GPIO

# Introduction

This application note describes the usage of GPIO pins on Talaria TWO module. Talaria TWO module allows up to 11 GPIOS to be configured as digital I/O pins.

# Demo Applications

## EVB-A Hardware Configuration

The demo application utilizes the LED D1 on the Talaria TWO EVB-A board. Use jumper on J3 header to connect LED D1 to GPIO14.

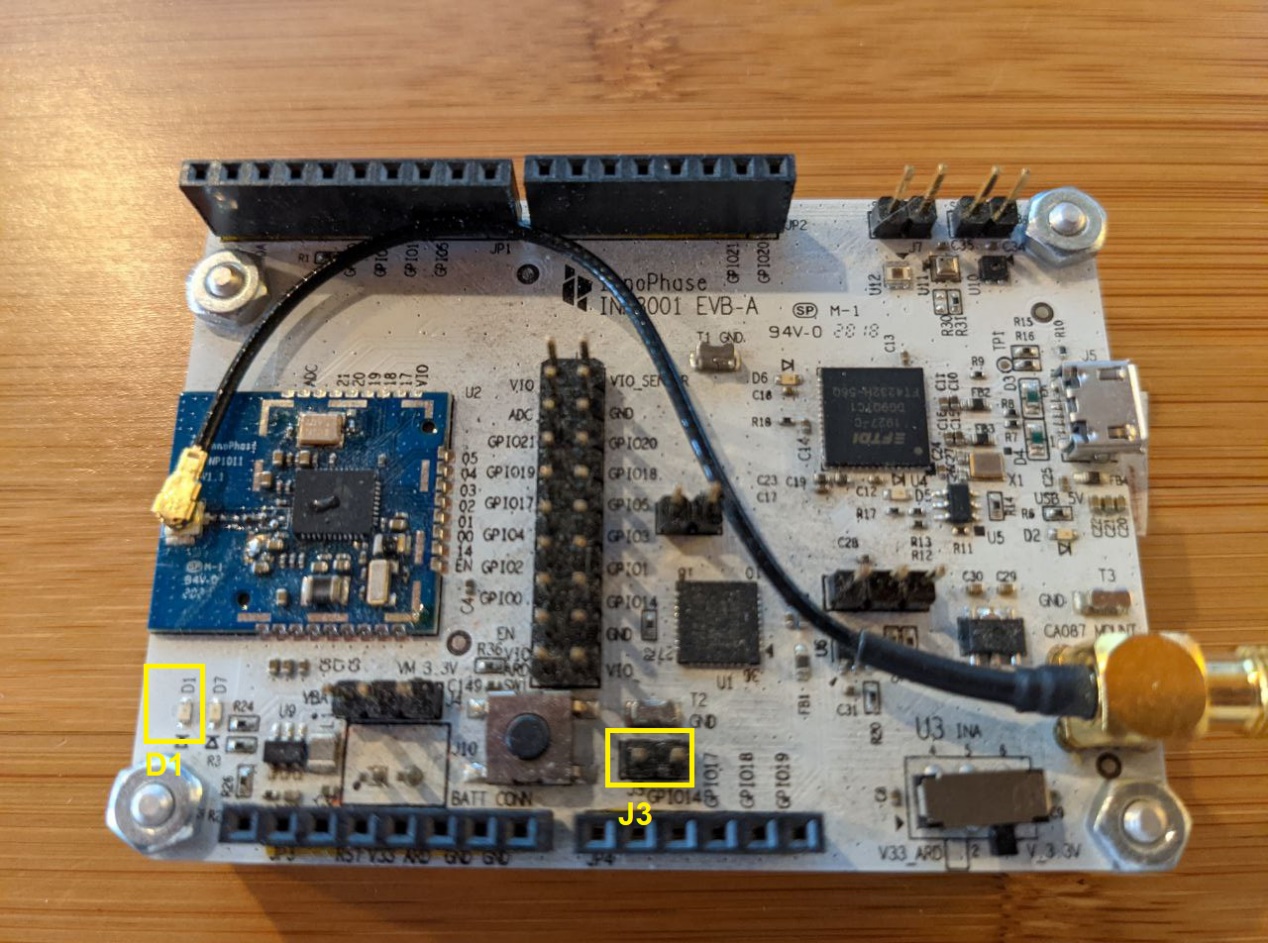


Figure : Talaria TWO EVB-A

## Building

To build the sample applications, execute the following commands from the SDK directory:

|  |
| --- |
| cd examples/gpio  make |

The make command generates the following files in the out directory:

1. gpio\_blink.elf
2. gpio\_suspend.elf
3. gpio\_interrupt\_awake.elf
4. gpio\_suspend\_mask.elf

# GPIO Demo APIs

1. os\_sem\_post(): Unlocks semaphore.

Increments the value of a semaphore and wakes the first thread waiting for this semaphore.

1. callout\_schedule\_at(): Schedules callback at absolute time.

This function is identical to callout\_schedule(), except that it takes an absolute time to specify when the callout expires.

1. os\_suspend\_enable(): Enables system suspend when idle.

Calling os\_suspend\_enable() will enable system suspend (deep sleep). If enabled, the system will go into a suspended mode when the processor is idle. Entering and leaving suspend mode takes additional time. Hence, enabling this will affect the real-time response of the system.

When in a suspended state, the system will still wake up if an interrupt occurs, but the latency will be much larger compared to running with suspend mode disabled.

1. os\_suspend\_disable(): Disables system suspend when idle.

When disabled, the system will never go into suspend mode. When the system is idle, the kernel puts the CPU into low-power mode, ready to swiftly resume execution if an interrupt occurs.

1. os\_gpio\_set\_irq\_level\_low(): Configures the GPIO input pin to signal interrupt on low level.
2. os\_gpio\_set\_irq\_level\_high(): Configures the GPIO input pin to signal interrupt on high level.
3. os\_gpio\_set\_value(os\_gpio\_set\_t gpios\_hi, os\_gpio\_set\_t gpios\_lo): Sets the value of GPIO pins.

This function is used to change the value of several GPIO pins at once. The pins in the gpio\_hi set will be driven to '1' and the pins in the gpios\_lo set will be set to level '0'.

Change of signals occur at the same time. This property might be important when "bit-banging" some wire protocol which requires two signals to change simultaneously.

Simpler functions for changing the output value: os\_gpio\_set\_pin, os\_gpio\_clr\_pin().

1. os\_sem\_wait(): Locks a semaphore.

If the value of the semaphore is greater than zero, decrement the counter. If the value is zero, it puts the current thread to sleep until the value becomes positive. This function cannot be used in interrupt context.

1. os\_gpio\_request(): Allocates and setup pins as GPIO.

Allocate the specified pins and configure them as GPIO. The pins are initially setup to be inputs. The direction of the pins can be changed using os\_gpio\_set\_output().

1. os\_gpio\_set\_input(): Sets up GPIO pins for input.

Configure the specified GPIO pins to be inputs.

1. os\_gpio\_set\_output(): Configures the specified GPIO pins to be output pins.
2. os\_gpio\_enable\_irq(): Enables the interrupt generation for the specified set of GPIO input pins.
3. os\_gpio\_set\_pull(): Configures the GPIO pull up/down, all pins except GPIO18 are pullup.
4. os\_gpio\_get\_value(): Reads the value of the specified GPIO pins.
5. os\_sem\_init(): Initializes a semaphore.

This must be performed before other operations. It is possible to statically initialize a semaphore with the OS\_SEM\_INITALIZER macro.

1. os\_create\_thread(): Creates a new thread.

This creates a new thread with priority specified in flags (via OS\_CRTHREAD\_PRIO macro). The thread is placed in the run queue, but there is no immediate reschedule. The thread continues to run until the entry point returns, at which point return value (a pointer) can be obtained with os\_join\_thread. If the return value is of no consequence, OS\_CRTHREAD\_DETACHED can be passed in flags. This causes the OS to reap the thread.

1. callout\_init(): Initializes a callout object.
2. os\_join\_thread(): Waits for a thread to terminate and destroy the thread.

Calling this function will suspend execution of the calling thread until the target thread exits. The memory used to hold the threads stack and control block is freed.

1. os\_gpio\_clr\_pin(): Clears the specified GPIO pins. Sets the output value of the GPIO pins '0'.
2. os\_gpio\_set\_pin(): Sets the specified GPIO pins. Sets the output value of the GPIO pins to '1'.
3. os\_gpio\_attach\_event(): Attaches an event handler function to a GPIO event.
4. os\_gpio\_detach\_event(): Detaches an attached event handler function from a GPIO event.

# Source Code Walkthrough

## GPIO Blink Demo Example

### Application Flow

The GPIO blink demo application demonstrates toggling a GPIO output and is connected to an LED on the Talaria TWO EVB-A.

Following steps describe the flow to toggle the GPIOs:

1. Allocates the specified pins and configures them as GPIO.
2. Sets up GPIO pins as output.
3. Toggles the GPIO to high and low.

### Sample Code Walkthrough

This section describes the procedure to configure the selected GPIO as digital I/O and toggle the LEDs connected to the selected GPIOs. The 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.

|  |
| --- |
| int led\_pin = 1 << LED\_PIN;  os\_gpio\_request(led\_pin);  os\_gpio\_set\_output(led\_pin);  os\_gpio\_clr\_pin(led\_pin); |

os\_gpio\_set\_pin() is used to set the GPIO high and os\_gpio\_clr\_pin() is used to set GPIO low. This is the main thread where the LED blinks every 500ms by toggling the LED to high and low.

|  |
| --- |
| os\_wait\_usec(500 \* 1000);  os\_gpio\_set\_pin(led\_pin);  os\_wait\_usec(500 \* 1000);  os\_gpio\_clr\_pin(led\_pin); |

### Running the Application

Program gpio\_blink.elf (sdk\_x.y\examples\gpio\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 gpio\_blink.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

Once the application is downloaded onto the Talaria TWO EVB-A, the LED D1 blinks every 500ms.

|  |
| --- |
| UART:NWWWWAE  Build $Id: git-b664be2af $  hio.baudrate=115200  flash: Gordon ready!  UART:NWWWAEBuild $Id: git-b664be2af $  Hello blinky |

## GPIO Suspend Demo Example

### Application Flow

The GPIO suspend demo application is to demonstrate configuring GPIO as an interrupt and is connected to an LED on the Talaria TWO EVB-A.

Following are the steps:

1. Initializes the semaphores, threads, and callouts.
2. Creates a scheduler.
3. If or else statement to put Talaria TWO into sleep or awake state by os\_suspend\_enable or disable APIs.
4. Worker thread used as a background task.

### Sample Code Walkthrough

This section describes the function definition for scheduler, os\_sem\_post() will increment the value of semaphores and wakes up the first thread, callout\_scedule\_at() will check for the absolute time to specify when the callout expires.

|  |
| --- |
| os\_sem\_post( &work\_sem );  wakeup\_time += t\_cycle;  callout\_schedule\_at(c, wakeup\_time); |

Inside event\_wakeup() configuration, the interrupt level and sleep state of the Talaria TWO depends on its current state.

If suspend is enabled os\_suspend\_enable() then the IRQ level is set to low. Here, Talaria TWO is at sleeping state.

If suspend is disabled os\_suspend\_disable() then the IRQ level is set to high. Here, Talaria TWO is awake.

|  |
| --- |
| static int \_\_irq  event\_wakeup(uint32\_t irqno, void \*arg)  {  if (state) {  os\_suspend\_enable();  os\_gpio\_set\_irq\_level\_low(wakeup\_pin);  os\_gpio\_set\_value(led\_pin, 0);  os\_printf("sleeping\n");  }  else {  os\_suspend\_disable();  os\_gpio\_set\_irq\_level\_high(wakeup\_pin);  os\_gpio\_set\_value(0, led\_pin);  os\_printf("awake\n");  }  state ^= 1;  os\_clear\_event(EVENT\_GPIO\_3);  return IRQ\_HANDLED;  } |

This section describes the worker thread. When the state is set to 0, the suspend is enabled. If not, it is disabled. This is commonly used as a background task.

|  |
| --- |
| static void\* my\_thread(void\* arg)  {  for(;;) {  os\_sem\_wait( &work\_sem );  if (state == 0) {  os\_gpio\_set\_value(0, led\_pin);  os\_suspend\_disable();  os\_usleep(t\_awake);  os\_suspend\_enable();  os\_gpio\_set\_value(led\_pin, 0);  os\_printf(".");  }  else {  os\_printf("'");  }  }  return NULL;  } |

This section configures a GPIO as an interrupt:

|  |
| --- |
| wakeup\_pin = 1<< os\_get\_boot\_arg\_int("wakeup", 18);  …  os\_gpio\_request(wakeup\_pin);  os\_gpio\_set\_input(wakeup\_pin);  os\_gpio\_set\_irq\_level\_low(wakeup\_pin);  os\_gpio\_enable\_irq(wakeup\_pin, 3);  os\_attach\_event(EVENT\_GPIO\_3, event\_wakeup, NULL);  os\_gpio\_set\_pull(GPIO\_PIN(wakeup\_pin)); |

Here, wakeup\_pin (18) is configured as an interrupt which is triggered whenever the GPIO18 is set to LOW. It is also configured to call event\_wakeup(), whenever Talaria TWO wakes up from GPIO interrupt source.

### Running the Application

Program gpio\_suspend.elf(sdk\_x.y\examples\gpio\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 gpio\_suspend.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

Once the application is downloaded onto the Talaria TWO EVB-A, the LED D1 is ON.

In the demo application, Talaria TWO goes into Sleep Mode and wakes up and prints a ‘.’ every 300ms.

Next, use a jumper on GPIO18 and pull it LOW or connect it to GND. Connecting GPIO18 to GND will wake the Talaria TWO from sleep, and then go back to sleep.

This is achieved by attaching a GPIO interrupt on GPIO18. It is observed that LED D1 is turned OFF every time Talaria TWO wakes up.

|  |
| --- |
| UART:NWWWWAE  Build $Id: git-b664be2af $  hio.baudrate=115200  flash: Gordon ready!  UART:NWWWAEBuild $Id: git-b664be2af $  SUSPEND test  awake  '''''''''''''''''''''sleeping  ......................................awake  '''''''''''''''''''''''''''''''''sleeping  awake  ''''''''sleeping  ..awake  sleeping |

## GPIO Suspend Mask Demo Example

### Application Flow

The GPIO suspend mask demo application demonstrates configuring the GPIOs as input or output when the Talaria TWO module is in suspend mode.

Following are the steps:

1. Configure the GPIOs to desired state.
2. Output enable (oe) for GPIO14 and GPIO5 using bit masking.
3. GPIO5 is set to HIGH when in suspend.
4. In a while (1) loop, GPIO5 is set to HIGH when in suspend and previous GPIO level is restored for every configured interval (5 seconds).

### Sample Code Walkthrough

To configure the GPIOs to a desired state when Talaria TWO is in suspend mode, os\_gpio\_masks\_suspend\_set(struct os\_gpio\_masks mask) can be used according to the os\_gpio\_masks struct.

|  |
| --- |
| Struct os\_gpio\_masks {  uint32\_t ie;  uint32\_t oe;  uint32\_t pe;  uint32\_t o; }; |

When the Talaria TWO module comes out of suspend mode, the kernel would configure the GPIOs to their previous state before entering suspend.

|  |
| --- |
| suspend\_mask.ie = 0x00;  suspend\_mask.oe = 0b100000000100000; //OE on gpio14 and gpio5  suspend\_mask.pe = 0x00;  suspend\_mask.o = 0b000000000100000; //gpio5 set to HIGH when suspend |

### Running the Application

Program gpio\_suspend\_mask.elf (*sdk\_x.y\examples\gpio\bin*) using the Download Tool:

1. Launch the Download Tool provided with InnoPhase Talaria TWO SDK.
2. In the GUI window:
3. Boot Target: Select the appropriate EVK from the drop-down.
4. ELF Input: Load the gpio\_suspend\_mask.elf by clicking on Select ELF File.
5. 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

As shown in Figure 2, connect the Talaria TWO board with the appropriate GPIOs to the Logic Analyzer with the required wires, and check the Logic analyzer’s output.

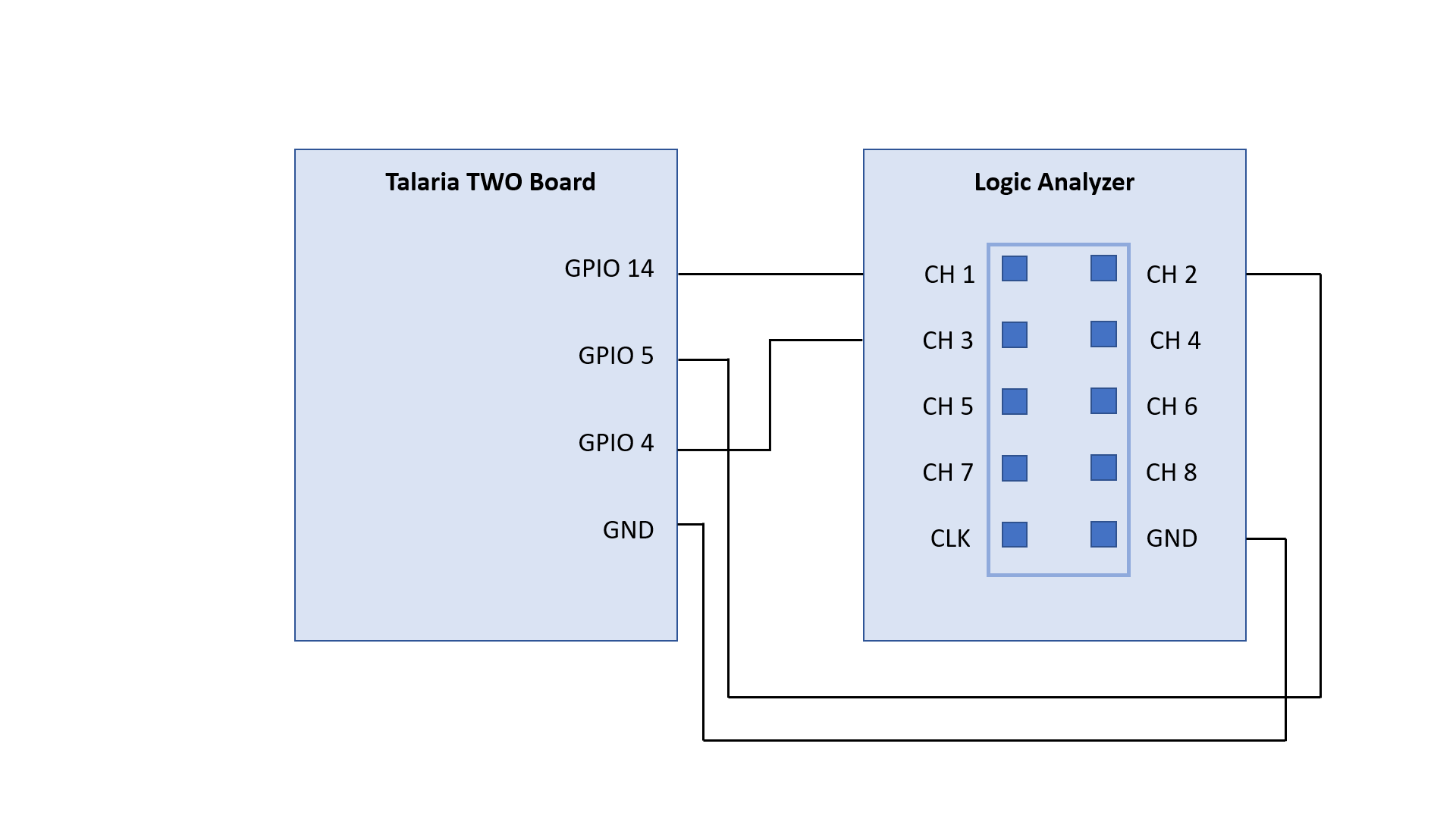


Figure : GPIO suspend mask demo- Block diagram

As shown in Figure 3, GPIO 5 is set to HIGH when in suspend and previous GPIO level is restored for every configure interval (5 seconds).

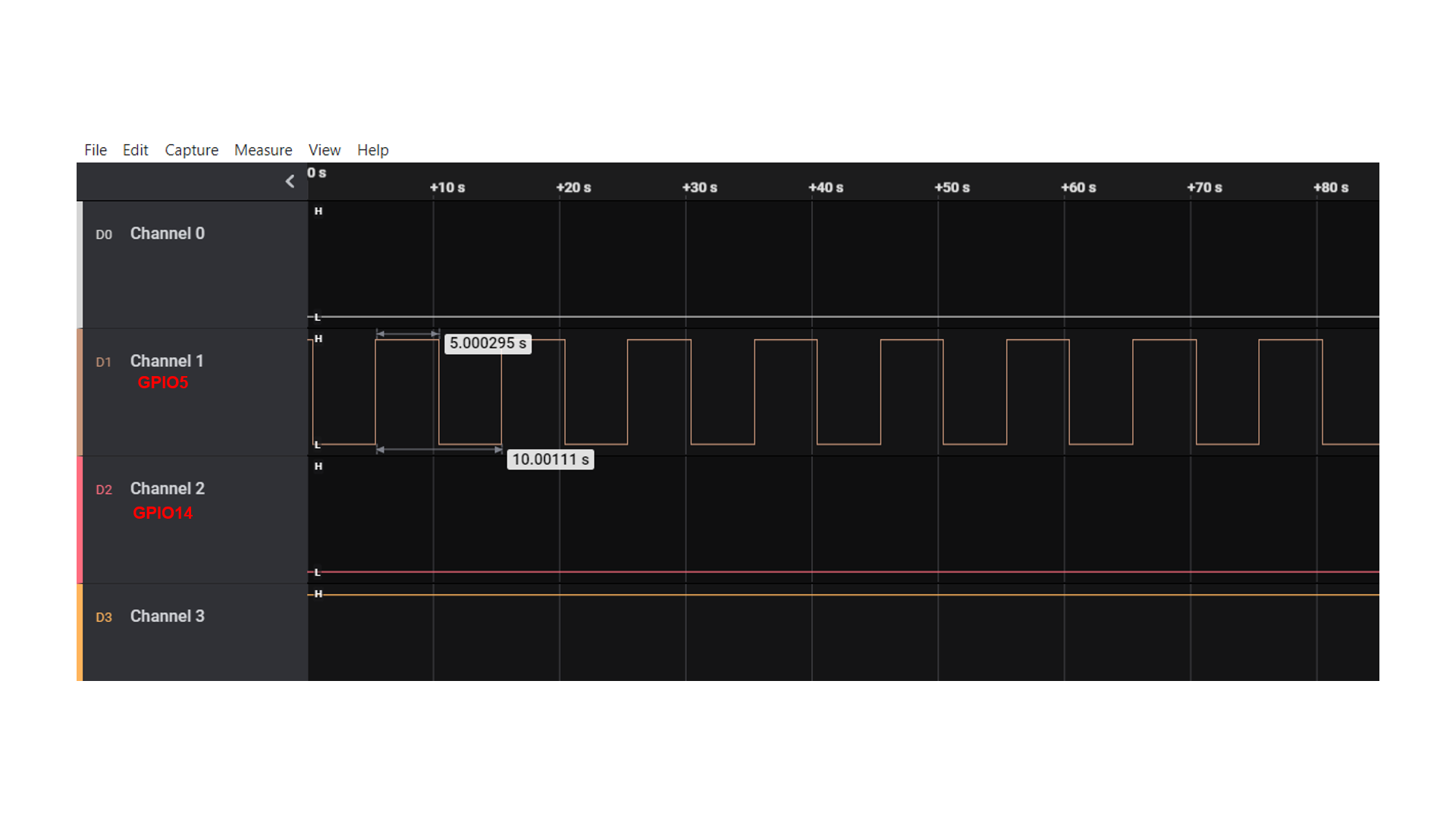


Figure : Logic Analyzer output

**Console output**:

|  |
| --- |
| Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PNWWWAEBuild $Id: git-34e3eddb8 $  GPIO Mask test  T2 sleeping  T2 awake  T2 sleeping  T2 awake  T2 sleeping  T2 awake  T2 sleeping  T2 awake  T2 sleeping  T2 awake  T2 sleeping  T2 awake  T2 sleeping  T2 awake  T2 sleeping  T2 awake  T2 sleeping  T2 awake  T2 sleeping  T2 awake  T2 sleeping  T2 awake  T2 sleeping  T2 awake  T2 sleeping  T2 awake  T2 sleeping  T2 awake  T2 sleeping  T2 awake  T2 sleeping  T2 awake  T2 sleeping  T2 awake  T2 sleeping  T2 awake |

## GPIO Interrupt Awake Demo Example

### Application Flow

The GPIO interrupt awake demo application demonstrates the configuration of two GPIO input pins to signal interrupt on low level.

Following are the steps:

1. Initialize the wakeup parameters.
2. Select a GPIO for waking up the Talaria TWO module and attach an event handler function to a GPIO event.
3. Identify the wakeup source array index of the wakeup signal currently being debounced.
4. Debouncing logic to detect the switch debouncing.
5. Callback function to show case enabling the interrupt.
6. Detach an attached event handler function from a GPIO event.

### Sample Code Walkthrough

This section describes the main function of the gpio\_interrupt\_awake application. It initializes the wakeup parameters and initiates the GPIO configured as an interrupt pin through boot arguments.

|  |
| --- |
| wakeup\_init();  wakeup\_gpio = os\_get\_boot\_arg\_int("key\_pin", 0xFF);  if(wakeup\_gpio != 0xFF)  {  if(wakeup\_gpio\_select(0, wakeup\_gpio, WAKEUP\_TRIG\_LVL\_LOW, DEBOUNCE\_TIME\_MS, callback\_1) == -1)  return -1;  }  wakeup\_gpio = os\_get\_boot\_arg\_int("pir\_pin", 0xFF);  if(wakeup\_gpio != 0xFF)  {  if(wakeup\_gpio\_select(1, wakeup\_gpio, WAKEUP\_TRIG\_LVL\_LOW, DEBOUNCE\_TIME\_MS, callback\_2) == -1)  return -1;  }  /\*Enter suspend state\*/  os\_suspend\_enable();  while(1){  os\_msleep(10000);  } |

wakeup\_init()initiates the wakeup parameters. wakeup\_gpio\_select() selects a GPIO to wake up the Talaria TWO module and then the Talaria TWO module will enter into suspend mode (sleep mode).

wakeup\_gpio\_select() selects a GPIO for waking up the Talaria TWO module through an interrupt event (GPIO to which the wakeup source (Push button switch) is connected).

os\_gpio\_set\_irq\_level\_low() is a logic level of the GPIO pin when a wakeup signal is invoked. This level is used for identifying the IRQ level.

This block of code will attach an event handler function to a GPIO event. The GPIO pin configured through the boot argument is an input pin and the os\_gpio\_attach\_event() attaches an event handler function to a GPIO event.

**Note:** In this example we have demonstrated event 3 (gpio\_event\_3).

|  |
| --- |
| if(!os\_gpio\_request(GPIO\_PIN(gpio\_pin)) || wakeup\_index >= MAX\_WAKEUP\_SOURCE\_CNT) {  os\_printf("\n wakeup\_gpio\_select for gpio %d failed..!\n", gpio\_pin);  return -1;  }  wakeup\_src[wakeup\_index].gpio = gpio\_pin;  os\_gpio\_set\_input(GPIO\_PIN(wakeup\_src[wakeup\_index].gpio));  if(trigger\_level == 0){  wakeup\_src[wakeup\_index].irq\_level = 0;  os\_gpio\_set\_irq\_level\_low(GPIO\_PIN(wakeup\_src[wakeup\_index].gpio));  }  wakeup\_src[wakeup\_index].wakeup\_clbk = fptr;  wakeup\_src[wakeup\_index].debounce\_ms = 1000UL \* debounce\_ms;  os\_gpio\_attach\_event(gpio\_event\_3, gpio\_changed, NULL);  callout\_init(&wakeup\_src[wakeup\_index].delay\_callout, debounce\_fn);  os\_gpio\_enable\_irq(GPIO\_PIN(wakeup\_src[wakeup\_index].gpio), 3);//Testing with gpio event 3 for key\_pin  os\_printf("\n gpio-%d is selected for the interrupt \n", wakeup\_src[wakeup\_index].gpio); |

After attaching the event, the callout function will be called to check the interrupt occurrence in the configured GPIO pin.

If an interrupt occurs in any of the configured GPIO pins, the gpio\_changed() handler function will be executed. The GPIO ISR handler finds which wake-up source triggered the IRQ.

|  |
| --- |
| wakeup\_index = wakeup\_index\_from\_irq();  if(wakeup\_index != 0xFF)  {  callout\_schedule(&wakeup\_src[wakeup\_index].delay\_callout, wakeup\_src[wakeup\_index].debounce\_ms);  // disable all wakeup source IRQs, debounce one at a time  disable\_wakeup\_irqs();  wakeup\_src[wakeup\_index].debouncing = true;  } |

The wakeup\_index\_from\_irq()identifies which wakeup source is causing the interrupt.

|  |
| --- |
| for(wakeup\_index = 0; wakeup\_index < MAX\_WAKEUP\_SOURCE\_CNT; wakeup\_index++)  {  if(wakeup\_src[wakeup\_index].gpio != 0xFF)  {  if(wakeup\_src[wakeup\_index].invoked == true)  {  irq\_level = !wakeup\_src[wakeup\_index].irq\_level;  }  else  {  irq\_level = wakeup\_src[wakeup\_index].irq\_level;  }  gpio\_val = os\_gpio\_get\_value(GPIO\_PIN(wakeup\_src[wakeup\_index].gpio));  gpio\_val = gpio\_val >> wakeup\_src[wakeup\_index].gpio;  if(irq\_level == (uint8\_t)gpio\_val)  {  return wakeup\_index;  }  } |

The callout\_schedule() executes the callout delay function and debounce logic to avoid registering multiple interrupts in a short period of time. The debouncing logic will execute the 40ms debouncing time.

The disable\_wakeup\_irqs() will disable all wakeup source IRQs, debounce one at a time and it will find which wakeup source is being debounced.

|  |
| --- |
| uint8\_t wakeup\_index;  for(wakeup\_index = 0; wakeup\_index < MAX\_WAKEUP\_SOURCE\_CNT; wakeup\_index++)  {  if(wakeup\_src[wakeup\_index].gpio != 0xFF)  {  os\_gpio\_disable\_irq(GPIO\_PIN(wakeup\_src[wakeup\_index].gpio));  }  } |

The debouncing\_wakeup\_src()identifies the wake-up source array index of the wake-up signal currently being debounced.

|  |
| --- |
| uint8\_t wakeup\_index;  for(wakeup\_index = 0; wakeup\_index < MAX\_WAKEUP\_SOURCE\_CNT; wakeup\_index++)  {  if(wakeup\_src[wakeup\_index].debouncing == true)  {  return wakeup\_index;  }  } |

The debounce\_fn()is called from the configured callout. This callout function is for debouncing the wakeup signal. The debounce logic will validate that the wake-up signal is in a stable state after the delay.

|  |
| --- |
| uint8\_t wakeup\_index = debouncing\_wakeup\_src();  if(wakeup\_src[wakeup\_index].invoked == false){  if(wakeup\_src[wakeup\_index].irq\_level == 0){  if(os\_gpio\_get\_value(GPIO\_PIN(wakeup\_src[wakeup\_index].gpio)) == false){  wakeup\_src[wakeup\_index].invoked = true;  os\_gpio\_set\_irq\_level\_high(GPIO\_PIN(wakeup\_src[wakeup\_index].gpio));  // send message to custom task  wakeup\_src[wakeup\_index].invoke\_count++;  wakeup\_src[wakeup\_index].wakeup\_clbk();  os\_suspend\_disable();  }  }  }  else{  if(wakeup\_src[wakeup\_index].irq\_level == 0){  if(os\_gpio\_get\_value(GPIO\_PIN(wakeup\_src[wakeup\_index].gpio)) != false){  wakeup\_src[wakeup\_index].invoked = false;  os\_gpio\_set\_irq\_level\_low(GPIO\_PIN(wakeup\_src[wakeup\_index].gpio));  os\_suspend\_enable();  }  }  }  if ((wakeup\_src[wakeup\_index].invoke\_count) == 10){  os\_gpio\_detach\_event(gpio\_event\_3, gpio\_changed);  os\_printf(" \n Interrupt\_count exceeded to -%d \n", wakeup\_src[wakeup\_index].invoke\_count);  }  enable\_wakeup\_irqs();  wakeup\_src[wakeup\_index].debouncing = false;  } |

**Note:** The debounce logic will count the number of interrupt occurrences. Once the number of interrupts equals 10, the os\_gpio\_detach\_event()will detach an attached event handler function from a GPIO event. Also, Talaria TWO module will enter awake state.

enable\_wakeup\_irqs()enables the interrupt from all wake-up sources once the switch is debounced.

|  |
| --- |
| uint8\_t wakeup\_index;  for(wakeup\_index = 0; wakeup\_index < MAX\_WAKEUP\_SOURCE\_CNT; wakeup\_index++)  {  if(wakeup\_src[wakeup\_index].gpio != 0xFF)  {  os\_gpio\_enable\_irq(GPIO\_PIN(wakeup\_src[wakeup\_index].gpio), 3);//Testing with gpio event 3 for key\_pin  }  } |

### Running the Application

Program gpio\_interrupt\_awake.elf(sdk\_x.y\examples\gpio\bin) using the Download tool:

1. Launch the Download tool provided with InnoPhase Talaria TWO SDK.
2. In the GUI window:
3. Boot Target: Select the appropriate EVK from the drop-down.
4. ELF Input: Load the gpio\_interrupt\_awake.elf by clicking on Select ELF File.
5. Boot Arguments: Pass the following boot arguments:

|  |
| --- |
| key\_pin=20, pir\_pin=21 |

1. 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

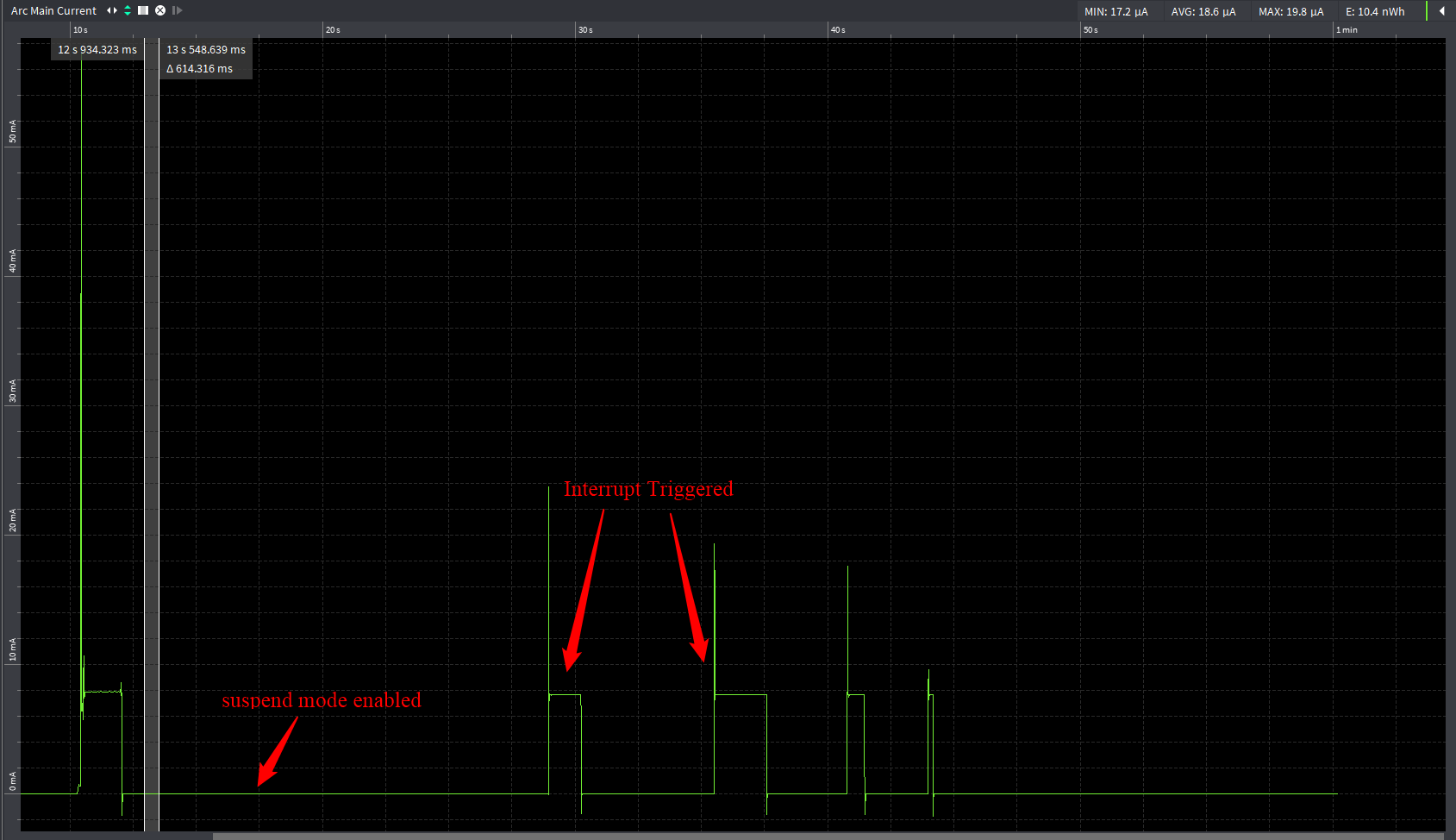
In the demo application, Talaria TWO goes into Sleep Mode.

Use a push button switch on which all the GPIO’s are configured as an Interrupt. Connecting push button switch to GND will wake the Talaria TWO from sleep, and then goes back to sleep.

After ten (10) interrupts in any one of the configured interrupt pins, the application will detach an attached event handler function from a GPIO event.

**Note**: In this example, GPIO 20 and 21 are used as an external interrupt pins.

|  |
| --- |
| Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PNWWWWAE4 DWT comparators, range 0x8000  Build $Id: git-6ed2e370c $  key\_pin=20 pir\_pin=21  Interrupt test  gpio-20 is selected for the interrupt  gpio-21 is selected for the interrupt  Pir\_pin interrupt enabled  Key\_Pin interrupt enabled  Pir\_pin interrupt enabled  Key\_Pin interrupt enabled |



*Figure 2: Otii capture for gpio\_interrupt\_awake application.*