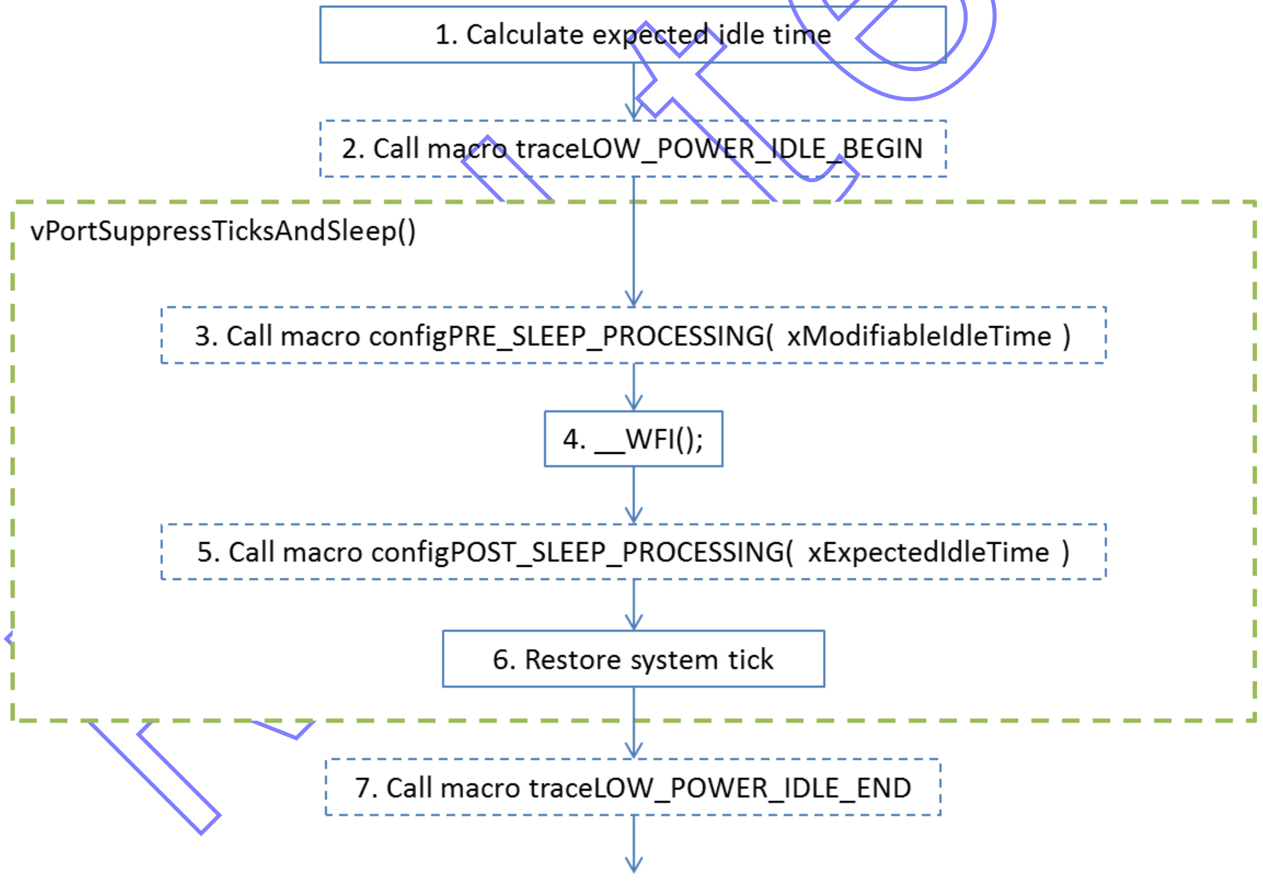

1. Calculate expected idle time
2. Call macro traceLOW_POWER_IDLE_BEGIN
vPortSuppressTicksAndSleep()
3. Call macro configPRE_SLEEP_PROCESSING(xModifiableIdleTime)
4. __WFI();
5. Call macro configPOST_SLEEP_PROCESSING(xExpectedIdleTime)
6. Restore system tick
7. Call macro traceLOW_POWER_IDLE_END

Figure 1 FreeRTOS Tickless in idle task

At step 1, it calculates expected idle time. Then it does some condition check. For example, it checks if the idle time is larger than configEXPECTED_IDLE_TIME_BEFORE_SLEEP, otherwise it aborts suppressing tick process.

Step 2 and step 7 are un-implemented macro. It is for tracing sleep process before/after vPortSuppressTickAndSleep(). Although it may abort sleep process inside vPortSuppressTickAndSLeep(), so **we can consider Step 2 as First gate** before entering sleep.

After step 2, there are some tasks. It double check sleep conditions, backup timer related registers, and check if some interrupt happens at this moment.

Step 3 and step 5 are un-implemented macro which are actually enter/leave sleep. So **we can consider Step 3 as second gate** before entering sleep.

After Step 3, it is enter sleep. FreeRTOS uses WFI which is AMR instruction for sleep.

1.2 Wakelock Feature

In some situations, we need system keep awake to receive certain events. Otherwise event might be missed when system is under sleep. An idea of wakelock is introduced that system cannot sleep if some module holding wakelock.

We implement wakelock api by implementing macro “traceLOW_POWER_IDLE_BEGIN” as a function and check wakelock status. If there is no one holding wakelock, then system is permit to sleep. If there are one or more modules holding wakelock, then we abort sleep.

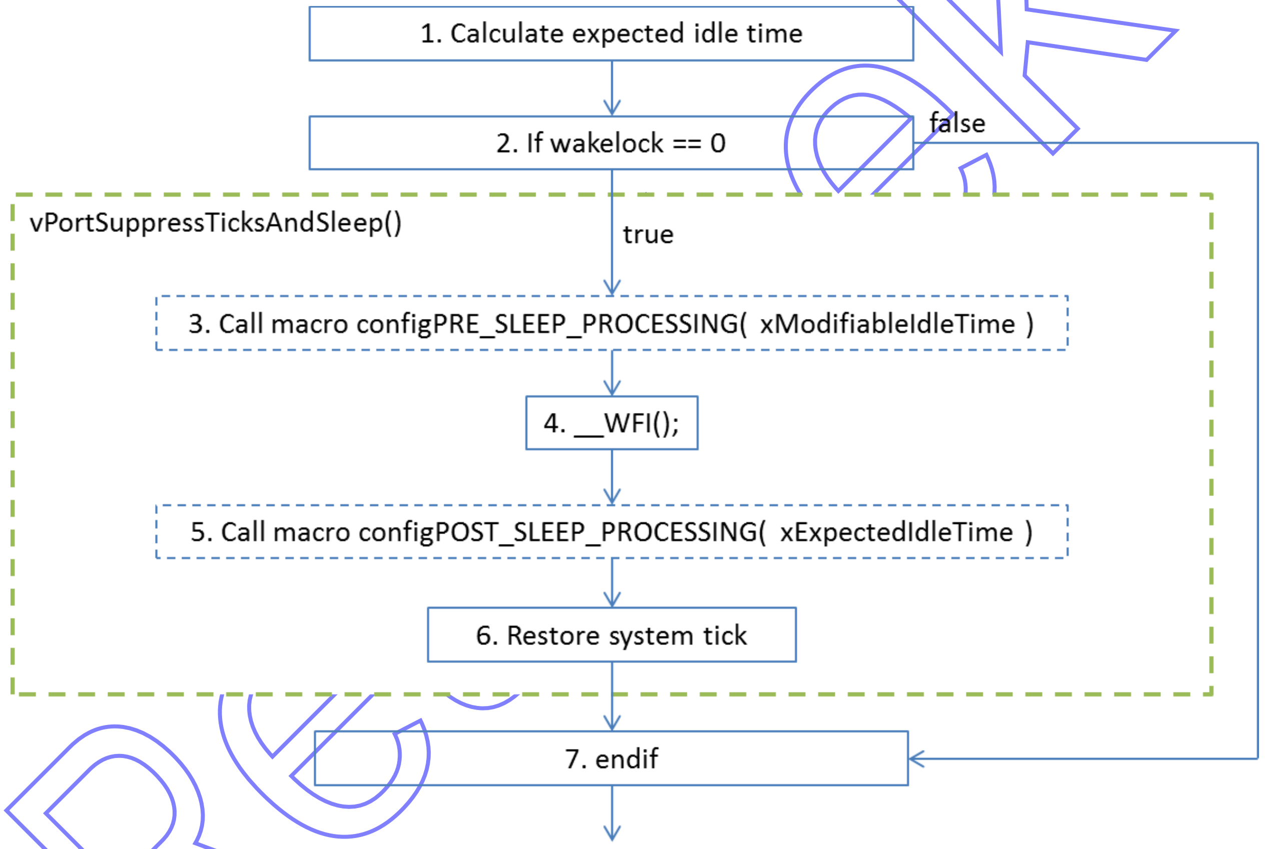
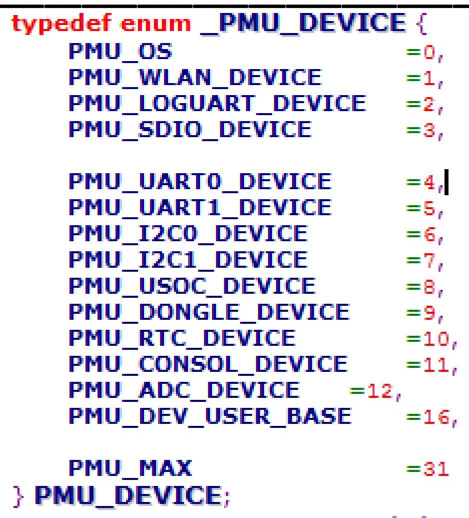

1. Calculate expected idle time
2. If wakelock == 0
false
vPortSuppressTicksAndSleep()
true
3. Call macro configPRE_SLEEP_PROCESSING(xModifiableIdleTime)
4. __WFI();
5. Call macro configPOST_SLEEP_PROCESSING(xExpectedIdleTime)
6. Restore system tick
7. endif

Figure 2 wakelock feature in FreeRTOS Tickless design

We can see step 2 and step 7 are modified as condition check for wakelock.

A wakelock bit map is for storing the wakelock status. Each module has its own bit in wakelock bit map. If the wakelock bit map equals to zero means there is no module holding wakelock. If the wakelock bit map larger than zero means there is some module holding wakelock. Bit 0~15 are reserved for ameba system usage, and user can use bit 16~31.



```
typedef enum _PMU_DEVICE {
    PMU_OS                  =0,
    PMU_WLAN_DEVICE         =1,
    PMU_LOGUART_DEVICE      =2,
    PMU_SDIO_DEVICE         =3,

    PMU_UART0_DEVICE        =4,
    PMU_UART1_DEVICE        =5,
    PMU_I2C0_DEVICE         =6,
    PMU_I2C1_DEVICE         =7,
    PMU_USOC_DEVICE         =8,
    PMU_DONGLE_DEVICE       =9,
    PMU_RTC_DEVICE          =10,
    PMU_CONSOL_DEVICE       =11,
    PMU_ADC_DEVICE      =12,
    PMU_DEV_USER_BASE       =16,

    PMU_MAX                 =31
} PMU_DEVICE;
```

Figure 3 wakelock bit map

1.3 Use Ameba sleep in tickless

By default FreeRTOS uses ARM WFI which only suppresses CPU tick. We can save more power consumption if we use Ameba sleep.

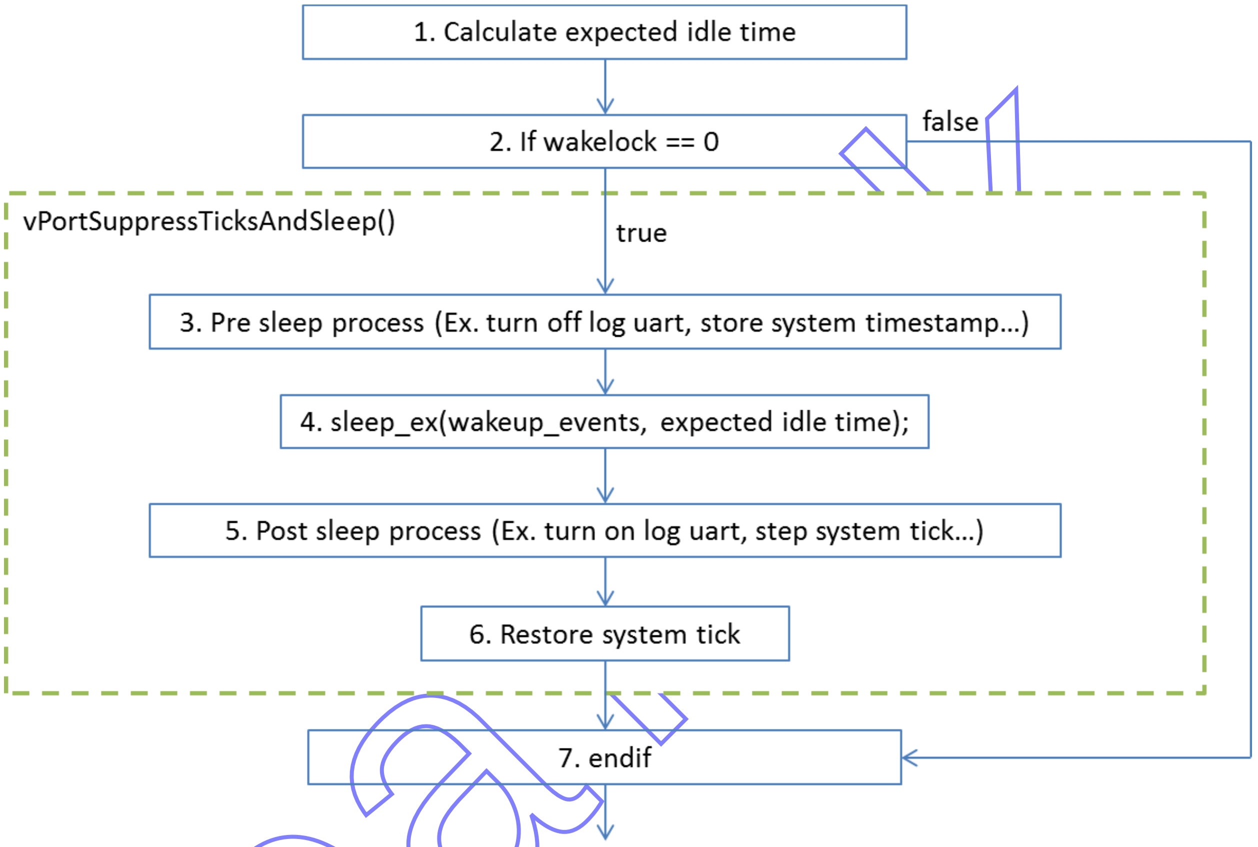

1. Calculate expected idle time
2. If wakelock == 0
false
vPortSuppressTicksAndSleep()
true
3. Pre sleep process (Ex. turn off log uart, store system timestamp…)
4. sleep_ex(wakeup_events, expected idle time);
5. Post sleep process (Ex. turn on log uart, step system tick…)
6. Restore system tick
7. endif

Figure 4 wakelock feature in FreeRTOS Tickless design

Step 3, 4, 5 are modified. In Step 3, we turn off peripherals like log uart. Please note that CPU tick won't update during sleep because we close CPU. It means software timer may works abnormal if we don't update system tick. So we store system timestamp which get from "us_ticker_api.h".

Step 4 performs the sleep. System sleeps with expected idle time if there is no other interrupt wake up system. The wakeup events include most events include system timer, gtimer, gpio interrupt, wlan protocol interrupt.

In Step 5 we turn on peripherals like log uart. And we check how much time passes, and update system tick accordingly.

1.4 Wakelock AT command

We provide AT command to use wakelock api. Below are the description

- **A**cquire wakelock
  ATSL=a[bitmap]
  Ex. ATSL=a[00010000], it acquires wakelock at bit 16

- **R**elease wakelock
  ATSL=r[bitmap]
  Ex. ATSL=r[00010000], it releases wakelock at bit 16

- Query wakelock status
  ATSL=?
  It print current wakelock bit map. We can use this command to debug why system doesn't enter sleep

AT command is sent through log uart. Log uart module acquire wakelock 10 seconds if user press "Enter", and release wakelock after 10 seconds (configurable).

If user want to enter several commands and don't want system enter tickless mode. He can follow below scenario:

1. ATSL=a[00010000]
2. Send desired several AT commands
3. ATSL=r[00010000]

2 Put WLAN into tickless design

In IEEE 802.11 power save management, it allows station enter their own sleep state. It defines station need keep awake in certain timestamp and may enter sleep state otherwise.

Wlan driver acquire wakelock to avoid system enter sleep in tickless design when wlan need keep awake. And it release wakelock when it is permitted to enter sleep state.

Below sections describe IEEE 802.11 power save, and its implementation on Ameba include tickless design.

2.1 IEEE 802.11 power management

IEEE 802.11 power management allows station enter power saving mode. Station cannot receive any frames during power saving. Thus AP need buffers these frames and requires station periodically wakeup to check beacon which has information of buffered frames.

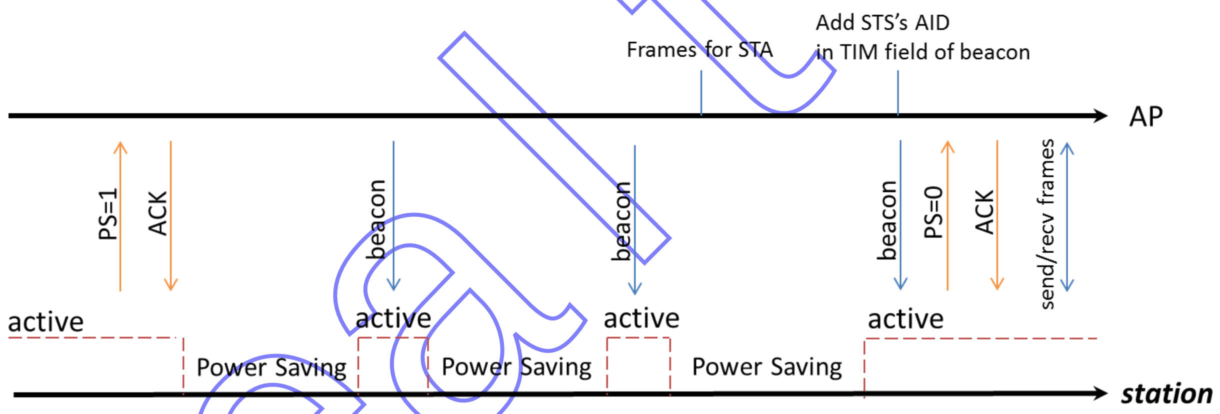


Figure 5 timeline of power saving

2.2 Ameba LPS

This feature is implemented in wlan driver. wlan driver enters LPS automatically without user application involved.

Ameba LPS (Leisure Power Save) implements IEEE 802.11 power management. Wlan driver enters LPS if flowing criteria meets:

(i) TX + RX packets count <= 8 in 2 seconds

(ii) RX packets count <= 2 in 2 seconds

It is checked in traffic status watch dog. The criteria are to keep high performance while traffic is busy. After enter LPS, there is PMU (Power Management Unit) control state machines.

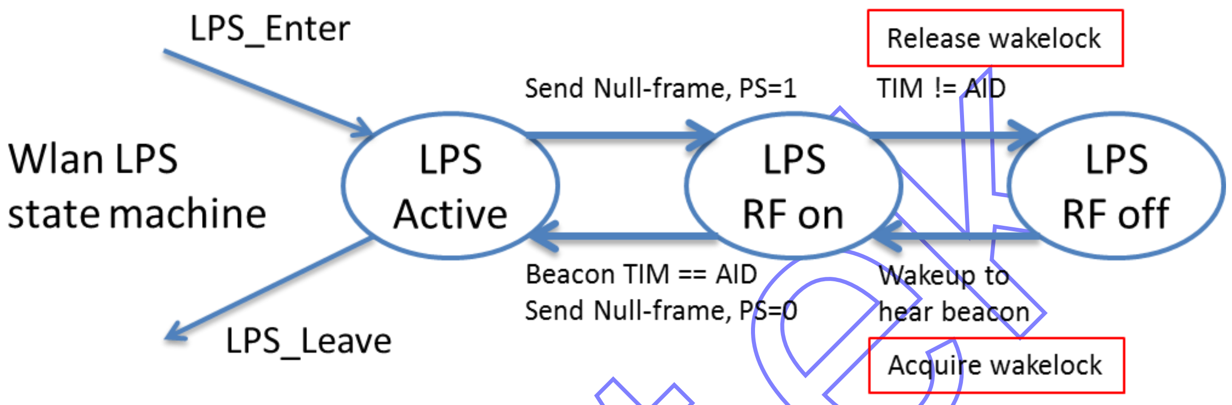


Figure 6 LPS state machine

2.3 Ameba IPS

This feature is implemented in wlan driver. Wlan driver enters IPS automatically without user application involved.

Ameba LPS is for situation that Ameba is associate to an AP. If Ameba is not associated to an AP, driver automatically turns off RF and other module to save power. Wlan Driver also releases wlan's wakelock at this time. When wlan driver needs to use RF related function, it automatically turns RF on and acquire wlan's wakelock. This scenario is called IPS (Inactive Power Save).

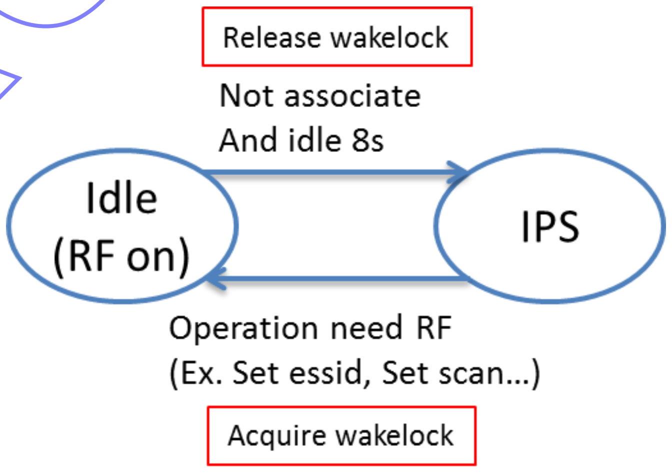


Figure 7 IPS state machine

3 Measure Power Consumption

3.1 Hardware preparation

In Ameba-Z reference board, there are other components that consume power. For example, there are cortex-M0 for DAP usage, LEDs, FT232, and capacitances. To measure power consumptions only for Ameba-Z, you need remove resistance at R43. And you can weld wires use J34:

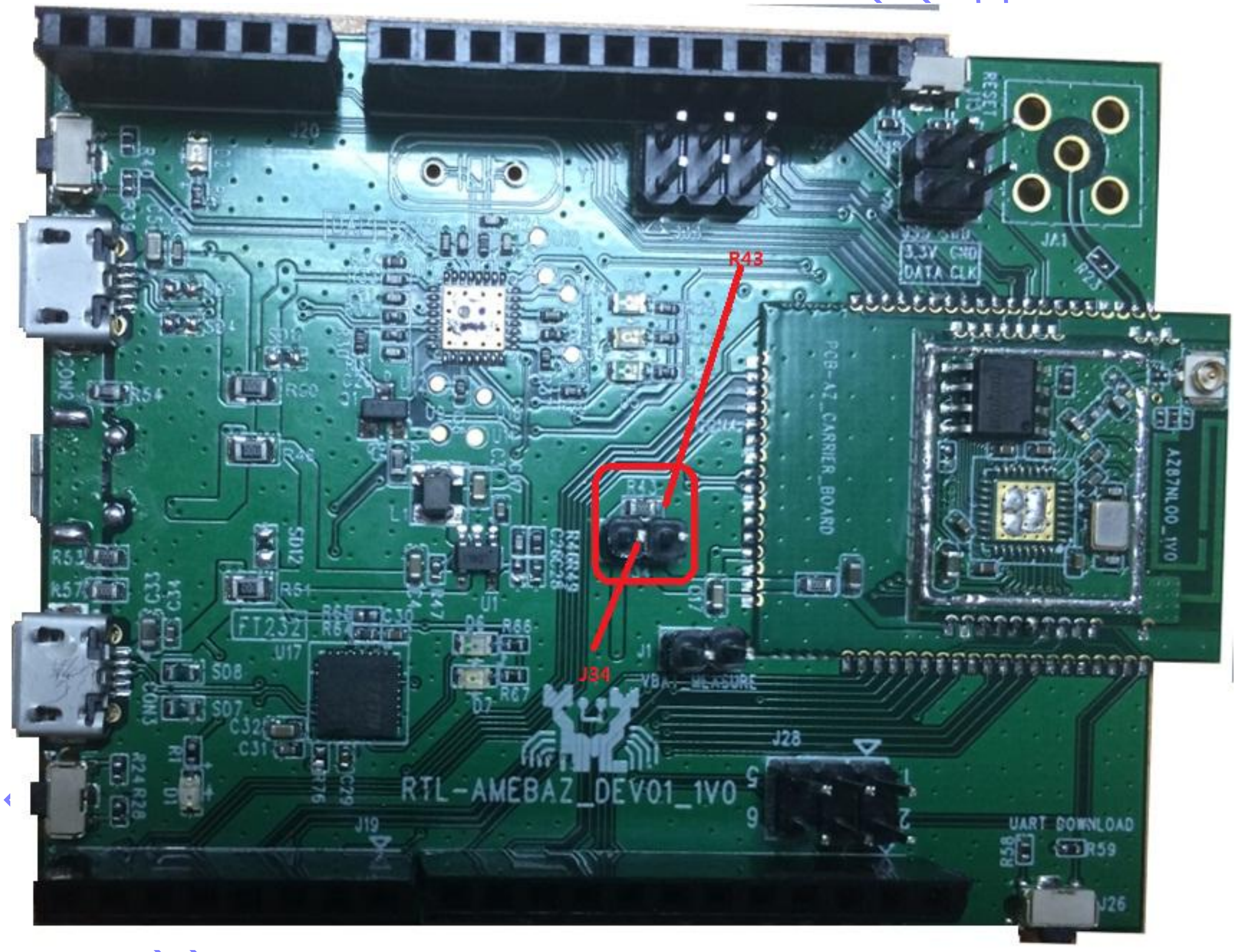


Figure 8 Power consumption measurement

You can use micro use to supply power for the board, and link current meter use J34 like following Figure:

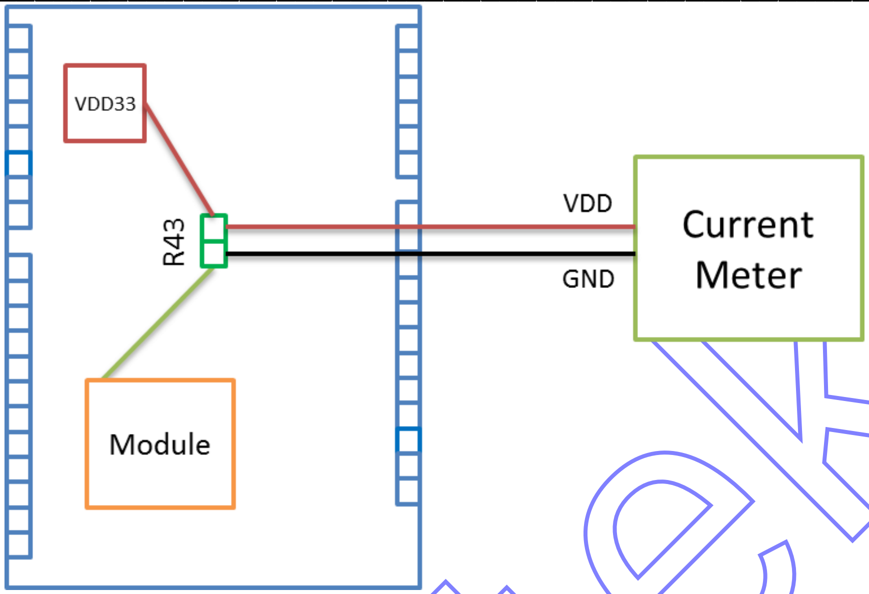


Figure 9 Measure power consumption from micro usb

3.2 Build SDK example

Example “**pm_tickless**” illustrate how to use wakelock in tickless mode and we can measure power consumption of wlan associate idle.

Below are suggesting operations to measure power consumption of wlan association idle:

1. When device boot up, **press several “Enter”** in log uart to avoid system enter tickless mode. Log uart module acquire wakelock 10 seconds at this moment.
2. Send AT command:

   ATW0=my_ap_name
   ATW1=my_ap_password
   ATWC

   Wait until wlan associate success.
3. CM4 will sleep again after 10 seconds
   It release user defined wakelock. Now system is under wlan association idle and tickless mode.