XMC1000 RadarSense2Go Framework for BGT24

Michael Abler
PMM IMC ACE CES

RadarSense2Go Framework for XMC1000

RadarSense2Go Framework for XMC1000

Overview – BGT24

BGT24 is available in different derivatives. It is a Silicon Germanium MMIC (monolithic microwave integrated circuit) for signal generation and reception, operating from 24.05 GHz up to 24.25 GHz. Taking advantage from the Doppler effect it can be used for motion detection.

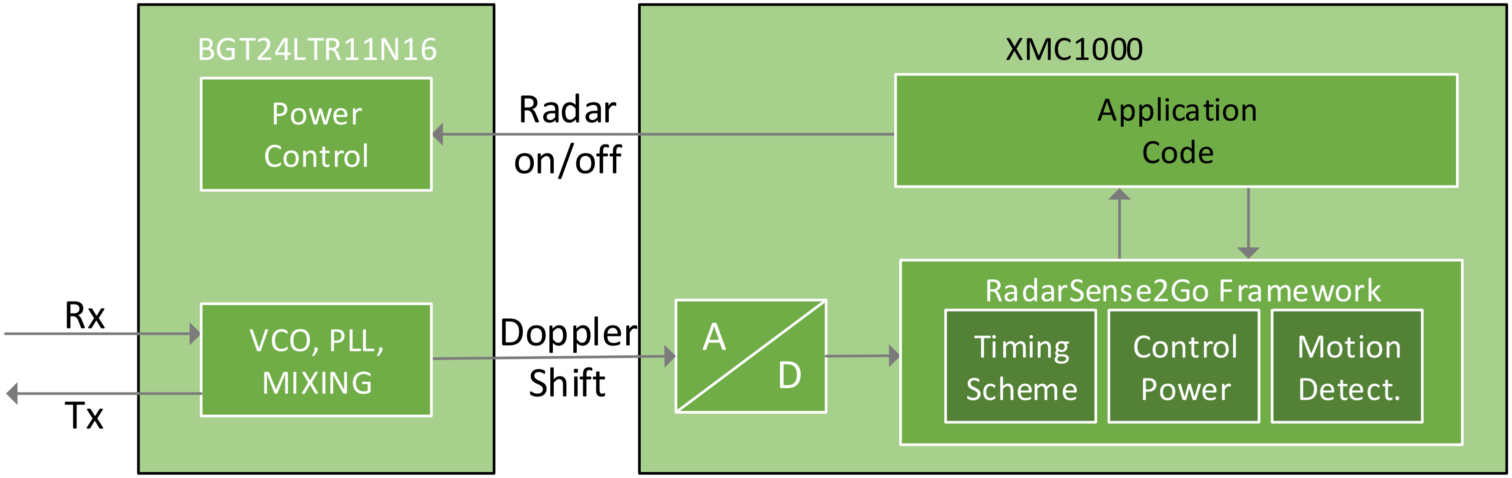
In its most simple setup it is used here, the VCO is stabilized within the ISM band by the BGT24 itself. No external PLL or microcontroller is needed for this purpose.

BGT24 transceiver is turned on/off by a signal on its input.

Once active, BGT24 provides on its output an analog signal having the corresponding component of the doppler shift frequency:

Speed	km/h	1	1.5	2	2.5	3	4	5	6	8	10
Doppler shift	Hz	44.4	66.7	88.9	111.1	133.3	177.8	222.2	266.7	355.6	444.4

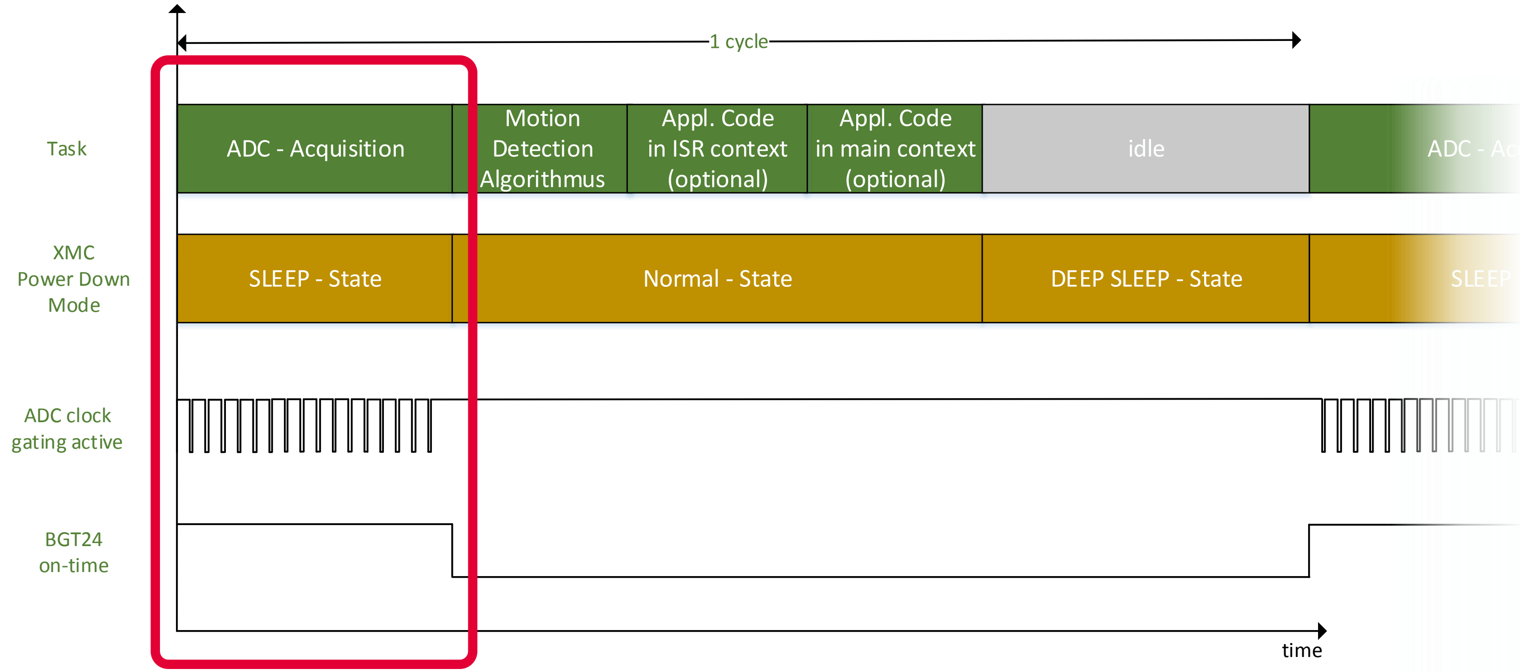
Overview – RadarSense2Go Framework



- Optimized configurable Motion Detection
  - Pre-processing interweaved with ADC sampling to extend DEEP SLEEP time.
  - Configurable 2^n-FFT for optimized application use case
- Optimized Power Control
  - Apply SLEEP and DEEP SLEEP for µC whenever possible
  - Clock gating of ADC wherever possible
- Easy configurable timing scheme optimized to cooperate with application
  - Callbacks to application
  - Application code executable in ISR and main context

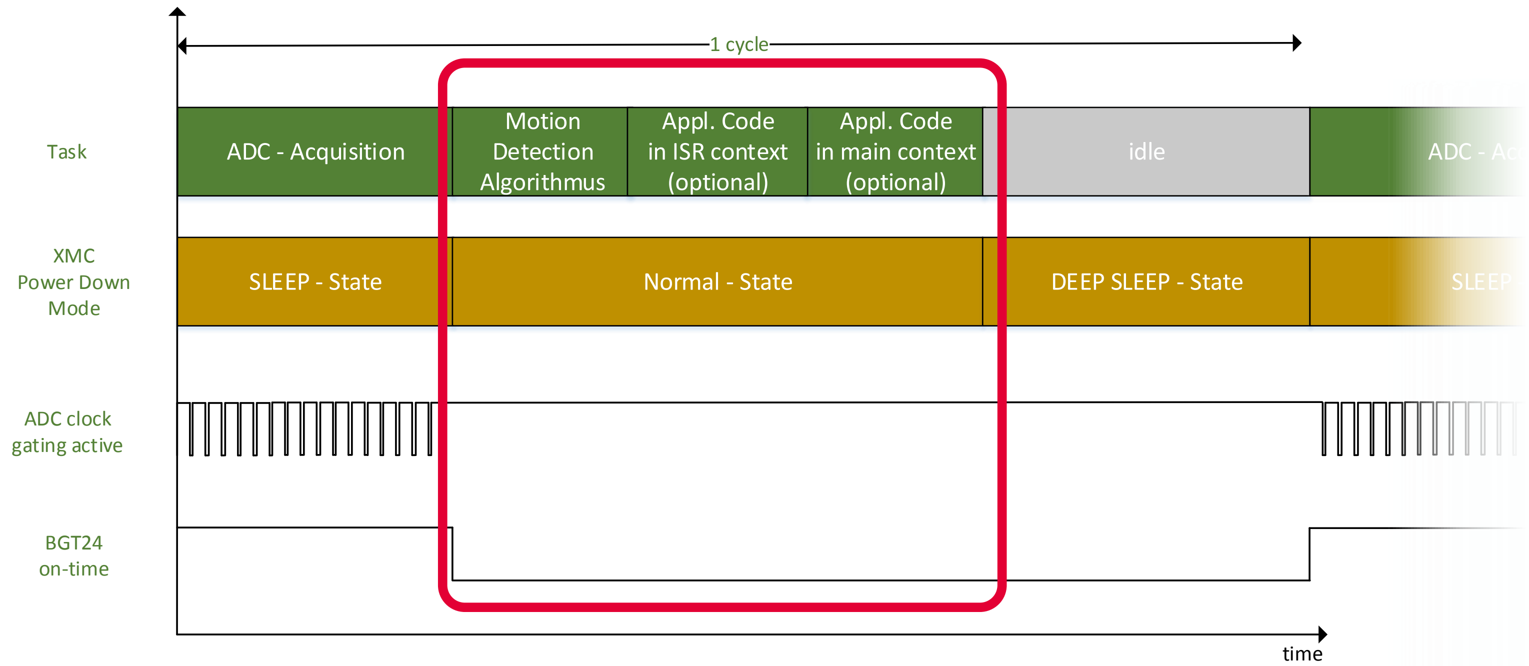
RadarSense2Go Framework for XMC1000

Motion detection – Timing scheme



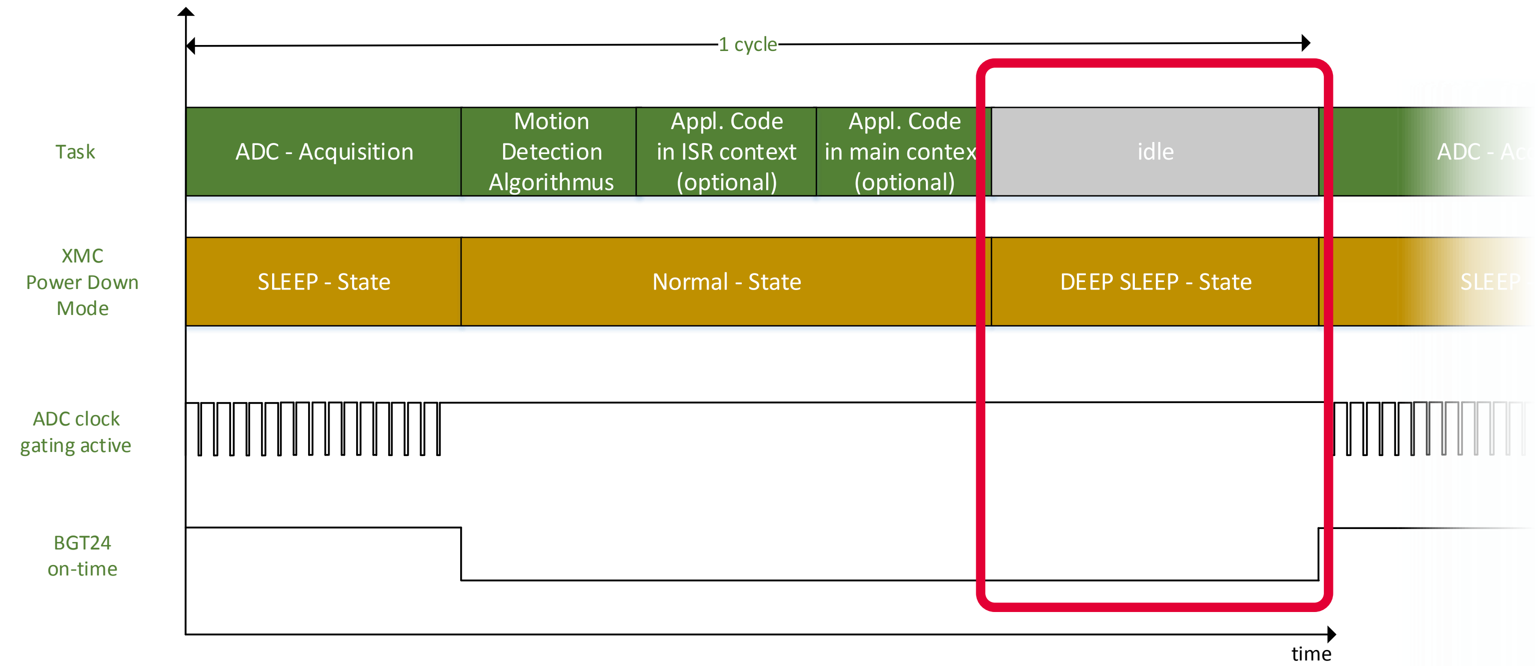
- XMC mostly in SLEEP-state; BGT24 in on-state
- Wake-up for some µs only to pick up ADC value
- ADC clock gating active when no conversion ongoing

Motion detection – Timing scheme



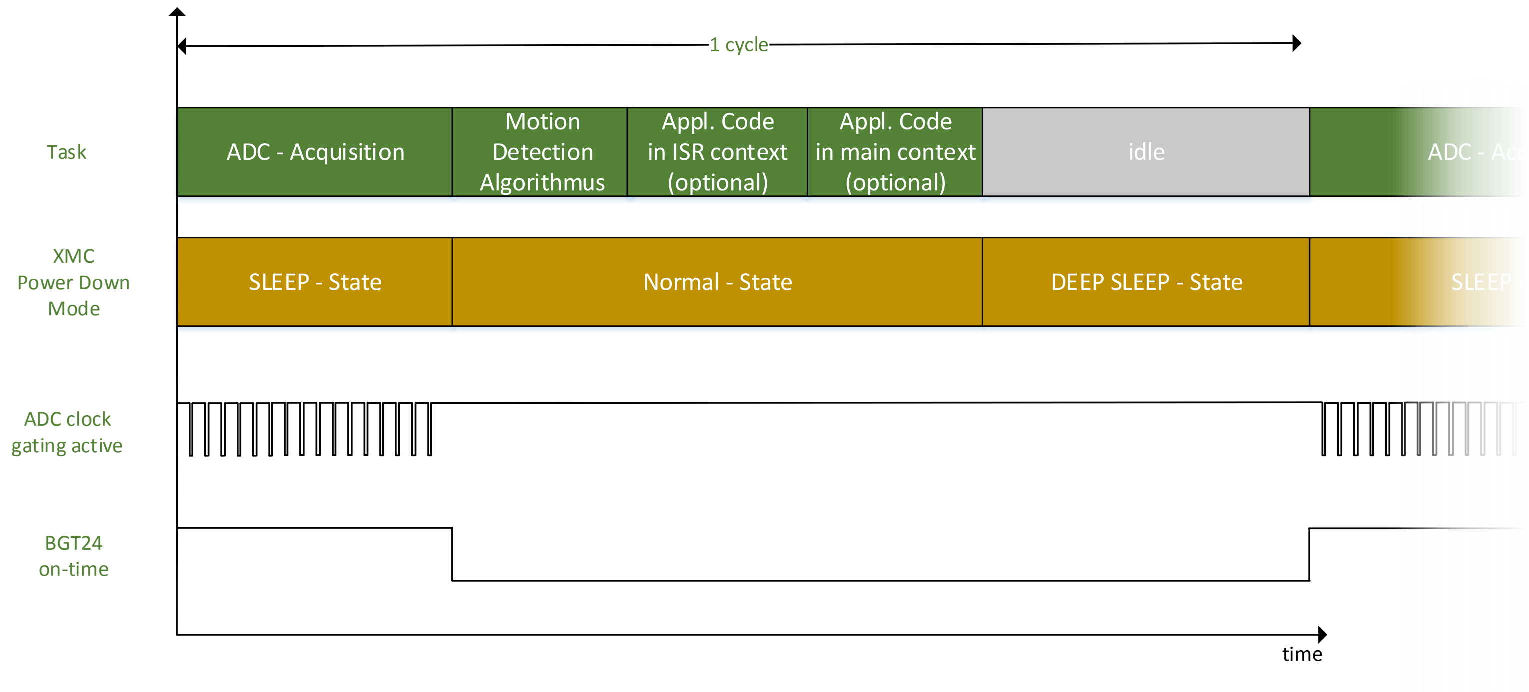
- XMC normal state; BGT24 in off-state
- Motion detection algorithmus
  - pre-processing (hanning-window, mean-filter)
  - FFT-processing to detect doppler frequency
  - post-processing (ampl.-calc., max-finding, threshold-trigger)
- Optional application code processed either in user or main-context

Motion detection – Timing scheme



- XMC in DEEP SLEEP state; BGT24 in off-state
- Wait for next cycle triggered by RTC

Motion detection – Timing scheme



- Hint: Box sizes inside this diagram do not match the relation to real typical timing figures
  Please see next slides for real life figures!

Motion detection – Real Life Timing Scheme

Motion Detection Algorithmus

BGT24 on-time

ADC clock gating

44Hz input signal

- Typical timing scheme:

f_{cycle} = 100ms

BGT24 on-time(FFT32) = 32 * 710 µs = 22.72 ms

f_{min} = 1.408kHz / 32 = 44Hz

f_{ADC} = 1.408kHz → T_{ADC} = 710 µs

Motion detection – Real Life Timing Scheme

Motion Detection Algorithmus

BGT24 on-time

ADC clock gating

44Hz input signal

› Typical timing scheme:

f_{cycle} = 100ms

BGT24 on-time(FFT32) = 32 * 710 µs = 22.72 ms

f_{min} = 1.408kHz / 32 = 44Hz

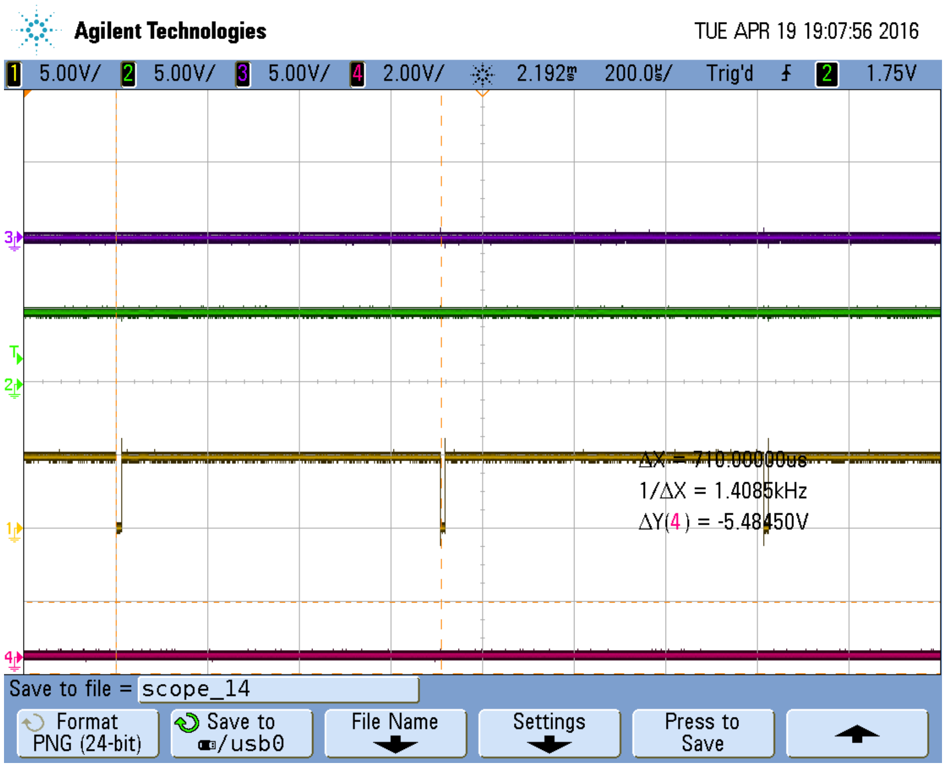
f_{ADC} = 1.408kHz → T_{ADC} = 710 µs

Motion detection – Real Life Timing Scheme

Motion Detection Algorithmus

BGT24 on-time

ADC clock gating



› Typical timing scheme:

f_{cycle} = 100ms

BGT24 on-time(FFT32) = 32 * 710 µs = 22.72 ms

f_{min} = 1.408kHz / 32 = 44Hz

f_{ADC} = 1.408kHz → T_{ADC} = 710 µs

Motion detection – Real Life Timing Scheme

Motion Detection Algorithmus

BGT24 on-time

ADC clock gating

1. µCore & ADC on-time during Acquisition phase: 32 x 10,6 µs

~ 0,340ms

› Typical timing scheme:

f_{cycle} = 100ms

BGT24 on-time(FFT32) = 32 * 710 µs = 22.72 ms

f_{min} = 1.408kHz / 32 = 44Hz

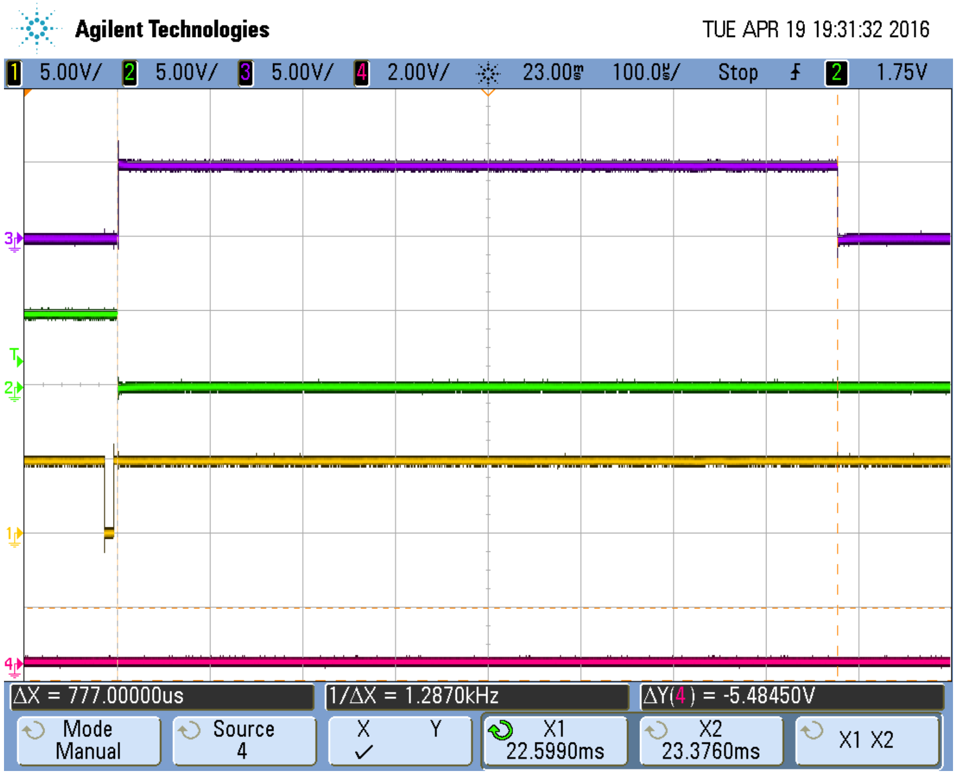
f_{ADC} = 1.408kHz → T_{ADC} = 710 µs

Motion detection – Real Life Timing Scheme

Motion Detection Algorithmus

BGT24 on-time

ADC clock gating



1. µCore & ADC on-time during Acquisition phase: 32 x 10,6 µs
~ 0,340ms

2. µCore on-time during motion detection algorithm:
~ 0,777 ms

- Typical timing scheme:

  f_{cycle} = 100ms

  BGT24 on-time(FFT32) = 32 * 710 µs = 22.72 ms

  f_{min} = 1.408kHz / 32 = 44Hz

  f_{ADC} = 1.408kHz → T_{ADC} = 710 µs

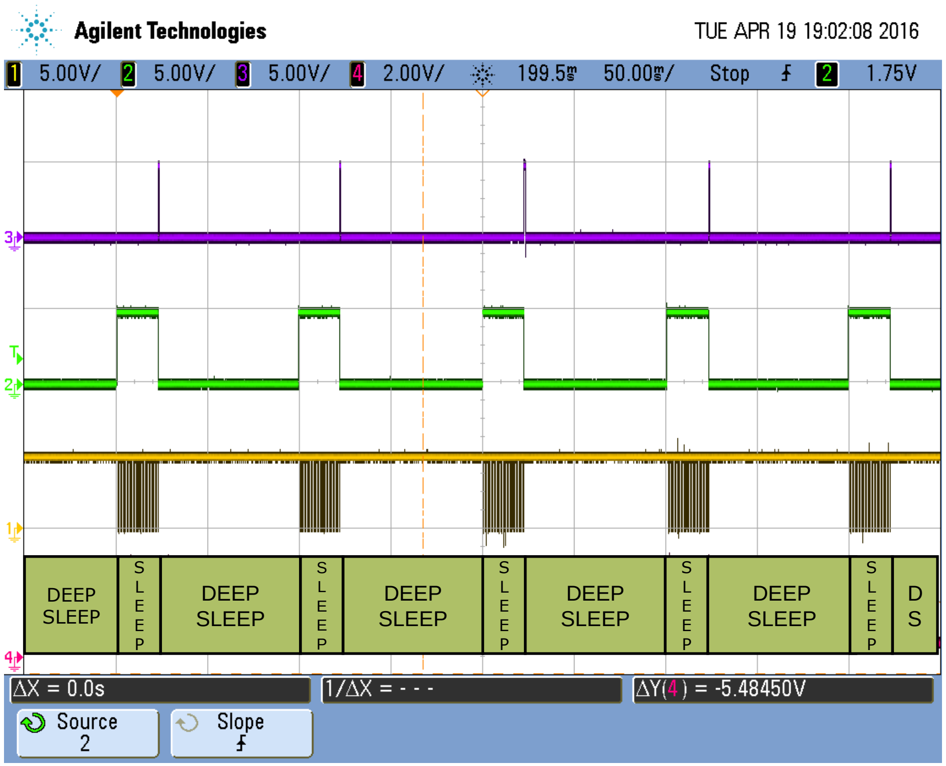
Motion detection – Real Life Timing Scheme

Motion Detection Algorithmus

BGT24 on-time

ADC clock gating

XMC Power Down Mode



1. µCore & ADC on-time during Acquisition phase: 32 x 10,6 µs
~ 0,340ms

2. µCore on-time during motion detection algorithm:
~ 0,777 ms

On-time of µCore & ADC is less than 1% of overall cycle time.

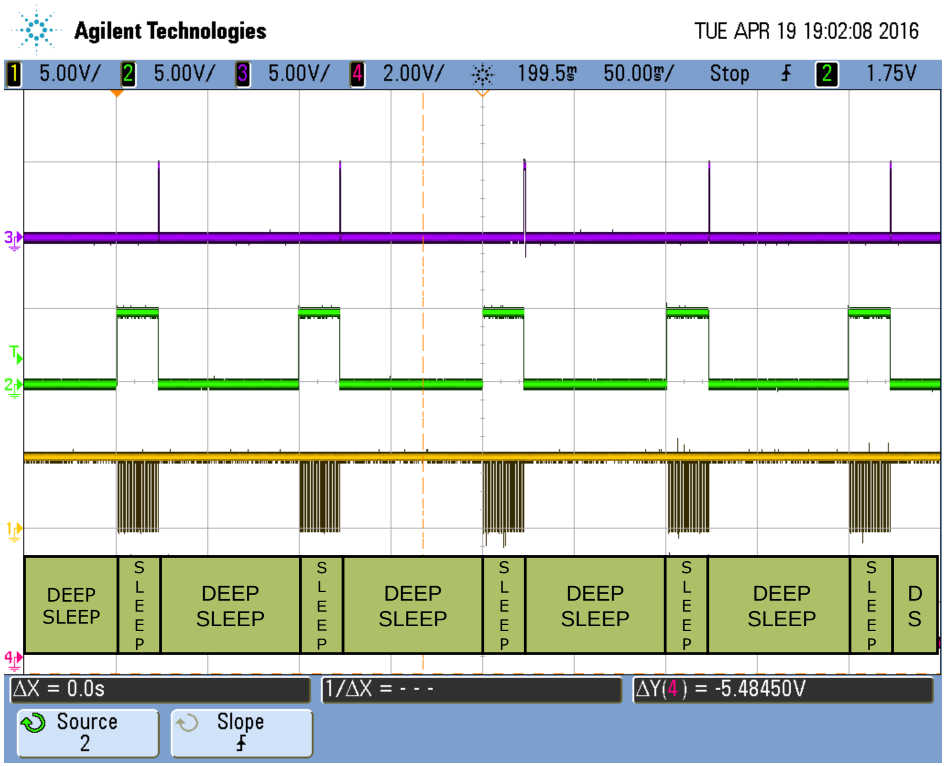
Motion detection – Real Life Timing Scheme

Motion Detection Algorithmus

BGT24 on-time

ADC clock gating

XMC Power Down Mode



1. µCore & ADC on-time during Acquisition phase: 32 x 10,6 µs
~ 0,340ms

2. µCore on-time during motion detection algorithm (n=5; 32 samples):
~ 0,777 ms
(timing scales lineary with samples)

On-time of µCore & ADC is less than 1% of overall cycle time.

RadarSense2Go Framework for XMC1000

Timing setup

BGT24 provides an analog signal. Depending on the object speed the following frequency components are present inside the signal:

Speed	km/h	1	1.5	2	2.5	3	4	5	6	8	10
Doppler shift	Hz	44.4	66.7	88.9	111.1	133.3	177.8	222.2	266.7	355.6	444.4

$$\boxed{f_{Doppler} = V_{Object} * 44.4 \text{ [Hz*h/km]}}$$

Timing setup

1. Maximum frequency you need to be able to detect is defined by the maximum speed of the object you want to detect.

2. Minimum frequency you need to be able to detect is defined by the minimum speed of the object you want to detect.

3. Minimum frequency delta you need to be able to meassure is defined by the minimum speed delta you want to meassure.

Timing setup

1. Maximum frequency you need to be able to detect is defined by the maximum speed of the object you want to detect.

$$\boxed{f_{ADC} \geq f_{Objectmax} * 2 \text{ (Nyquist criteria)}}$$

2. Minimum frequency you need to be able to detect is defined by the minimum speed of the object you want to detect.

$$f_{Objectmin} = f_{ADC} / 2^{n_{FFT}} \rightarrow \boxed{n_{FFT} \geq \log_2(f_{ADC} / f_{Objectmin})}$$

~~3. Minimum frequency delta you need to be able to meassure is defined by the minimum speed delta you want to meassure.~~

→ no relevance for pure motion detection

Timing setup

I. $f_{ADC} \geq f_{Objectmax}*2$ (Nyquist criteria)

II. $n_{FFT} \geq \log_2(f_{ADC} / f_{Objectmin})$

Hints:

For pure motion detection nyquist criteria is not such a strict criteria, because anyway higher frequencies are convulated back and still can be detected as long as $f_{ADC} >> f_{Objectmax}$.

The lower frequencies you want to meassure, the longer the on-time of BGT24 will be.

Timing setup – Example

V_{max} = 10 km/h → f_{max} = 444,4 Hz

V_{min} = 0,31 km/h → f_{min} = 13,8 Hz

Result:

I. $f_{ADC} \geq$ 888,8 Hz

II. $n_{FFT} \geq$ 6 bit

BGT24 on-time:

64 /888,8 Hz ~ 72 ms

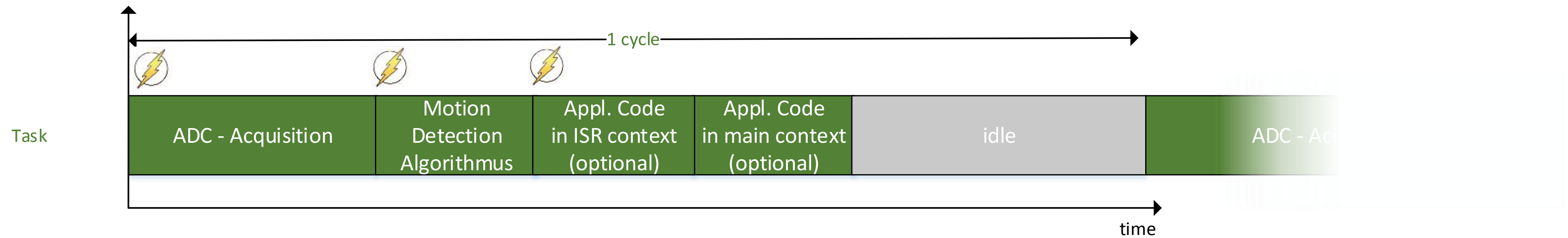
µCore on-time (FFT64):

1,6 ms(FFT) + 64 * 10,6 µs(ADC)

~ 2,3 ms

RadarSense2Go Framework for XMC1000

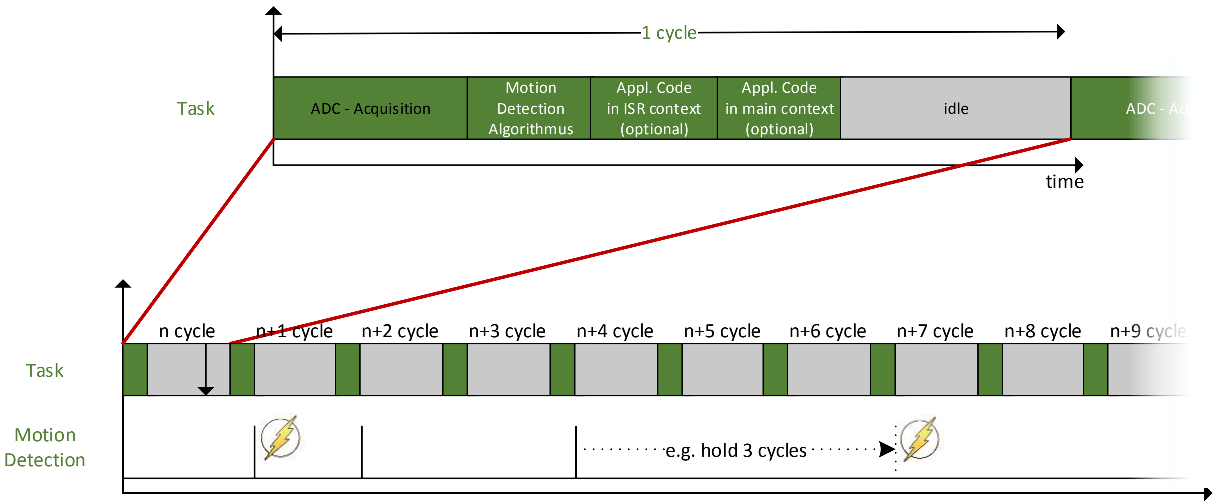
Callbacks executed inside timing scheme in ISR-context



Inside the timing scheme your application can register to callback-functions.
All callback functions are executed inside ISR context:

1. Before ADC acquisition starts.
   main purpose: Switch BGT24 to on-state by port-pin.
2. After ADC acquisition has finished.
   main purpose: Switch BGT24 to off-state by port-pin.
3. After motion detection algorithm has finished (optional).
   main purpose: Get result of motion detection for e.g. calibration
   - Amplitude spectrum result of FFT.
   - Peak magnitude inside amplitude spectrum.
   - Frequency of peak inside amplitude spectrum.
   - Motion detection result of last measurement.

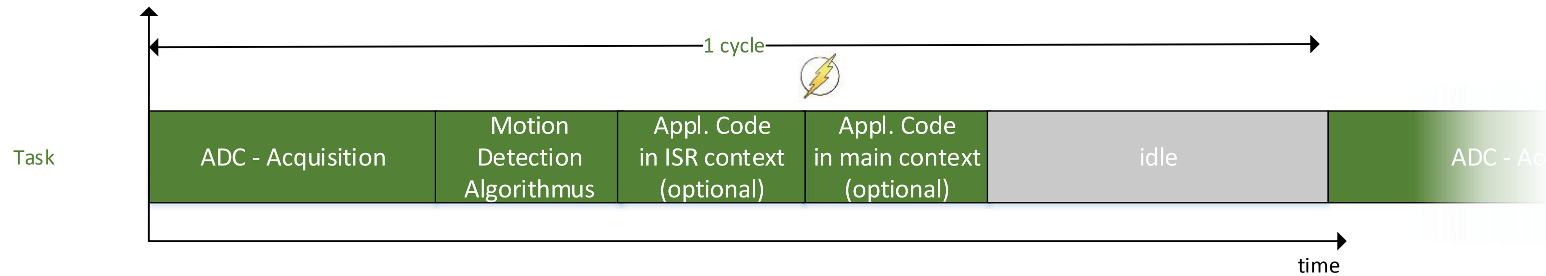
Callbacks executed inside timing scheme in ISR-context



You can define a hold-on time filter for the motion detection result. Register a callback to receive regular updates on the filtered detect result. You will receive a call on this callback for the first detection to indicate the positive detection. Once there was no motion detected for the configurable number of cycles (e.g. 3 cycles) you will get another call, indicating the negative detection.

4. Receive filtered motion detect result.
   main purpose: Filtered result of the motion detection state.

Executing application code inside main context



During initialization of the framework you can enable execution of your application code inside main context.

When the ISR-context execution has finished, your code will automatically proceed, where it has been stopped before.

Once you finished your task inside main context you just call a framework API-function to proceed with the scheme.
(here: idle/DEEP SLEEP-state).

RadarSense2Go Framework for XMC1000

1 Overview

2 Understanding motion detection timing scheme

3 Defining the timing

4 Integrating application and library

5 Hands on – The RadarSense2Go framework

Hands on the RadarSense2Go Framework

During initialization the framework can be configured in the following aspects:

- Timing
  - ADC sampling time in µs
  - Cycle time of timing scheme in ms
  - Number of samples 2^n ($n = 3 – 8$)
- Motion Detection algorithm and sensitivity
  - Number of hold-on cycles for filtering
  - Threshold to trigger detection
  - Enable square root calculation on amplitudes (disable to save µCore time)
- Power Saving options (to disable feature for development)
  - Enable sleep / deep-sleep state inside timing scheme
  - Enable VADC clock gating inside timing scheme
  - Enable execution of main loop

Hands on the RadarSense2Go Framework

```
/******************************************************************************************/

XMC_RADARSENSE2GOL_TIMING_t radarsense2gol_timing =
{
        .t_sample_us = (1 / SAMPLING_FREQ_HZ) * 1000 * 1000, /* sample time in us = (1/sample_frequency) * 1000 * 1000 */
        .t_cycle_ms = 300,           /* 300 ms */
        .N_exponent_samples = 8     /* 2^8samples * 710us = 182ms BGT24 on-time (settle-time ignored) */
};

XMC_RADARSENSE2GOL_ALG_t radarsense2gol_algorithm =
{
        .hold_on_cycles = 1,      /* hold-on cycles to trigger detection */
        .trigger_det_level = DETECTION_THRESHOLD,  /* trigger detection level */
        .rootcalc_enable = XMC_RADARSENSE2GOL_DISABLED /* root calculation for magnitude disabled */
};

XMC_RADARSENSE2GOL_POWERDOWN_t radarsense2gol_powerdown =
{
        .sleep_deepsleep_enable   = XMC_RADARSENSE2GOL_ENABLED, /* sleep / deepsleep enabled */
        .mainexec_enable          = XMC_RADARSENSE2GOL_ENABLED, /* main exec enabled */
        .vadc_clock_gating_enable = XMC_RADARSENSE2GOL_ENABLED  /* vadc clock gating enabled */
};

/******************************************************************************************/
```

Setup the framework

1. Timings
2. Detection threshold
3. Sleep/Clock gating

Hands on the RadarSense2Go Framework– nothing but a simple main

```c
/***********************************************************************************************/

int main(void)
{
    bool running = false;

    DAVE_Init(); /* Initialization of DAVE APPs  */

    // turn off all leds
    DIGITAL_IO_SetOutputHigh(&LED_ORANGE);
    DIGITAL_IO_SetOutputHigh(&LED_RED);
    DIGITAL_IO_SetOutputHigh(&LED_BLUE);

    // turn on BGT
    DIGITAL_IO_SetOutputLow(&BGT24);

    radarsense2gol_init(
            radarsense2gol_timing,
            radarsense2gol_algorithm,
            radarsense2gol_powerdown,
            &TIMER_0
    );

    // register call backs
    radarsense2gol_regcb_result ( radarsense2gol_result );

    while (1)
    {
        if (running == false)
        {
            if (g_start == true)
            {
                running = true;
                radarsense2gol_start();
            }
```

Initialize the framework

Hands on the RadarSense2Go Framework– nothing but a simple main

```
/***********************************************************************************************************/

int main(void)
{
    bool running = false;

    DAVE_Init(); /* Initialization of DAVE APPs  */

    // turn off all leds
    DIGITAL_IO_SetOutputHigh(&LED_ORANGE);
    DIGITAL_IO_SetOutputHigh(&LED_RED);
    DIGITAL_IO_SetOutputHigh(&LED_BLUE);

    // turn on BGT
    DIGITAL_IO_SetOutputLow(&BGT24);

    radarsense2gol_init(
            radarsense2gol_timing,
            radarsense2gol_algorithm,
            radarsense2gol_powerdown,
            &TIMER_0
    );

    // register call backs
    radarsense2gol_regcb_result ( radarsense2gol_result );

    while (1)
    {
        if (running == false)
        {
            if (g_start == true)
            {
                running = true;
                radarsense2gol_start();
            }
```

Register your callbacks

Hands on the RadarSense2Go Framework– nothing but a simple main

```
// register call backs
radarsense2gol_regcb_result ( radarsense2gol_result );

while (1)
{
    if (running == false)
    {
        if (g_start == true)
        {
            running = true;
            radarsense2gol_start();
        }
    }
    else
    {
        if (g_start == false)
        {
            running = false;
            radarsense2gol_stop();
        }

        radarsense2gol_set_detection_threshold(radarsense2gol_algorithm.trigger_det_level);

        /* place your application code for main execution here */
        /* e.g. communication on peripherals */
        radarsense2gol_exitmain(); /* only need to be called if
                              mainexec_enable is enabled during init */
    }

}
```

Start the timing scheme

Hands on the RadarSense2Go Framework– nothing but a simple main

```
// register call backs
radarsense2gol_regcb_result ( radarsense2gol_result );

while (1)
{
    if (running == false)
    {
        if (g_start == true)
        {
            running = true;
            radarsense2gol_start();
        }
    }
    else
    {
        if (g_start == false)
        {
            running = false;
            radarsense2gol_stop();
        }

        radarsense2gol_set_detection_threshold(radarsense2gol_algorithm.trigger_det_level);

        /* place your application code for main execution here */
        /* e.g. communication on peripherals */
        radarsense2gol_exitmain(); /* only need to be called if
                                  mainexec_enable is enabled during init */
    }

}
```

radarsense2go_exitmain()

- to proceed with the timing scheme (DEEP SLEEP)

Hands on the RadarSense2Go framework – register callbacks

```
/*********************************************************************************************/
// callback executed after new data is available from algorithm
void radarsense2gol_result( uint32_t *fft_magnitude_array,
        uint16_t size_of_array_mag,
        int16_t *adc_aqc_array_I,
        int16_t *adc_aqc_array_Q,
        uint16_t size_of_array_acq,
        XMC_RADARSENSE2GOL_MOTION_t motion,
        uint32_t max_frq_mag,
        uint32_t max_frq_index)
{
    // copy raw data and fft data and motion indicator to global variables used in micrium GUI
    memcpy(g_sampling_data_I, adc_aqc_array_I, size_of_array_acq * sizeof(uint16_t));
    memcpy(g_sampling_data_Q, adc_aqc_array_Q, size_of_array_acq * sizeof(uint16_t));
    memcpy(g_fft_data, &fft_magnitude_array[1], (size_of_array_mag - 1) * sizeof(uint32_t));
    g_motion = motion;

    // calc doppler frequency and velocity
    g_doppler_frequency = calcDopplerFrequency(max_frq_index);
    g_doppler_velocity = calcDopplerSpeed(g_doppler_frequency);

    /* Dump raw IQ ADC samples to HOST PC via UART
     * UART Configurations: Full-duplex, Direct mode, 9600 baud rate, 8 data-bits, 1 stop-bit, no parity
     * Data format of transmission: Completely transmits I_adc samples of buffer length BUFF_SIZE,
     * followed by Q_adc samples of same buffer length of BUFF_SIZE
     * so in total 2 * BUFF_SIZE samples are transmitted
     */
    dumpRawIQ_uint16(g_sampling_data_I, g_sampling_data_Q, (uint16_t)BUFF_SIZE);
```

Receive the results for every cycle of the scheme and process it.

Hands on the RadarSense2Go framework – register callbacks

main.c

```c
void radarsense2go_result( uint32_t *magnitude_array,
                           uint16_t size_of_array_mag,
                           int16_t *adc_aqc_array,
                           uint16_t size_of_array_acq,
                           XMC_RADARSENSE2GO_MOTION_t motion,
                           uint32_t max_frq_mag,
                           uint32_t max_frq_index )
{
  /* place your application code for ISR context execution here */
  /* e.g. threshold calibration */
  return;
}

void radarsense2go_startacq(void)
{
  static uint32_t BGT24_settle;
  /* Turn BGT24 on */
  DIGITAL_IO_SetOutputLow(&BGT24);
  /* delay until BGT24 is settled */
  BGT24_settle=150000;
  while(BGT24_settle!=0)
      BGT24_settle--;
  return;
}

void radarsense2go_endacq(void)
{
  /* BGT24 off time */
  DIGITAL_IO_SetOutputHigh(&BGT24);
  return;
}

void radarsense2go_trigger(XMC_RADARSENSE2GO_MOTION_t detection_state)
{
  motion_last=detection_state;
  if (detection_state==XMC_MOTION_DETECT)
  {
    DIGITAL_IO_SetOutputLow(&LED);
  }
  else
  {
    DIGITAL_IO_SetOutputHigh(&LED);
  }
  return;
}
```

Turn BGT24 on/off when ADC acquisition starts/stops. After turning BGT24 on some delay is needed for settling.

Hands on the RadarSense2Go framework – register callbacks

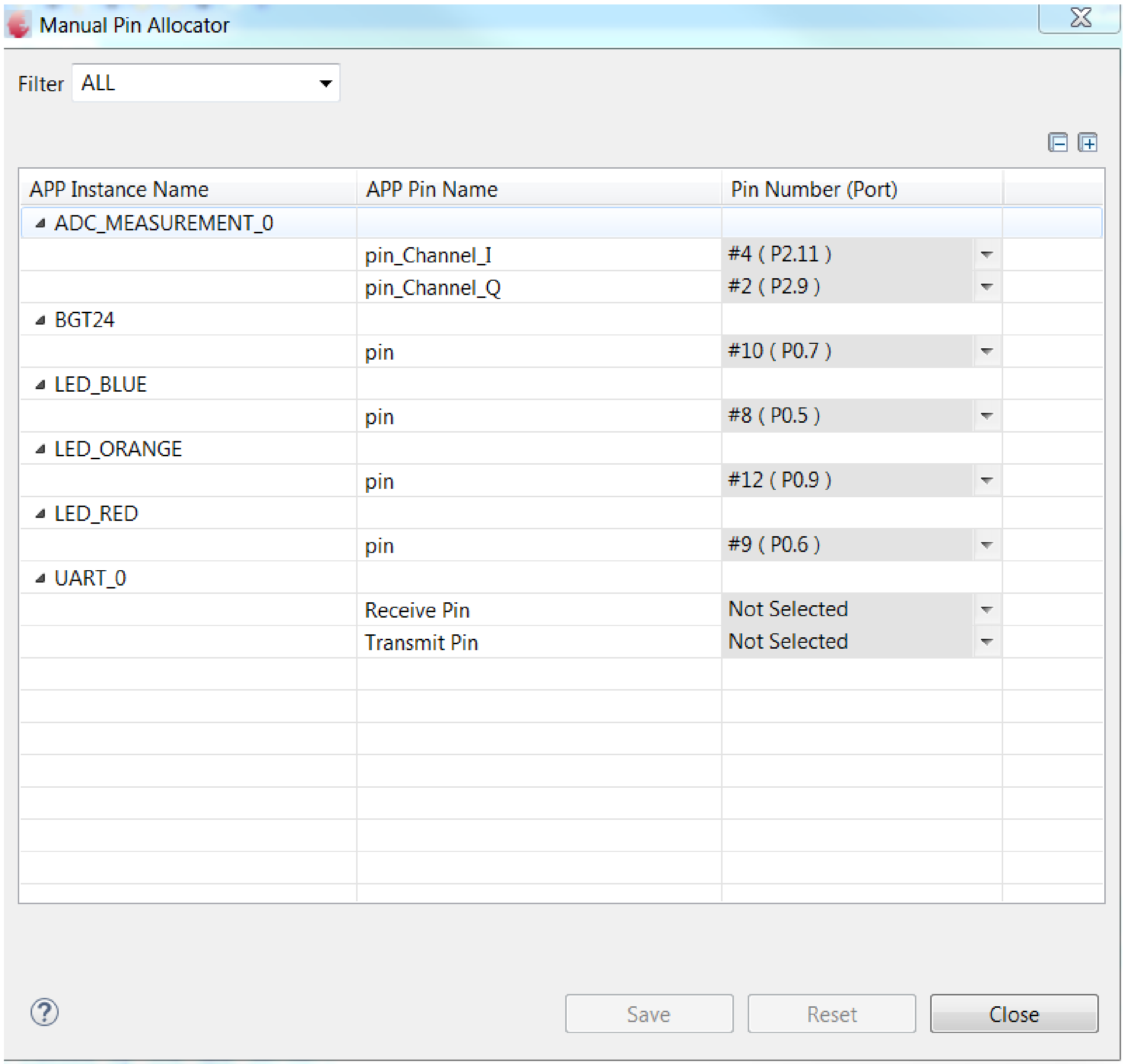
main.c

```c
void radarsense2go_result( uint32_t *magnitude_array,
                           uint16_t size_of_array_mag,
                           int16_t *adc_aqc_array,
                           uint16_t size_of_array_acq,
                           XMC_RADARSENSE2GO_MOTION_t motion,
                           uint32_t max_frq_mag,
                           uint32_t max_frq_index )
{
  /* place your application code for ISR context execution here */
  /* e.g. threshold calibration */
  return;
}

void radarsense2go_startacq(void)
{
  static uint32_t BGT24_settle;
  /* Turn BGT24 on */
  DIGITAL_IO_SetOutputLow(&BGT24);
  /* delay until BGT24 is settled */
  BGT24_settle=150000;
  while(BGT24_settle!=0)
      BGT24_settle--;
  return;
}

void radarsense2go_endacq(void)
{
  /* BGT24 off time */
  DIGITAL_IO_SetOutputHigh(&BGT24);
  return;
}

void radarsense2go_trigger(XMC_RADARSENSE2GO_MOTION_t detection_state)
{
  motion_last=detection_state;
  if (detection_state==XMC_MOTION_DETECT)
  {
    DIGITAL_IO_SetOutputLow(&LED);
  }
  else
  {
    DIGITAL_IO_SetOutputHigh(&LED);
  }
  return;
}
```

Process the trigger when motion is detected and not detected.

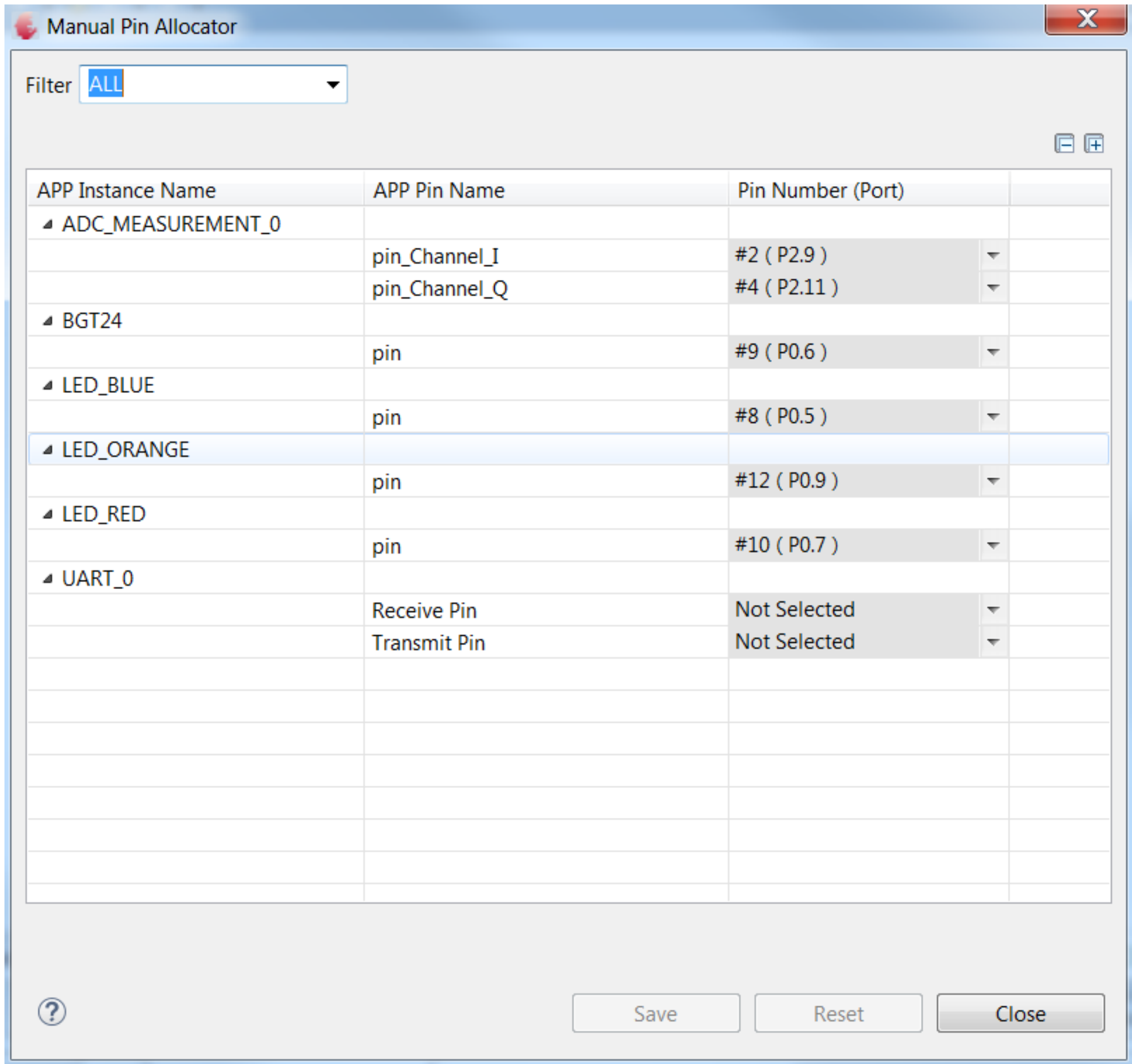
Here as an example a LED is turned on/off.

Sense2GoL ED01 Pin Mapping



APP Instance Name	APP Pin Name	Pin Number (Port)
ADC_MEASUREMENT_0		
	pin_Channel_I	#4 (P2.11)
	pin_Channel_Q	#2 (P2.9)
BGT24		
	pin	#10 (P0.7)
LED_BLUE		
	pin	#8 (P0.5)
LED_ORANGE		
	pin	#12 (P0.9)
LED_RED		
	pin	#9 (P0.6)
UART_0		
	Receive Pin	Not Selected
	Transmit Pin	Not Selected

Sense2GoL V1.2 Pin Mapping



Manual Pin Allocator

Filter ALL

APP Instance Name	APP Pin Name	Pin Number (Port)
ADC_MEASUREMENT_0		
	pin_Channel_I	#2 (P2.9)
	pin_Channel_Q	#4 (P2.11)
BGT24		
	pin	#9 (P0.6)
LED_BLUE		
	pin	#8 (P0.5)
LED_ORANGE		
	pin	#12 (P0.9)
LED_RED		
	pin	#10 (P0.7)
UART_0		
	Receive Pin	Not Selected
	Transmit Pin	Not Selected

Save Reset Close

Part of your life. Part of tomorrow.