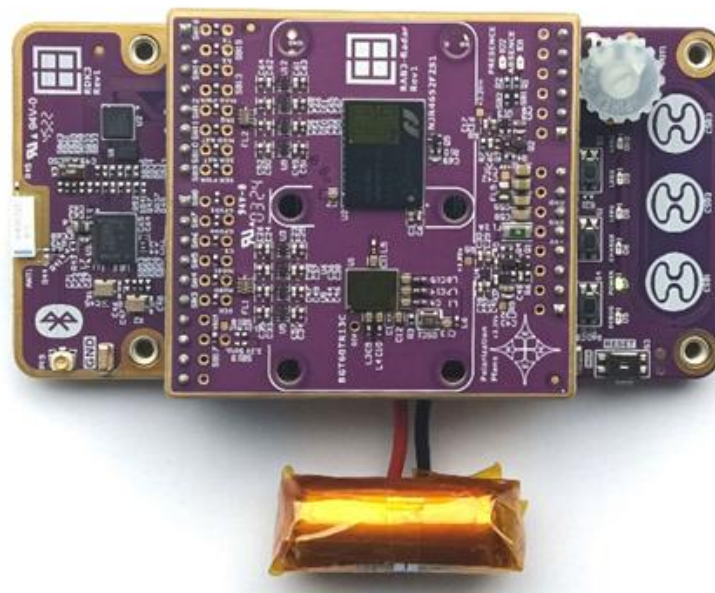


Using Infineon BGT60TR13C for presence detection

Application note



Versions

Version	Date	Rationale
0.1	April 29, 2024	First release. Authors: ROJ
1.0	May 7, 2024	New chapters, formatting. Authors: ROJ, KOA

Legal Disclaimer

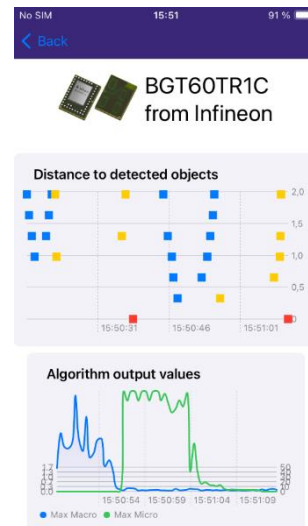
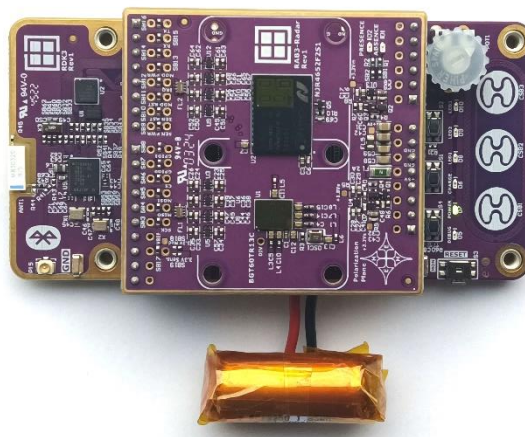
The evaluation board is for testing purposes only and, because it has limited functions and limited resilience, is not suitable for permanent use under real conditions. If the evaluation board is nevertheless used under real conditions, this is done at one's responsibility; any liability of Rutronik is insofar excluded.

Table of Contents

Versions	2
Legal Disclaimer	2
Table of Contents	2
Overview	3
Introduction	3
Useful terms	3
Needed hardware	3
Hardware	4
Software	5
Overview	5
Firmware Structure	5
Radar sensor API	6
Radar presence detection library	7
Configuration of presence detection algorithm	9
Overview	9
Limitation due to radar configuration	11
Test up to 10 meters	11
Configuration of the radar sensor	13
Configuration flow	13
Change configuration and generate header file	13
Radar parameters	16
Samples per chirp	16
Chirps per frame	16
Chirp repetition time	16
Frame rate	16
Start and end frequency	17

Overview

Introduction



The solution enables to detect presence using the Infineon BGT60TR13C. The RDK3 board collects the measurement data from the BGT60TR13C and process them. The results of the algorithm are sent over BLE to the iOS and Android app.

Concerning the colours displayed in the app:

- **Blue** points: macro motions
- **Yellow** points: micro motions
- **Red** points: absence detected

Useful terms

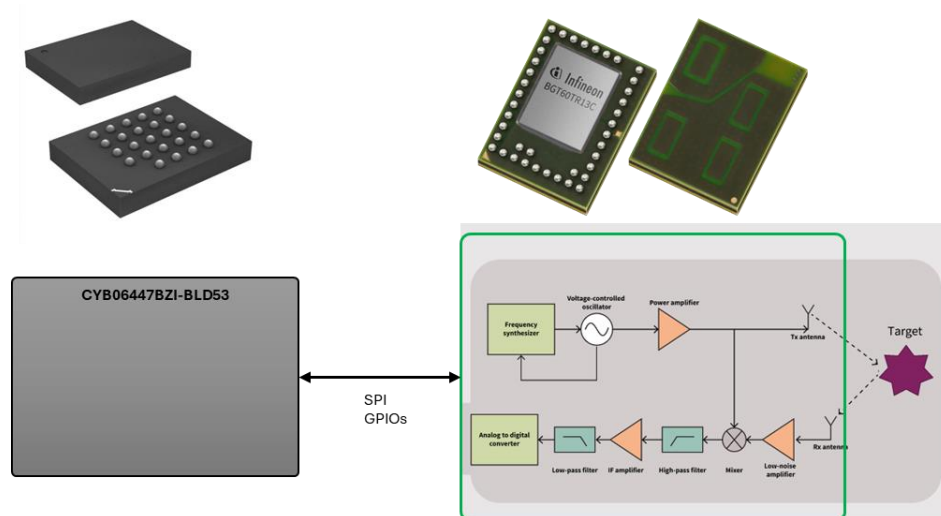
Micro-motions: detecting small movements like gestures or small head movements in a typical smart home environment for instance while working on laptop/keyboard. Micro-motion also includes detection of stationary humans (normally breathing and blinking eyes) in sitting or standing positions (in line of sight).

Macro-motions: detecting major movements into or through the field of view (Motion Detection).

Needed hardware

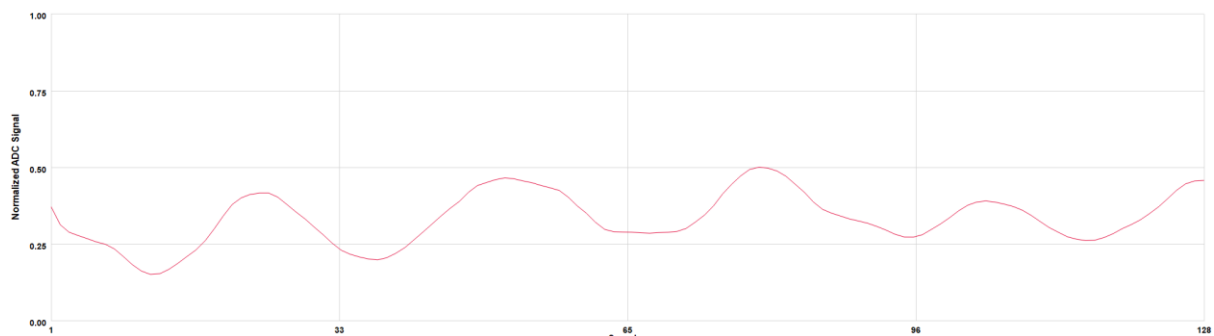
- RDK3 board.
- RAB3-Radar board.
- *Optional:* battery.

Hardware

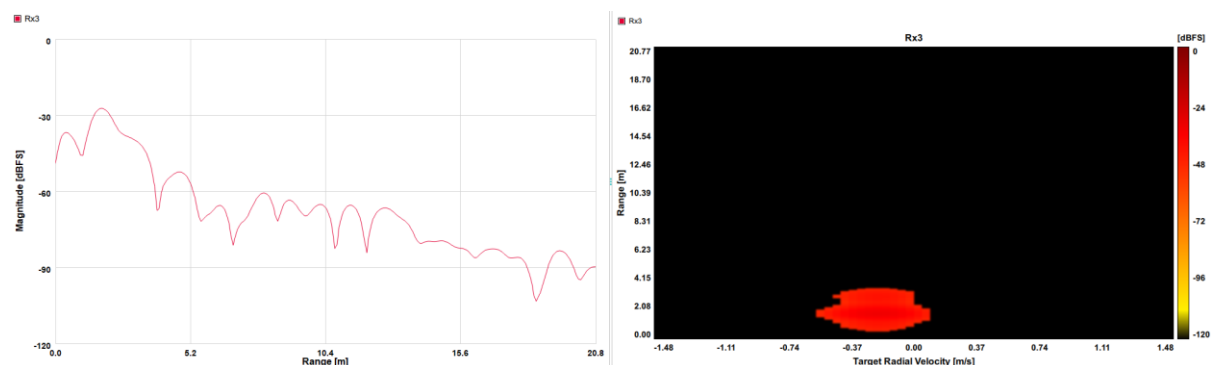


The BGT60TR13C embeds antennas and signal conditioning block. It delivers raw measurements to the microcontroller over SPI. The maximum SPI communication frequency is 50MHz. In our example with the RDK3, we use a frequency of 18.75MHz.

The values sent by the BGT60TR13C to the microcontroller are ADC samples that look like:



Using digital signal processing, it is possible to extract information from those raw values, like a fact of presence for example. This is the role of the microcontroller.

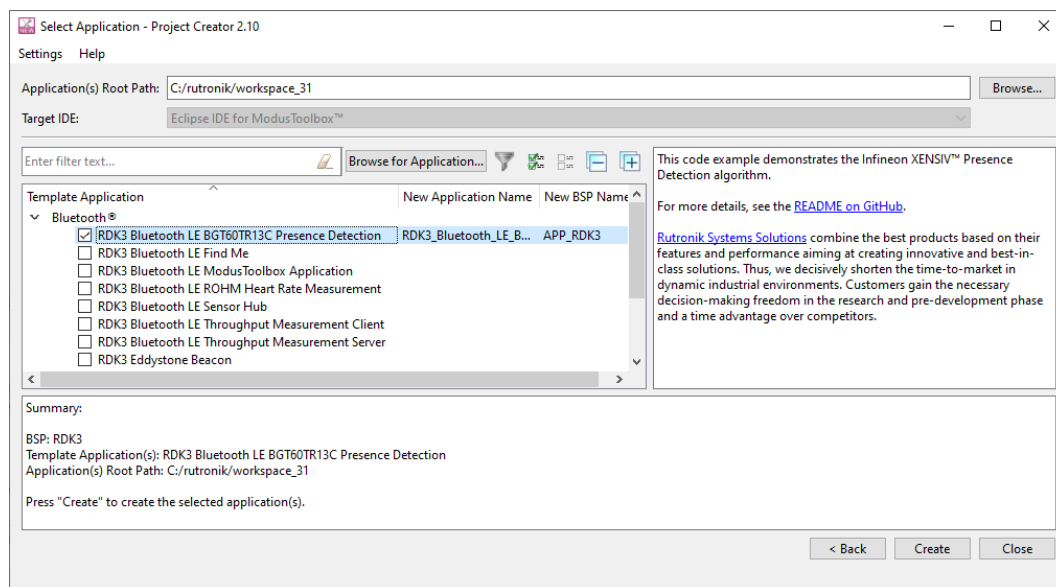


Screenshots of the Radar Fusion GUI

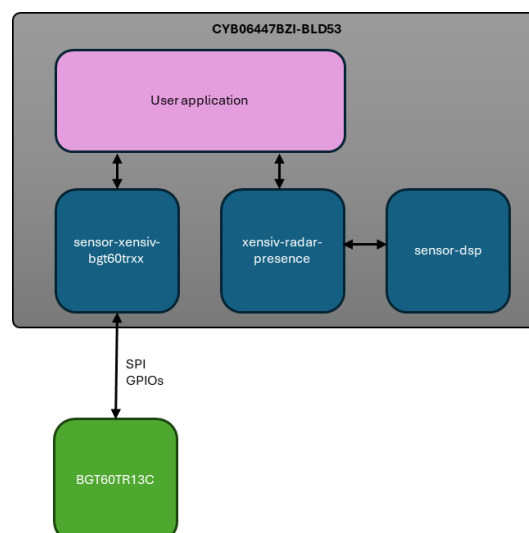
Software

Overview

The firmware is available on [GitHub](#) and is directly available inside Modus Toolbox project creator (File -> New -> ModusToolbox Application).



Firmware Structure



sensor-xensiv-bgt60trxx library enables to configure the radar sensor and to get access to the raw data.

xensiv-radar-presence library enables to detect a presence.

sensor-dsp library is needed for xensiv-radar-presence.

Radar sensor API

First, you need to initialize the BGT60TR13C.

<code>static xensiv_bgt60trxx_mtb_t sensor;</code>	
<code>xensiv_bgt60trxx_mtb_init(</code>	
<code>&sensor,</code>	Pointer to the structure that will be initialized
<code>&spi_obj,</code>	Pointer to SPI object. Need to be initialized first using the cyhal_spi_init function.
<code>ARDU_CS,</code>	Chip select GPIO to be used (during SPI communication)
<code>ARDU_I04,</code>	GPIO to be used to reset the module.
<code>register_list,</code> <code>XENSIV_BGT60TRXX_CONF_NUM_REGS);</code>	Configuration of the radar sensor. It can be generated using the Radar Fusion GUI tool. The configuration is stored inside the file <code>radar_settings.h</code>

Then, configure how the radar module generates interrupts when the data is available.

<code>xensiv_bgt60trxx_mtb_interrupt_init(</code>	
<code>&sensor,</code>	Pointer to the structure that was previously initialized.
<code>NUM_SAMPLES_PER_FRAME,</code>	Number of samples to be acquired before generating an interrupt. We want an interrupt to be generated after each frame.
<code>ARDU_I06,</code>	GPIO to be used as interrupt line. When a new data packet is available the module will signal it using this GPIO.
<code>CYHAL_ISR_PRIORITY_DEFAULT,</code> <code>xensiv_bgt60trxx_mtb_interrupt_handler,</code> <code>NULL);</code>	Priority and function to be called when an interrupt is detected.

In our case, the interrupt handler function only signals to the main task that a value is available:

<code>void xensiv_bgt60trxx_mtb_interrupt_handler(void *args, cyhal_gpio_event_t event)</code>
<code>{</code>
<code> CY_UNUSED_PARAMETER(args);</code>
<code> CY_UNUSED_PARAMETER(event);</code>
<code> app.radar_values_available = true;</code>
<code>}</code>

Then, you can start the radar measurements:

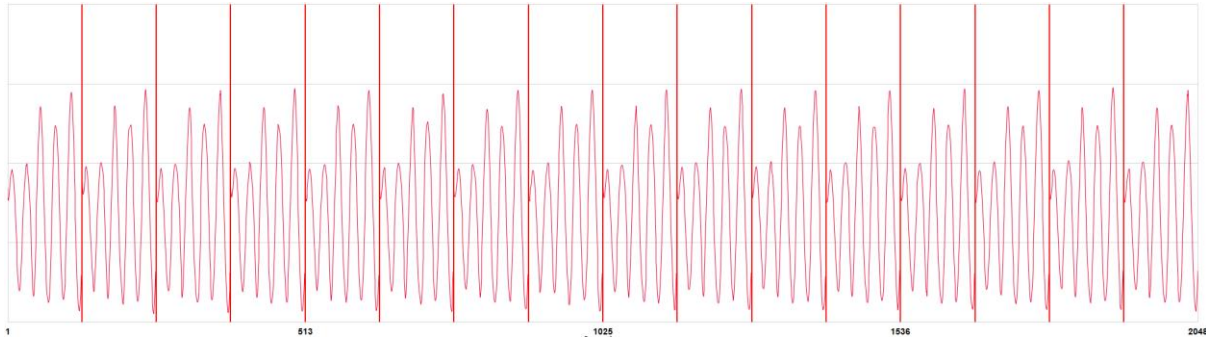
<code>xensiv_bgt60trxx_start_frame(&sensor.dev, true)</code>
--

After starting the measurements, interrupt will be generated and caught by the interrupt handler (the signals that data are available inside the radar's FIFO).

To read the data from the radar sensor, call this function:

```
static uint16_t samples[NUM_SAMPLES_PER_FRAME];
xensiv_bgt60trxx_get_fifo_data(&sensor.dev, samples, NUM_SAMPLES_PER_FRAME);
```

The “samples” buffer will then contain the raw values of the radar sensor. The range is between 0 and 4096. In our case, the values look like:



There are 2048 samples, because we have 16 chirps with 128 samples per chirp.

Radar presence detection library

You first need to initialize the library:

<code>xensiv_radar_presence_set_malloc_free(malloc, free);</code>	Tells to the library which malloc and free functions it should use.
<code>xensiv_radar_presence_alloc(&presence_handle, &default_config)</code>	Initializes the detection algorithm (you can have a look to the configuration below).
<code>xensiv_radar_presence_set_callback(presence_handle, presence_detection_cb, NULL);</code>	Tells to the algorithm the function to be called when a presence is detected. In our case, it is the function presence_detection_cb . You can see how it should be implemented below.

```
static const xensiv_radar_presence_config_t default_config =
{
    .bandwidth
XENSIV_BGT60TRXX_CONF_START_FREQ_HZ,
    .num_samples_per_chirp
    .micro_fft_decimation_enabled
    .micro_fft_size
    .macro_threshold
    .micro_threshold
    .min_range_bin
    .max_range_bin
    .macro_compare_interval_ms
    .macro_movement_validity_ms
    .micro_movement_validity_ms
    .macro_movement_confirmations
    = XENSIV_BGT60TRXX_CONF_END_FREQ_HZ -
    = XENSIV_BGT60TRXX_CONF_NUM_SAMPLES_PER_CHIRP,
    = false,
    = 128,
    = 0.5f,
    = 12.5f,
    = 1,
    = 6,
    = 250,
    = 1000,
    = 4000,
    = 0,
```

```
.macro_trigger_range      = 1,  
.mode                    = XENSIV_RADAR_PRESENCE_MODE_MICRO_IF_MACRO,  
.macro_fft_bandpass_filter_enabled = false,  
.micro_movement_compare_idx = 5  
};
```

```
void presence_detection_cb(xensiv_radar_presence_handle_t handle,  
                          const xensiv_radar_presence_event_t* event,  
                          void *data)  
{  
    (void)handle;  
    (void)data;  
    float distance = 0;  
    distance = xensiv_radar_presence_get_bin_length(presence_handle) * (float) event->range_bin;  
    switch(event->state)  
    {  
        case XENSIV_RADAR_PRESENCE_STATE_MACRO_PRESENCE:  
            break;  
        case XENSIV_RADAR_PRESENCE_STATE_MICRO_PRESENCE:  
            break;  
        case XENSIV_RADAR_PRESENCE_STATE_ABSENCE:  
            break;  
    }  
}
```

Once the library is initialized, you can start to process the radar values that you read using the **xensiv_bgt60trxx_get_fifo_data** function.

Remark: the raw radar values first need to be converted into a range 0 to 1.

To process the data, call the function:

```
xensiv_radar_presence_process_frame(presence_handle, frame, app.timestamp)
```

If a presence (or absence) is detected, the callback function (**presence_detection_cb** in our case) will be called.

Configuration of presence detection algorithm

Overview

You can find the documentation of Infineon by following this link:
https://infineon.github.io/xensiv-radar-presence/html/group_group_board_libs.html

```
static const xensiv_radar_presence_config_t default_config =
{
    .bandwidth
XENSIV_BGT60TRXX_CONF_START_FREQ_HZ,
    .num_samples_per_chirp
    .micro_fft_decimation_enabled
    .micro_fft_size
    .macro_threshold
    .micro_threshold
    .min_range_bin
    .max_range_bin
    .macro_compare_interval_ms
    .macro_movement_validity_ms
    .micro_movement_validity_ms
    .macro_movement_confirmations
    .macro_trigger_range
    .mode
    .macro_fft_bandpass_filter_enabled
    .micro_movement_compare_idx
};
```

You can define the range in which the presence detection algorithm detects a presence using the parameters “**min_range_bin**” and “**max_range_bin**”.

The units of those 2 parameters is not in meters but in “bin”. To convert from meters to “bin” you need to consider the bandwidth you are using (in Hz) and the speed of light ($c = 299792458\text{m/s}$).

The conversion is then as follow:

$$\text{bin (m)} = c / 2 * \text{bandwidth} * (\text{FFT size} / \text{samples per chirp})$$

In our case:

$$\text{FFT size} = \text{samples per chirp} = 128 \rightarrow \text{FFT size} / \text{samples per chirp} = 1$$

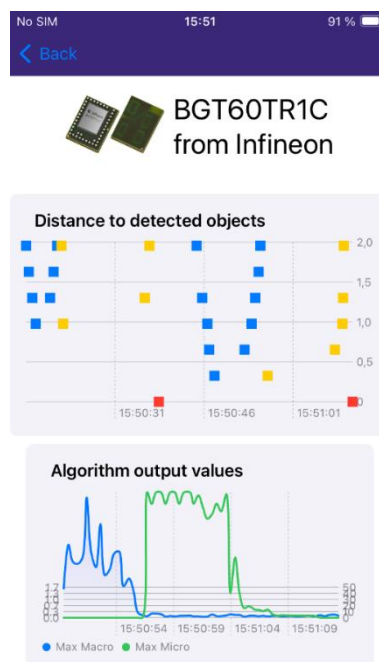
$$\begin{aligned} \text{Bandwidth} &= \text{XENSIV_BGT60TRXX_CONF_END_FREQ_HZ} - \text{XENSIV_BGT60TRXX_CONF_START_FREQ_HZ} \\ \text{Bandwidth} &= 6148000000 - 6102000000 = 460\,000\,000\text{ Hz} \end{aligned}$$

$$\text{bin (m)} = 299792458 / 2 * 460\,000\,000 = 0.325861\text{ meters}$$

The presence detection algorithm will detect a presence within this range:

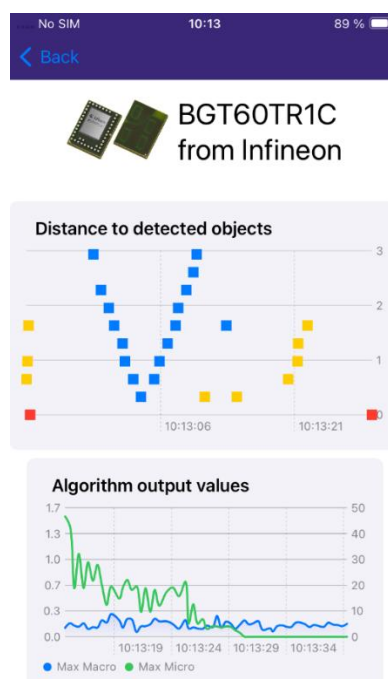
$$\begin{aligned} \text{min_range_bin} = 1 &\rightarrow 0.325861\text{ meters} \\ \text{max_range_bin} = 6 &\rightarrow 1.95516\text{ meters} \end{aligned}$$

Therefore, when using the app with the standard configuration, you will only see presence detected up to ~ 2 meters:



We can change the configuration of the presence detection algorithm to ~3 meters by changing the parameters:

$\text{min_range_bin} = 1 \rightarrow 0.325861 \text{ meters}$
$\text{max_range_bin} = 9 \rightarrow 2.932749 \text{ meters}$



The algorithm now detects presence up to 3 meters.

Limitation due to radar configuration

In the previous chapter, we enhanced the detection range of the presence detection algorithm by changing some software parameters.

The maximum theoretical range that the radar sensor can achieve is given by the formula:

$$R_{\max} = (\text{Num Samples per chirp} / 2) * (c / 2 * \text{bandwidth})$$

In our case, we have:

Num Samples per chirp = 128 and Bandwidth = 460 MHz

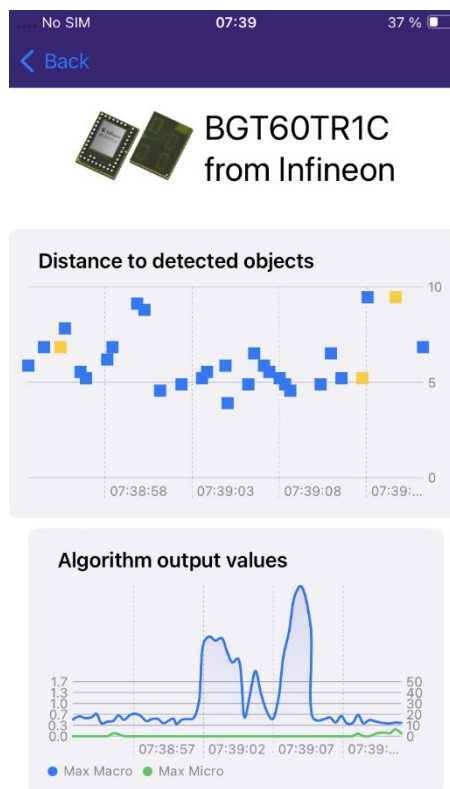
$$R_{\max} = (128/2) * (299792458 / (2 * 460\,000\,000)) = 20.86 \text{ meters}$$

This means that the radar can detect a target up to 20.86 meters if the signal is above noise floor (which depends on Tx power, environment noise and radar cross section of the target).

Test up to 10 meters

min_range_bin = 1 → 0.325861 meters

max_range_bin = 30 → 9.77583 meters

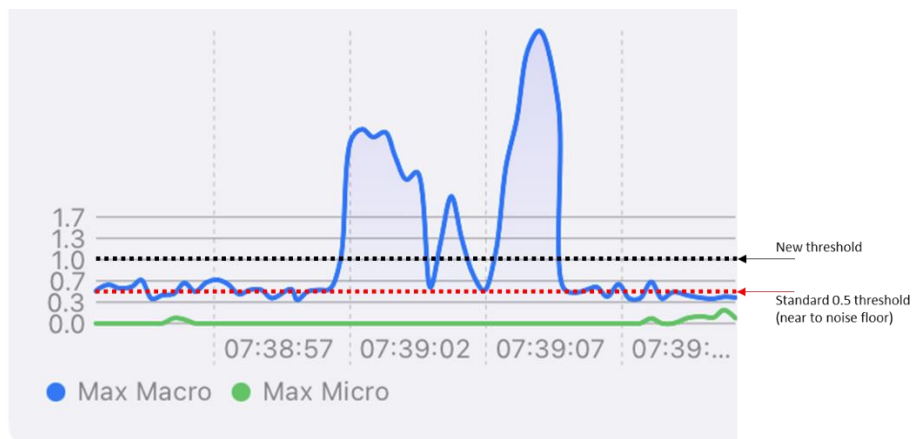


Just changing the “**max_range_bin**” parameter is not enough, because of environmental noise (might depends on the environment).

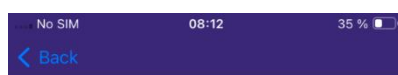
At the moment, the radar sensor detects presence, even if nobody is in front of the sensor.

To make the sensor more reliable, we can change the threshold of the presence detection algorithm. By default, it is set to 0.5f, but we will set it to 1.f.

```
min_range_bin = 1 → 0.325861 meters
max_range_bin = 30 → 9.77583 meters
macro_threshold = 1.f
```



With the new configuration, the results look like:



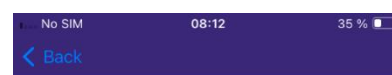
 BGT60TR1C
from Infineon

Distance to detected objects

Algorithm output values



Without motion, the “blue” signal remains below 1 (which is our threshold)

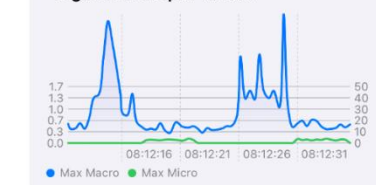


 BGT60TR1C
from Infineon

Distance to detected objects



Algorithm output values



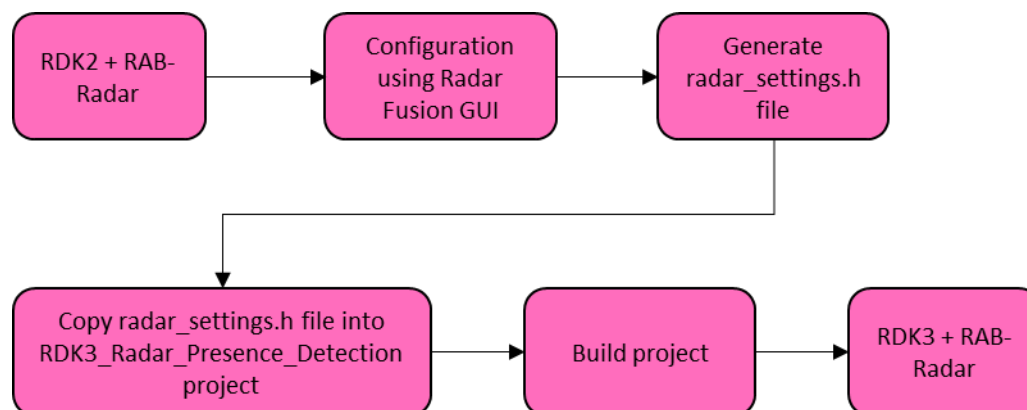
A motion is currently detected up to 8 meters (noisy indoor environment)

Configuration of the radar sensor

The configuration provided inside the “RDK3 Radar Presence Detection” example works with 1 antenna, and has a theoretical max detection range of 20.9 meters.

If you want to change it, you will need generate a new configuration using the “Radar Fusion GUI” tool of Infineon (it enables to generate a new “radar_settings.h” file).

Configuration flow



It is a good idea to store the configuration as JSON file in order to load it again later on.

Change configuration and generate header file

The Radar Fusion GUI is free:

<https://softwaretools.infineon.com/tools/com.ifx.tb.tool.radarfusiongui>

Hardware requirements:

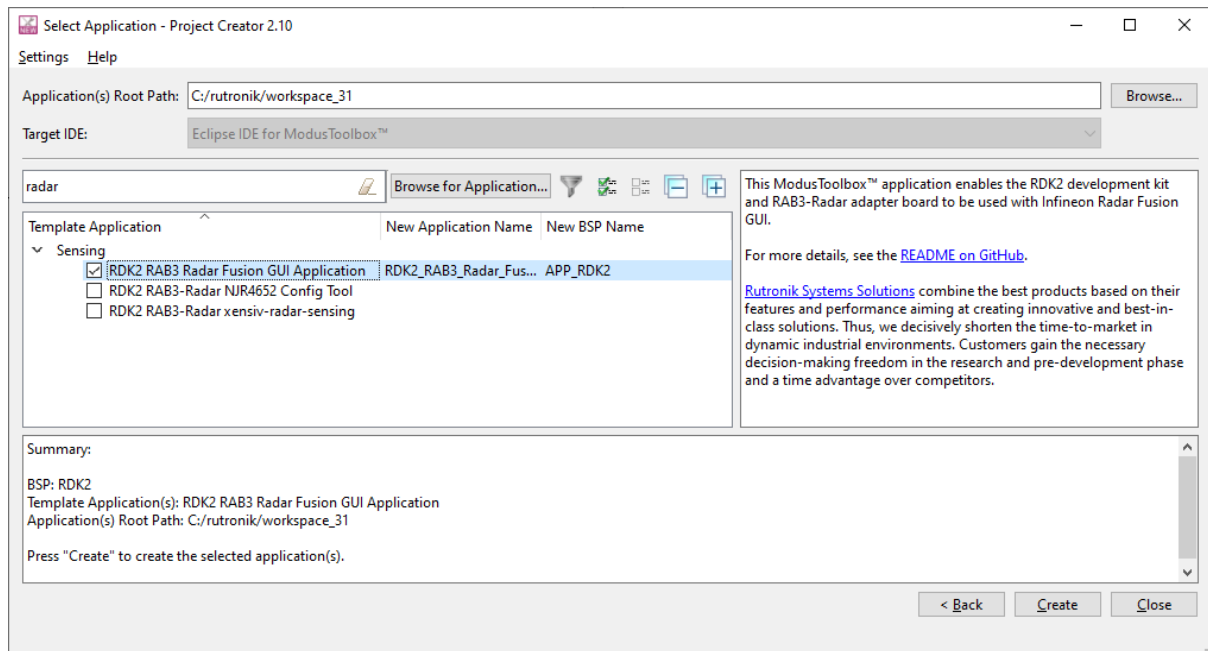
- RDK2
- RAB3-RADAR.

1. Program this ModusToolbox project on your RDK2:

https://github.com/RutronikSystemSolutions/RDK2_RAB3_Radar_Fusion

Go File -> New -> Modus Toolbox Application -> PSoC 6 BSPs -> RDK2 ->

Sensing -> RDK2 RAB3 Radar Fusion GUI Application

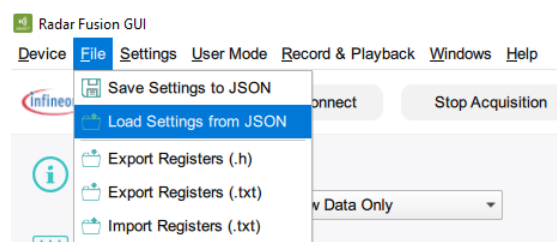


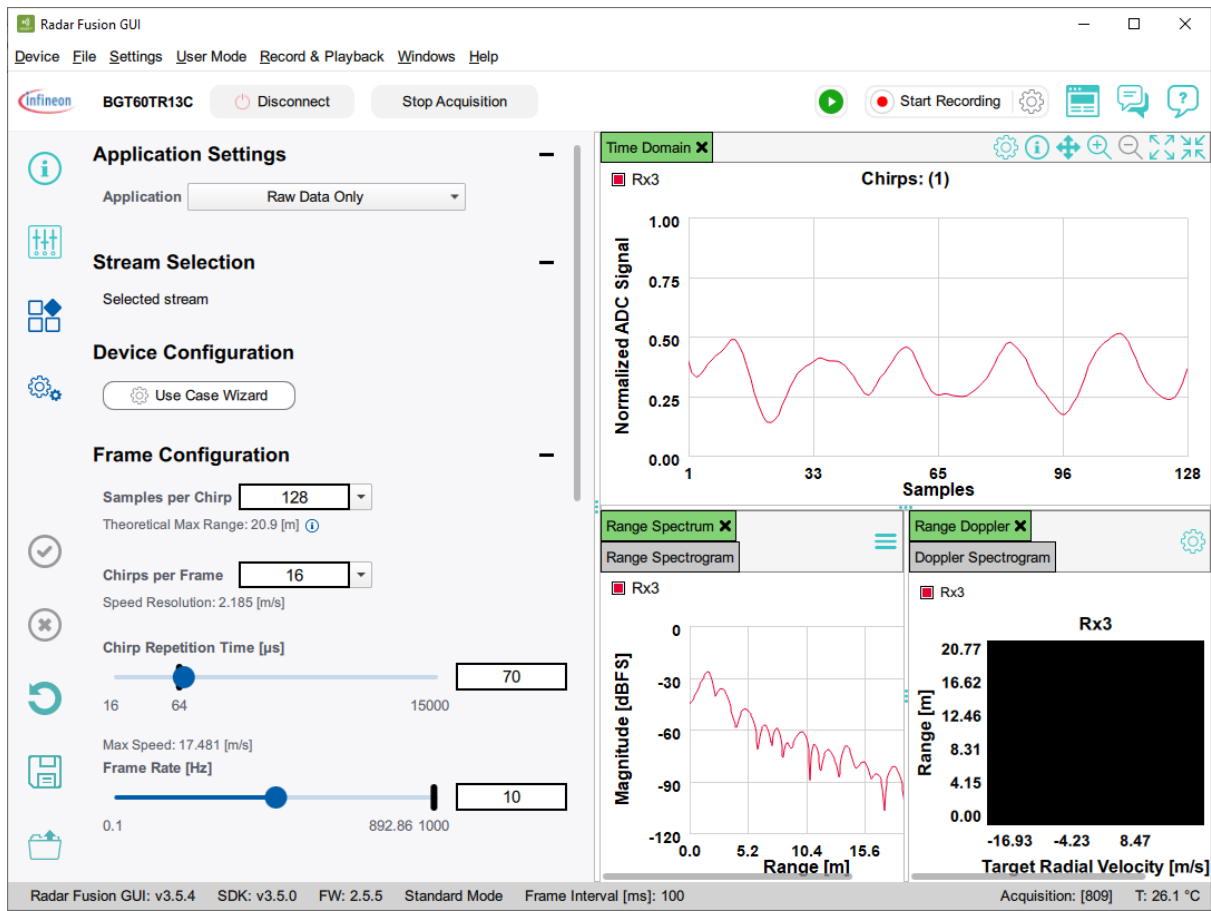
2. After programming the RDK2 with the correct firmware (using Kit Prog3), plug an USB-C cable and open the Radar Fusion GUI.



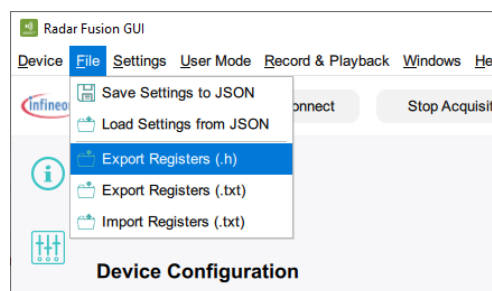
3. Use the menu to load the actual configuration (the file radar_settings.json is available inside the RDK3 Radar Presence Detection project).

If you are using the last version of the GUI (3.5.4 at this time) open the radar_settings_v4.json file.





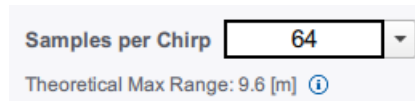
4. Change the [parameters](#) of the radar sensor.
5. Once you are done with the configuration, you can export it as a header file:



6. Replace the content of the file "radar_settings.h" of the RDK3_Radar_Presence_Detection project, build the project and program it to the RDK3.

Radar parameters

Samples per chirp



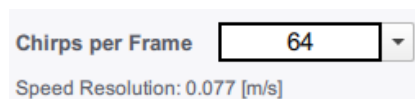
Samples per Chirp

Theoretical Max Range: 9.6 [m] ⓘ

Influences the theoretical maximum range (other parameters are the band-width and the sampling rate).

Why theoretical: because the maximum range also depends on the environment noise.

Chirps per frame



Chirps per Frame

Speed Resolution: 0.077 [m/s]

Influences the speed resolution (the more chirps per frame, the better the resolution).

Chirp repetition time



Chirp Repetition Time [μs]

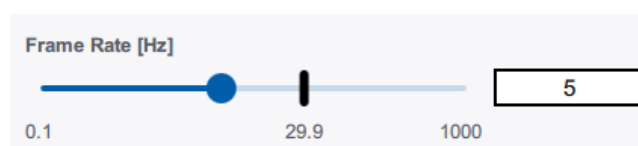
16 33 15000

506

Max Speed: 2.45 [m/s]

Influences the maximum speed that can be detected.

Frame rate



Frame Rate [Hz]

0.1 29.9 1000

5

No direct influence on the performance of the measurement, but the more frames we get, the more data processing we can do on the measurement to reduce the noise.

Start and end frequency

These settings influent the range resolution (resolution when computing distance to a target).

