Low Power UART

# Introduction

The Talaria TWO device contains one high performance UART which can handle bitrates upto 2.4Mbps. This document describes using the UART with the help of UART APIs provided as part of the SDK.

Current scope of this document is limited to 2-wire UART with software flow control.

Graphical user interface, diagram, application

Description automatically generated

Figure : Talaria TWO device and HOST device serial communication diagram

The following sample application code and script are included with this application note:

|  |  |
| --- | --- |
| **File** | **Description** |
| uart\_tx.c | Sample Talaria TWO application describing the procedure to send data on serial port using UART |
| uart\_rx.c | Sample Talaria TWO application describing the procedure to read data on serial port using UART |
| uart\_wakeup.c | Sample Talaria TWO application describing sleep/wakeup procedure |
| uart\_wakeup.py | Sample HOST side implementation to wake Talaria TWO from sleep |

Table : List of files included in the application note

# Basic Usage of UART

This section provides a quick introduction to the UART APIs and describes communication with the host. For a comprehensive description of all the APIs, refer to section 7.

## Opening the Serial Port

Before opening a serial port, a handler must first be allocated.

|  |
| --- |
| struct uart \*handle; |

Once the UART handler is available, serial port can be opened using two different functions:

1. uart\_open – Opens serial port using default PINs and configured baud rate.

|  |
| --- |
| handle = uart\_open(baudrate); |

1. uart\_open\_ex – Opens the serial port using the configured baud rate, TXD, and RXD PINs.

|  |
| --- |
| #define TXD\_PIN 1  #define RXD\_PIN 2  …  int baudrate = 115200;  handle = uart\_open\_ex(baudrate, TXD\_PIN, RXD\_PIN, 0); |

Here, we configure the UART at a baud rate of 115200, GPIO1 as the TXD pin, and GPIO2 as the RXD pin.

## Closing the Serial Port

uart\_close is used to close the serial port.

|  |
| --- |
| uart\_close(handle); |

## Transmitting a Single Character

uart\_putchar is used to transmit a single character.

|  |
| --- |
| char c = ‘z’;  …  uart\_putchar(handle, c); |

## Transmitting a String

uart\_puts is used to transmit a string.

|  |
| --- |
| const char \*message = ”Hello”;  …  uart\_puts(handle, message); |

## Receiving (Blocking)

uart\_getchar is used to read a character from the serial port. This function call blocks until a byte is read.

|  |
| --- |
| int c = uart\_getchar(handle); |

## Receiving (Blocking with Timeout)

uart\_getchar\_tmo is used to read a character from the serial port with a timeout. This function call blocks until a byte is read or it times out.

|  |
| --- |
| int c = uart\_getchar\_tmo(h, 1000000); //1s sec timeout  if (c == -1) //uart\_getchar\_tmo returns -1 on timeout  break;  else  os\_printf("%d\_", c); |

## Register Event Callback (called when Talaria TWO wakes from sleep)

|  |
| --- |
| static void \_\_irq  handle\_event(struct uart \*u, enum uart\_event type, void \*priv)  {  //do stuff here  }  …  int main()  {  …  uart\_set\_event\_callback(handle, handle\_event, NULL);  } |

## Register Callback (called when special character is received)

|  |
| --- |
| const char special = ‘z’;  …  static void uart\_cb(void)  {  //do stuff here  }  …  int main()  {  uart\_set\_callback(handle, uart\_cb, special);  } |

# Recommended Protocol for 2-wire UART

Since 2-wire UART has no hardware flow control, a software protocol is recommended in this section.

## Power-ON

1. After powering ON Talaria TWO, UART initialization is completed with required baud rate.
2. Talaria TWO then sends a known byte pattern to the host indicating the readiness for UART transactions as illustrated in Figure 2.

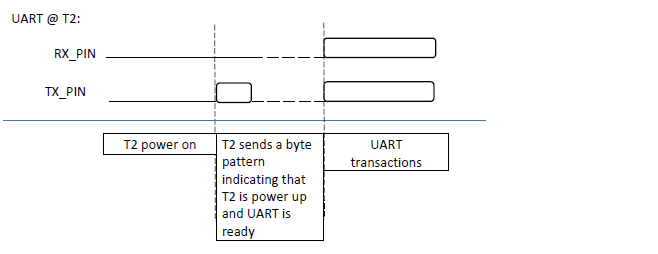


Figure : Recommended Talaria TWO protocol for power-ON case

# UART APIs

1. uart\_open – Opens UART port.
2. uart\_close – Closes UART port.
3. uart\_getchar - Reads the next character from UART. If no character is available, the function will block the calling thread until a character arrives in UART.
4. uart\_getchar\_tmo – Reads the next character from the UART. If no character is available, the function will block the calling thread until either timeout occurs or a character arrives in UART.
5. uart\_putchar – Write one character to UART.
6. uart\_puts - Write a string to UART.
7. uart\_flush – Flushes the output buffer.
8. uart\_open\_ex – Opens UART port with extended parameters.
9. uart\_set\_event\_callback – Enables callback on events described by the enum uart\_event.
10. uart\_set\_callback – Enables callback, when a certain character is entering the UART. This callback is executed in interrupt context.
11. uart\_suspend\_enable – Enables sleep mode.
12. os\_gpio\_request – Allocates and sets up PINs as GPIO.
13. os\_gpio\_set\_output – Sets up GPIO PINs for output.
14. os\_gpio\_set\_pin – Sets the specified GPIO PINs.
15. os\_gpio\_clr\_pin – Clears the specified GPIO PINs.
16. uart\_getchar\_tmo – Reads one character from the serial port, with timeout.
17. uart\_putchar – Writes one character to the serial port.
18. os\_msg\_release – Frees an allocated message.
19. os\_join\_thread – Waits for a thread to terminate and destroy the thread.

## UART-RX

### Code walkthrough

uart\_rx.c is used to read data on serial port using UART. This receives the character sent from the host device to Talaria TWO through the UART. Data will be printed on the console which can be used to verify the working of the application.

struct uart is an opaque object representing the UART. This function will initialize and allocate buffer space for the UART functions.

|  |
| --- |
| struct uart \*handle; |

The function will return a pointer to an opaque object which is used as a handle in subsequent calls to the UART functions. It opens the serial port using baud rate 921600.

|  |
| --- |
| handle = uart\_open(921600);  if (handle == NULL)  return 1; |

uart\_puts() function writes the null terminated string pointed to by str to the serial port. The terminating null byte ('\0') will not be written.

|  |
| --- |
| int c = uart\_getchar\_tmo(handle, 1000000); |

uart\_getchar\_tmo() function reads the next character from the serial port. If no character is available, the function will block the calling thread with timeout. Here, the timeout is 1 second.

|  |
| --- |
| if(c !=-1)  os\_printf("%c", c); |

uart\_close() function closes the serial port and releases the memory claimed by the uart\_open() call. Any characters left in the output buffers will be discarded.

|  |
| --- |
| uart\_close(handle); |

### Running the Application

Program uart\_rx.elf (*sdk\_x.y\examples\lp\_uart\bin*) using the Download tool:

1. Launch the Download tool provided with InnoPhase Talaria TWO SDK.
2. In the GUI window:
   1. Boot Target: Select the appropriate EVK from the drop-down
   2. ELF Input: Load the uart\_rx.elf by clicking on Select ELF File.
   3. Programming: Prog RAM or Prog Flash as per requirement.

For more details on using the Download tool, refer to the document: UG\_Download\_Tool.pdf (path: *sdk\_x.y\pc\_tools\Download\_Tool\doc*).

**Note**: x and y refer to the SDK release version. For example: *sdk\_2.5\doc*.

### Expected Output

uart\_rx console output:

|  |
| --- |
| UART:NWWWWAEBuild $Id: git-f92bee540 $  $App:git-7bdfd62  SDK Ver: sdk\_2.4  Lpuart Rx Demo App |

Open a minicom terminal with a baud rate of 921600, with no hardware flow control and add the serial device.

Once the minicom starts, reset Talaria TWO which displays the output string.

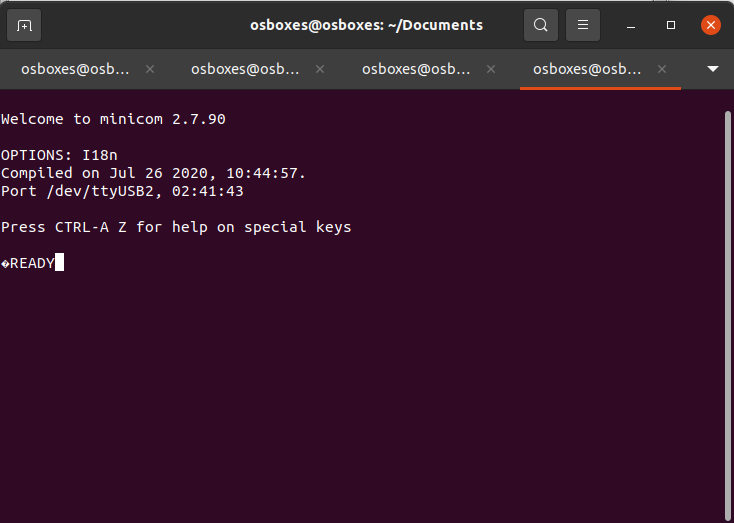


Figure : UART - RX Minicom Output

## UART-TX

### Code walkthrough

The uart\_tx.c is used to send data on serial port using UART. This transmits a string of data "Hello world" from Talaria TWO to host through the UART.

The struct uart opaque object represents the UART. This function will initialize and allocate buffer space for the UART functions.

|  |
| --- |
| struct uart \*handle; |

The function will return a pointer to an opaque object that is used as a handle in subsequent calls to the UART functions. It opens the serial port using baud rate 921600.

|  |
| --- |
| handle = uart\_open(921600);  if (handle == NULL)  return 1; |

This function will write the null terminated string pointed to by str to the serial port. The terminating null byte ('\0') will not be written.

|  |
| --- |
| uart\_puts(handle, "Hello World\n"); |

uart\_flush()calls the thread until all characters currently buffered for output have been written to the serial port.

|  |
| --- |
| uart\_flush(handle); |

The uart\_close() function closes the serial port and releases the memory claimed by the uart\_open() call.

|  |
| --- |
| uart\_close(handle); |

### Running the Application

Program uart\_tx.elf (*sdk\_x.y\examples\lp\_uart\bin*) using the Download tool:

1. Launch the Download tool provided with InnoPhase Talaria TWO SDK.
2. In the GUI window:
   1. Boot Target: Select the appropriate EVK from the drop-down
   2. ELF Input: Load the uart\_tx.elf by clicking on Select ELF File.
   3. Programming: Prog RAM or Prog Flash as per requirement.

For more details on using the Download tool, refer to the document: UG\_Download\_Tool.pdf (path: *sdk\_x.y\pc\_tools\Download\_Tool\doc*).

**Note**: x and y refer to the SDK release version. For example: *sdk\_2.5\doc*.

### Expected Output

uart\_tx console output:

|  |
| --- |
| UART:NWWWWAEBuild $Id: git-f92bee540 $  $App:git-7bdfd62  SDK Ver: sdk\_2.4  Lpuart Tx Demo App |

Open a Minicom terminal with a baud rate of 921600, with no Hardware flow control and add the serial device.

Once the minicom starts, reset the Talaria TWO device which displays the output string.

Text

Description automatically generated

Figure : UART – TX Minicom Output

## UART Suspend/Wakeup

### Code Walkthrough

uart\_wakeup.c

The sample application uart\_wakeup.c shows a basic implementation of UART suspend/wakeup using UART\_BREAK signal.

This section describes the state of the UART, whether the link is up or down.

|  |
| --- |
| enum uart\_link\_state {  UART\_LINK\_UP = 0,  UART\_LINK\_DOWN,  } link |

handle\_event() handles the UART events and checks if the type is UART\_EVENT\_WAKEUP. If yes, then the uart\_status will be set to UART\_WAKE until suspend state is enabled.

|  |
| --- |
| ;  static void \_\_irq  handle\_event(struct uart \*h, enum uart\_event type, void \*priv)  {  if (type == UART\_EVENT\_WAKEUP)  {  pr\_always("UART\_EVENT\_WAKEUP\n");  uart\_status=UART\_WAKE;  struct os\_msg \*m = os\_msg\_alloc(APP\_MSG\_AWAKE\_UART, sizeof(\*m));  os\_sendmsg(app\_thread, m);  }  } |

os\_gpio\_request() API requests the pins and configures the pin as GPIO, os\_gpio\_set\_output() API configures the GPIO pins as digital output pins, os\_gpio\_clr\_pin() is used to clear the specified GPIO pins, os\_gpio\_set\_pin() is used to set the GPIO high and os\_gpio\_clr\_pin() is used to set GPIO low.

|  |
| --- |
| void\* app\_main(void \*arg)  {  struct os\_msg \*m;  os\_gpio\_request(sig\_pin);  os\_gpio\_set\_output(sig\_pin);  uart\_suspend\_enable(u);  while (1) {  os\_gpio\_set\_pin(sig\_pin);  /\* blocking. \*/  m = os\_recvmsg(false);  os\_gpio\_clr\_pin(sig\_pin);  if(m->msg\_type==APP\_MSG\_AWAKE\_UART)  {  if(uart\_status != UART\_INACTIVE)  {  int c = uart\_getchar\_tmo(u, UART\_TMO\_CLR); // read and drop garbage byte  rx\_ready = true;  uart\_putchar(u, 'R'); // can be used for syncing    while(rx\_ready)  {  c = uart\_getchar\_tmo(u, UART\_TMO\_FIRST\_BYTE); //first byte may take a while depending on when the event gets triggered  while(c!=-1)  {  /\*\*  \* We look for '\*' to signify end of message.  \* Ideally the HOST and T2 would use an agreed upon protocol to determine length and validity of packet from HOST  \*\*/  if((char)c == '\*')  {  uart\_putchar(u, 'A');    }  c = uart\_getchar\_tmo(u, UART\_TMO\_BYTE);  }  // process message here    // wait and check if HOST plans to send more packets  c = uart\_getchar\_tmo(u, UART\_TMO\_EXTENDED); //should catch the break  if(c == -1)  {  //no additional packet/s  rx\_ready = false;  }  }  uart\_putchar(u, 'S');  pr\_always("uart\_suspend\_enable\n");  uart\_suspend\_enable(u);  uart\_status=UART\_INACTIVE;  }  }  os\_msg\_release(m);    }  return NULL;  } |

wcm\_create()API from the Wi-Fi Connection Manager is used to connect to a Wi-Fi network. Initially, the Wi-Fi network interface is created using wcm\_create().

wifi\_connect\_to\_network()API, from components library, connects to the Wi-Fi network using the AP credentials provided.

|  |
| --- |
| /\*Create a Wi-Fi network interface\*/  wcm\_handle = wcm\_create(NULL);  /\* Read the configuration and connect to desired network \*/  rval = wifi\_connect\_to\_network(&wcm\_handle, WCM\_CONN\_WAIT\_INFINITE, &wcm\_connected);  if(rval < 0) {  os\_printf("\nError: Unable to connect to network\n");  return 0;  }  if(wcm\_connected != true) {  os\_printf("\nCouldn't Connect to network ");  wcm\_disconnect(wcm\_handle);  } |

uart\_wakeup\_host.c

The sample application uart\_wakeup\_host.c shows implementation of a HOST MCU which can wakeup Talaria TWO using UART.

A serial port with a baud rate of 115200 is opened using uart\_open().

**Single Packet Test**:

Single packet test prints the errors along with the 1000 packets.

Pins and GPIOs are used to emulate a long UART\_BREAK signal. Here uart\_getchar\_tmo() reads one character “R” from the serial port which indicates Ready. If no character is available, the function will block the calling thread until timeout occurs.

Host will again wait for Talaria TWO to echo back the last end sync character which is the acknowledgement “A” packet.

|  |
| --- |
| u = uart\_open(115200);  int count = 0;  int errors = 0;  int packets = 1000;  char count\_buffer[32];  bool retransmit = false;  os\_printf("Single packet test\n");  while(count < packets)  {  //Use pin ad GPIO to emulate a long UART\_BREAK signal  int tx\_pin = 1 << TX\_PIN;  os\_gpio\_set\_mode(tx\_pin, GPIO\_MODE);  os\_gpio\_request(tx\_pin);  os\_gpio\_set\_output(tx\_pin);  os\_gpio\_clr\_pin(tx\_pin);  os\_sleep\_us(1100, OS\_TIMEOUT\_NO\_WAKEUP);  os\_gpio\_set\_pin(tx\_pin);  os\_gpio\_set\_input(tx\_pin);  os\_gpio\_set\_mode(tx\_pin, GPIO\_FUNCTION\_MODE);  os\_gpio\_free(tx\_pin);    while(1)  {  //Wait for ready 'r' from T2  int sync = uart\_getchar\_tmo(u, 1000);  if(sync == -1)  {  break;  }  else if((char)sync == 'R')  {  if(retransmit)  {  errors++;  os\_printf("retransmitting!!!\n");  }  memset(count\_buffer, 0, 32);  sprintf(count\_buffer, "count: %d\n", count);  uart\_write((void\*)msg1, strlen(msg1));  uart\_write((void\*)msg2, strlen(msg2));  uart\_write((void\*)count\_buffer, strlen(count\_buffer));  uart\_write((void\*)msg4, strlen(msg4)); //send end sync char '\*'  uart\_flush(u);  //Wait for T2 to echo back the last end sync char '\*'  sync = uart\_getchar\_tmo(u, 2000);  retransmit = true;  while(sync != -1)  {  if(sync == 'A')  {  retransmit = false;  count++;  break;  }  sync = uart\_getchar\_tmo(u, 2000);  }  break;  }  }  if(!retransmit)  {  os\_sleep\_us(50000, OS\_TIMEOUT\_NO\_WAKEUP);  }  }  os\_printf("%d errors to send %d packets\n", errors, packets); |

**Multiple Packet Test**:

Multiple packet test prints the errors along with the 1000 packets.

PINs and GPIOs are used to emulate a long UART\_BREAK signal. Here uart\_getchar\_tmo() reads one character “R” from the serial port which indicates Ready. If no character is available, the function will block the calling thread until timeout occurs.

Host will again wait for Talaria TWO to echo back the last end sync character which is the acknowledgement “A” packet.

|  |
| --- |
| count = 0;  os\_printf("Multi packet test\n");  while(count < packets)  {  //Use pin ad GPIO to emulate a long UART\_BREAK signal  int tx\_pin = 1 << TX\_PIN;  os\_gpio\_set\_mode(tx\_pin, GPIO\_MODE);  os\_gpio\_request(tx\_pin);  os\_gpio\_set\_output(tx\_pin);  os\_gpio\_clr\_pin(tx\_pin);  os\_sleep\_us(1100, OS\_TIMEOUT\_NO\_WAKEUP);  os\_gpio\_set\_pin(tx\_pin);  os\_gpio\_set\_input(tx\_pin);  os\_gpio\_set\_mode(tx\_pin, GPIO\_FUNCTION\_MODE);  os\_gpio\_free(tx\_pin);    while(1)  {  //Wait for ready 'r' from T2  int sync = uart\_getchar\_tmo(u, 1000);  if(sync == -1)  {  break;  }  else if((char)sync == 'R')  {  if(retransmit)  {  errors++;  os\_printf("retransmitting!!!\n");  }  memset(count\_buffer, 0, 32);  sprintf(count\_buffer, "count: %d\n", count);  uart\_write((void\*)msg1, strlen(msg1));  uart\_write((void\*)msg2, strlen(msg2));  uart\_write((void\*)count\_buffer, strlen(count\_buffer));  uart\_write((void\*)msg4, strlen(msg4)); //send end sync char '\*'  uart\_flush(u);  //Wait for T2 to echo back the last end sync char '\*'  sync = uart\_getchar\_tmo(u, 2000);  retransmit = true;  while(sync != -1)  {  if(sync == 'A')  {  retransmit = false;  count++;  break;  }  sync = uart\_getchar\_tmo(u, 2000); } |

Using PINs and GPIOs emulate a long UART\_BREAK signal and send the second packet before Talaria TWO UART goes into suspend state. uart\_flush() and uart\_close() are used to empty the output buffers before closing.

|  |
| --- |
| os\_sleep\_us(5000, OS\_TIMEOUT\_NO\_WAKEUP);  // send second packet before T2 UART goes into suspend  //Use pin ad GPIO to emulate a long UART\_BREAK signal  int tx\_pin = 1 << TX\_PIN;  os\_gpio\_set\_mode(tx\_pin, GPIO\_MODE);  os\_gpio\_request(tx\_pin);  os\_gpio\_set\_output(tx\_pin);  os\_gpio\_clr\_pin(tx\_pin);  os\_sleep\_us(1100, OS\_TIMEOUT\_NO\_WAKEUP);  os\_gpio\_set\_pin(tx\_pin);  os\_gpio\_set\_input(tx\_pin);  os\_gpio\_set\_mode(tx\_pin, GPIO\_FUNCTION\_MODE);  os\_gpio\_free(tx\_pin);      memset(count\_buffer, 0, 32);  sprintf(count\_buffer, "count: %d\n", count);  uart\_write((void\*)msg1, strlen(msg1));  uart\_write((void\*)msg2, strlen(msg2));  uart\_write((void\*)count\_buffer, strlen(count\_buffer));  uart\_write((void\*)msg4, strlen(msg4)); //send end sync char '\*'  uart\_flush(u);  //Wait for T2 to echo back the last end sync char '\*'  sync = uart\_getchar\_tmo(u, 2000);  retransmit = true;  while(sync != -1)  {  if(sync == 'A')  {  retransmit = false;  count++;  break;  }  sync = uart\_getchar\_tmo(u, 2000);  }  break;  }  }  if(!retransmit)  {  os\_sleep\_us(50000, OS\_TIMEOUT\_NO\_WAKEUP);  }  }  os\_printf("%d errors to send %d packets\n", errors, packets);  /\* Make sure the output buffers are empty before closing. \*/  uart\_flush(u);  uart\_close(u);  return 0;  } |

### Hardware Connection

Using two Talaria TWO boards and a logic analyzer make the connection using the required wires and breadboard as shown in Figure 5.

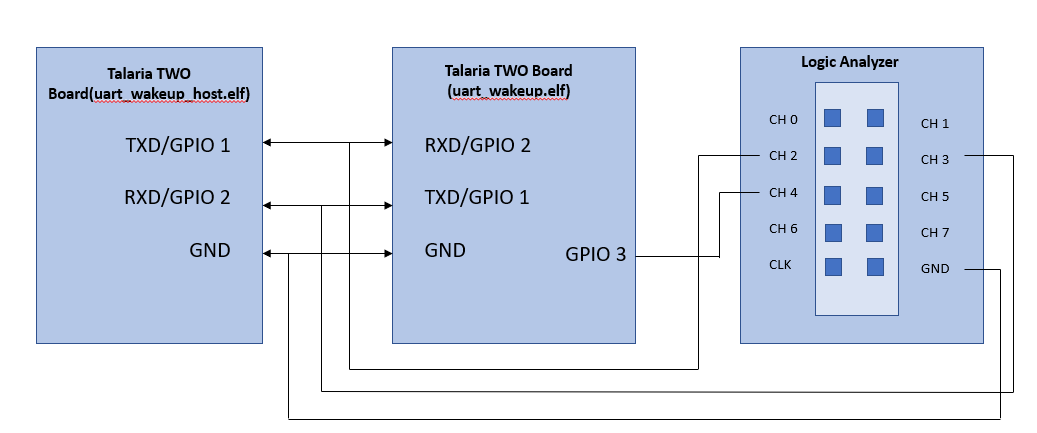


Figure : UART suspend/wakeup - hardware connection

### Running the Application

Program uart\_wakeup\_host.elf and uart\_wakeup.elf (*sdk\_x.y\examples\lp\_uart\bin*) onto Talaria TWO boards using the Download tool:

1. Launch the Download Tool provided with InnoPhase Talaria TWO SDK.
2. In the GUI window:
   1. Boot Target: Select the appropriate EVK from the drop-down
   2. ELF Input: Load uart\_wakeup\_host.elf and uart\_wakeup.elf by clicking on Select ELF File.
   3. AP Options: Pass the appropriate SSID and passphrase to connect to an Access Point for uart\_wakeup.elf.
   4. Programming: Prog RAM or Prog Flash as per requirement.

For more details on using the Download Tool, refer to the document: UG\_Download\_Tool.pdf (path: *sdk\_x.y\pc\_tools\Download\_Tool\doc*).

**Note**: x and y refer to the SDK release version. For example: *sdk\_2.5\doc*.

### Expected output

Once the hardware connection is complete as described in section 7.3.2, flash Talaria TWO boards with uart\_wakeup\_host.elf and uart\_wakeup.elf . Simultaneously reset on Talaria TWO’s GUI or press the hard reset button on Talaria TWO boards and observe the following console outputs:

uart\_wakeup.elf:

|  |
| --- |
| Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PNWWWWWAEBuild $Id: git-ba65998b7 $  np\_conf\_path=/data/nprofile.json ssid=InnoPhase passphrase=43083191  Sleep wakeup Test  $App:git-58d3fe67  SDK Ver: sdk\_2.5  UART Wakeup  addr e0:69:3a:00:13:90  Connecting to added network : InnoPhase  [0.770,776] CONNECT:00:5f:67:cd:c5:a6 Channel:11 rssi:-67 dBm  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS  [1.618,831] MYIP 192.168.0.106  [1.618,994] IPv6 [fe80::e269:3aff:fe00:1390]-link  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_CONNECTED  Connected to added network : InnoPhase  os\_suspend\_enable  [218.009,847] UART\_EVENT\_WAKEUP  [218.021,902] uart\_suspend\_enable  [218.061,961] UART\_EVENT\_WAKEUP  [218.074,016] uart\_suspend\_enable  [218.583,219] UART\_EVENT\_WAKEUP  [218.595,274] uart\_suspend\_enable  [218.635,338] UART\_EVENT\_WAKEUP  [218.647,393] uart\_suspend\_enable  [218.687,494] UART\_EVENT\_WAKEUP  [218.699,549] uart\_suspend\_enable  [218.739,607] UART\_EVENT\_WAKEUP  [218.751,662] uart\_suspend\_enable  [218.791,726] UART\_EVENT\_WAKEUP  [218.803,781] uart\_suspend\_enable  [218.843,845] UART\_EVENT\_WAKEUP  [218.855,901] uart\_suspend\_enable  [218.895,994] UART\_EVENT\_WAKEUP  [218.908,050] uart\_suspend\_enable  [218.948,114] UART\_EVENT\_WAKEUP  [218.960,169] uart\_suspend\_enable  [219.000,232] UART\_EVENT\_WAKEUP  [219.012,287] uart\_suspend\_enable  [219.052,351] UART\_EVENT\_WAKEUP  [219.064,406] uart\_suspend\_enable  [219.104,500] UART\_EVENT\_WAKEUP  [219.116,555] uart\_suspend\_enable  [219.156,619] UART\_EVENT\_WAKEUP  [219.168,674] uart\_suspend\_enable  [219.208,738] UART\_EVENT\_WAKEUP  [219.220,793] uart\_suspend\_enable  [219.260,857] UART\_EVENT\_WAKEUP  [219.272,912] uart\_suspend\_enable  [219.365,156] UART\_EVENT\_WAKEUP  [219.377,211] uart\_suspend\_enable  [219.417,245] UART\_EVENT\_WAKEUP  [219.429,300] uart\_suspend\_enable  [219.469,365] UART\_EVENT\_WAKEUP  [219.481,420] uart\_suspend\_enable |

uart\_wakeup\_host.elf:

|  |
| --- |
| Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PNWWWAEBuild $Id: git-ba65998b7 $  Single packet test  0 errors to send 1000 packets  Multi packet test  0 errors to send 1000 packets |

Following output is observed on the logic analyzer:

The UART interface on Talaria TWO can be placed in suspend mode while there is no UART activity between Talaria TWO and Host.   
To wake up Talaria TWO from sleep and re-enable the UART interface, the Host needs to send the UART\_BREAK signal for a minimum duration of 1ms.

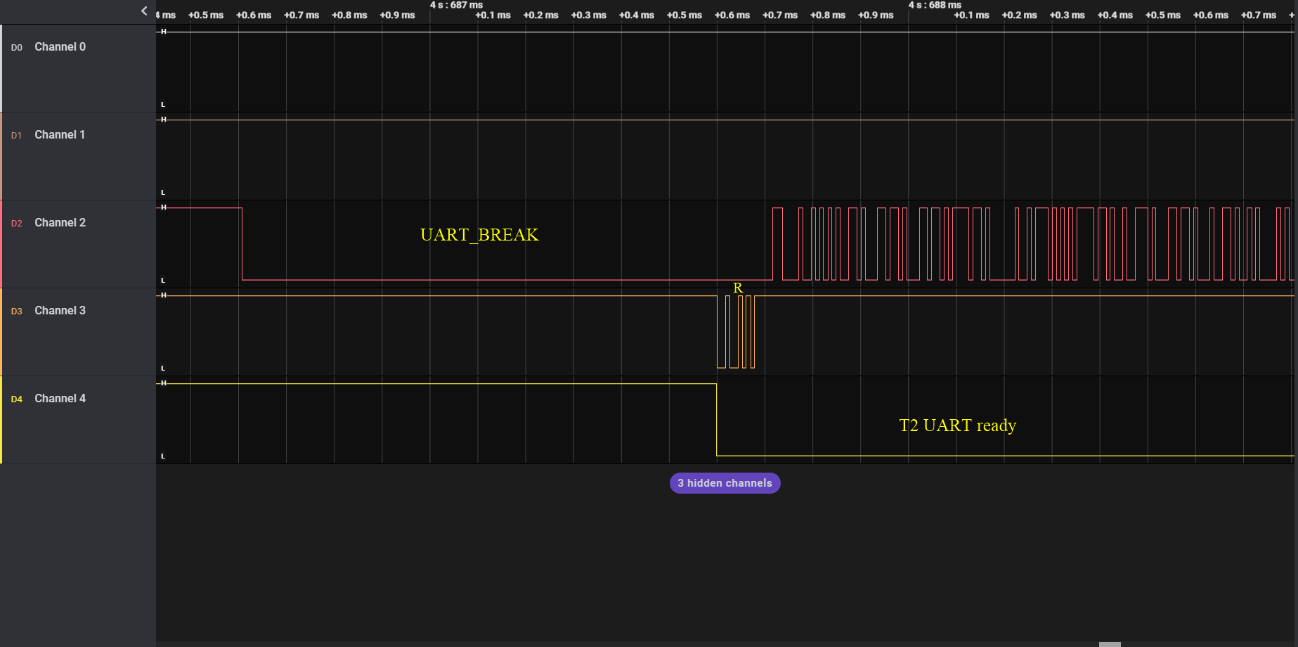


Figure : UART suspend/wakeup - Wakeup from UART\_BREAK

uart\_wakeup example follows the subsequent protocol:

1. HOST sends break.
2. Talaria TWO wakes up and takes UART out of SUSPEND. Talaria TWO sends a byte ‘R’ to inform the Host that UART interface is ready.
3. Host (optionally) waits for ~1ms or ‘R’ and then sends data.
4. Talaria TWO receives the data.

Note: Talaria TWO performs error checking and lets the Host know if re-transmit is required.

* 1. Talaria TWO can determine whether the Host is done sending either by a timeout or similar protocol.
  2. Talaria TWO sends an acknowledgement byte ‘A’ after validating the packet.
  3. Talaria TWO optionally waits (~10 ms) for more incoming packets.

1. Talaria TWO sends a byte ‘S’ to the Host to inform the Host that the UART is entering SUSPEND state.



Figure : Wakeup from UART\_BREAK and IPC communication with HOST