VL6180 API Integration Guide

life.augmented

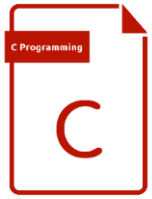
History

Version	Date	Comments
1.0	19 June 2015	Initial
1.2	10 July 2015	- Added API benefits, description of scaling, WAF and Dmax - Added error codes description - Now covering both VL6180X and VL6180 products - Updated for API 3.1.0 (new features) - Added slides to describe porting from API 3.0.2 to API 3.1.0
1.3	11 August 2015	- Focus only on VL6180
1.4	13 October 2015	- API 3.2.0 update : errorStatus 17 removed

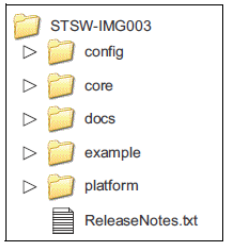
VL6180 API Description

Integration

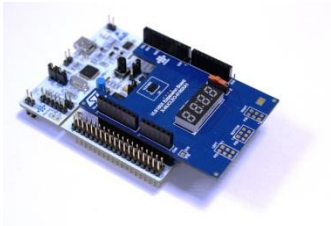
Set of documented C functions
To enable development of end-user applications through simple & high level functions : Init, Prepare, Range, Als, etc…



Structured in a way it is easily portable
Through a well isolated platform layer (mainly for low level i2C access).



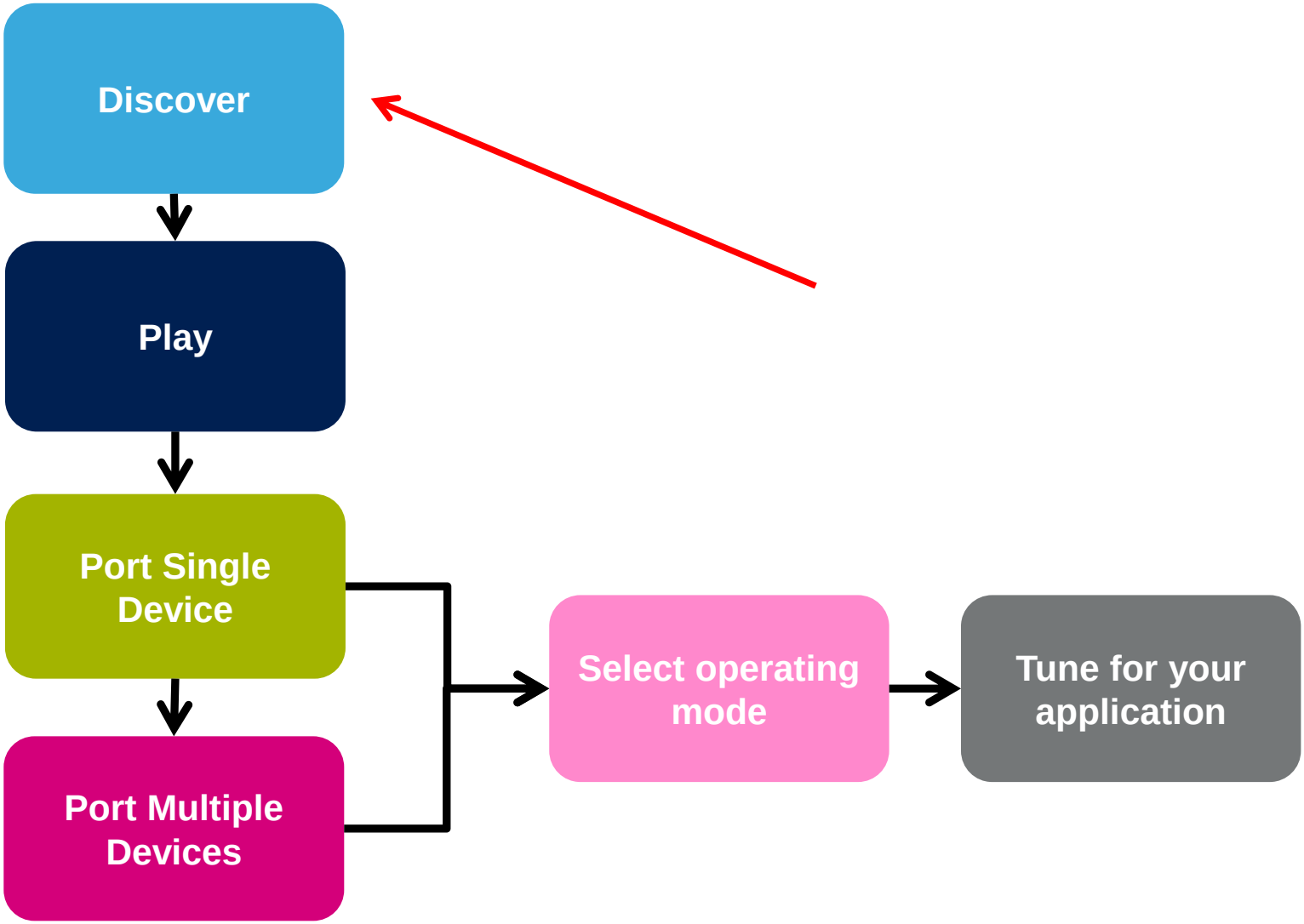
Validated on Nucleo pack
NUCLEO STM32F401RE & L053R8 with X-NUCLEO-6180XA1 Expansion Board



ALS, Ranging, and multi-satellites examples
Two examples : RangingAndAls & RangingWithSatellites example codes with pre-compiled binaries and Keil & IAR projects

life.augmented

API Integration Procedure



life.augmented

Discover

VL6180 API scope

- VL6180 API is compatible with both VL6180X and VL6180 products but is delivered as 2 different SW packages
- All API functions are prefixed by VL6180x which means either VL6180X or VL6180 devices
- What differs between the 2 packages is the static configuration of the API (see later)
  - Proximity+ALS configuration dedicated to VL6180X
  - Extended-range configuration dedicated to VL6180
- From SW point of view, the extended-range configuration could be used with VL6180X product **but** ranging performances (max distance,accuracy) are only guaranteed with VL6180 product (optimized design and factory calibration)

life.augmented

Discover

API Benefits

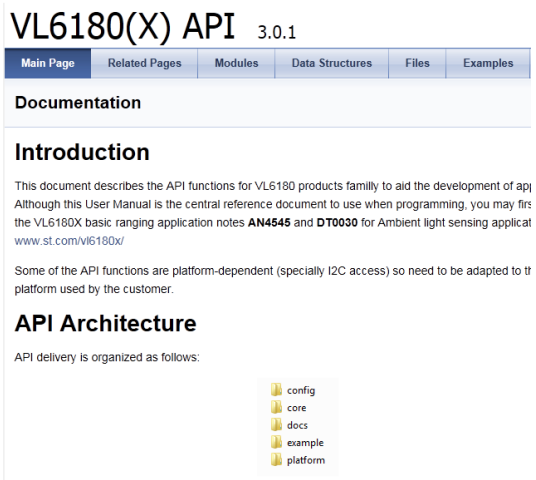
- Getting ranging from an I2C-based device looks simple but Time-Of-Flight technology is complex
- API is there to :
  - Keep your application simple
  - Ensure best performances of VL6180 device whatever the conditions
  - Ease VL6180 device integration into your platform
- API is a mandatory SW layer that implements extra features which may be key for your applications (see the “Tune for your application section”)
  - Ranging scaling
  - Wrap-around filter
  - Dmax estimation

life.augmented

Discover

API Documentation

- Unzip VL6180_API_x.x.x_Extended_Range.zip
- Read API documentation in docs directory
  - chm
  - html
- Focus your attention on following pages in this order
  - Main Page
  - Examples
  - Modules



life.augmented

Discover

API Functions

- API functions are grouped in different modules
  - API Low level functions
  - API High level functions
  - Configuration
  - Platform
- Only 3 API functions are enough to get a ranging measure from the device
  - VL6180x_InitData()
  - VL6180x_Prepare()
  - VL6180x_RangePollMeasurement()
- **Always read the *errorStatus* returned by range measurement functions to know if the returned measure is valid or not (see the “Tune for your application” section)**

Discover

API Static Configuration

- API static configuration is gathered in a single **vl6180x_cfg.h** that customer should not modify (ST guarantees the device performances only with the static configuration provided in this file)

Definition	Proximity Ranging+ALS (VL6180X)	Extended Ranging (VL6180)
VL6180x_UPSCALE_SUPPORT	-1	-3
VL6180x_ALS_SUPPORT	1	0
VL6180x_HAVE_DMAX_RANGING	1	1
VL6180x_WRAP_AROUND_FILTER_SUPPORT	1	1
VL6180x_EXTENDED_RANGE	0	1

- See next slide for more details on each parameter…

Discover

Extended-Range configuration

- Extended-Range configuration
  - #define VL6180x_UPSCALE_SUPPORT -3
    - Default scaling is x3 to enable ranging up to 500 mm (depending on conditions)
    - The minus (-3) means it is also possible to change the scaling (to x2 or x1) through the **VL6180x_UpscaleSetScaling()** API function call (in case more precision is required at low distance)
  - #define VL6180x_ALS_SUPPORT 0
    - ALS feature is disabled so all ALS-related functions of the API won't be compiled. This can help to reduce the code size if it is a concern
  - #define VL6180x_HAVE_DMAX_RANGING 1
    - Enable Dmax calculation for ranging applications (Hybrid AF). Even if this macro is set to 1, it is possible to disable this feature with the **VL6180x_DMaxSetState()** API function in case application does not need Dmax (to avoid useless calculation in the API). Setting this macro to 0 could also help to reduce the code size if it is a concern
  - #define VL6180x_WRAP_AROUND_FILTER_SUPPORT 1
    - Wrap-around filter (WAF) is enabled to filter potential wrong distances that may be reported with high reflective targets (like mirror) placed between 60 cm and 1.2 m from the device. This must be kept to 1 for Hybrid-AF applications
  - #define VL6180x_EXTENDED_RANGE 1
    - Set the device in a configuration which is optimal for long distance measurements. This must be kept to 1 for Hybrid AF applications

Discover

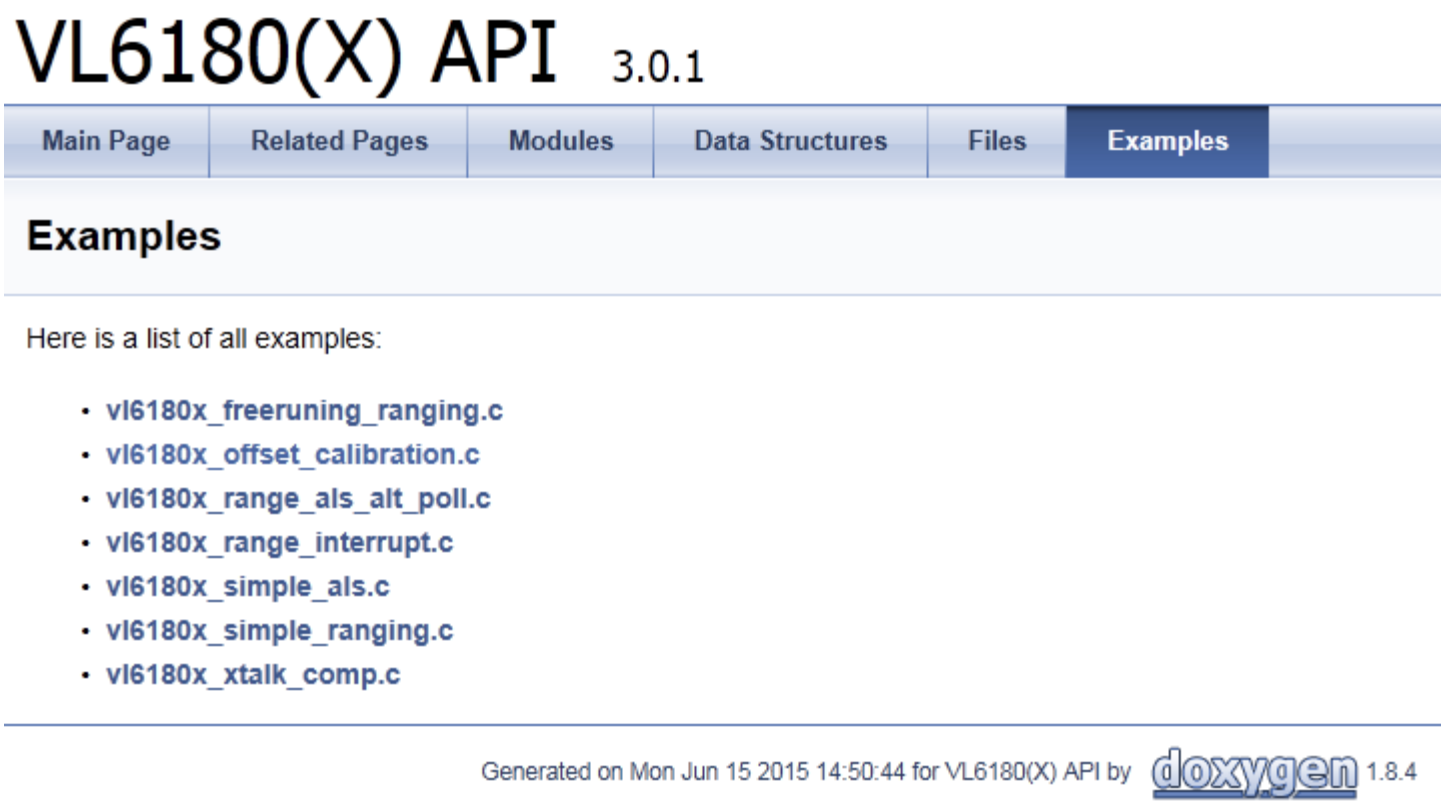
Proximity+ALS configuration

- Proximity+ALS configuration
  - #define VL6180x_UPSCALE_SUPPORT -1
    - Default scaling is x1 to enable ranging up to 200 mm
    - The minus (-1) means it is also possible to change the scaling (to x2 or x3) through the **VL6180x_UpscaleSetScaling()** API function call to extend ranging capabilities. Note that ranging performances (max distance, accuracy) with scaling x2 or x3 are not guaranteed by ST with VL6180X device (only with VL6180 device)
  - #define VL6180x_ALS_SUPPORT 1
    - ALS feature is enabled so all ALS-related functions of the API are compiled and can be used in the application.
  - #define VL6180x_HAVE_DMAX_RANGING 1
    - Enable Dmax calculation for ranging applications. Even if this macro is set to 1, it is possible to disable this feature with the **VL6180x_DMaxSetState()** API function in case application does not need Dmax (to avoid useless calculation in the API). Setting this macro to 0 could also help to reduce the code size if it is a concern.
  - #define VL6180x_WRAP_AROUND_FILTER_SUPPORT 1
    - Wrap-around filter (WAF) is enabled to filter potential wrong distances that may be reported (if WAF is off) with high reflective targets (like mirror) placed between 60 cm and 1.2 m from the device. If scaling factor is maintained at x1 (proximity ranging application), WAF should probably be disabled using the **VL6180x_FilterSetState()** API function. For higher scaling factors, WAF should probably be maintained enabled.
  - #define VL6180x_EXTENDED_RANGE 1
    - Set the device in a configuration which allows to increase scaling factor from x1 to x2 or x3

Discover

API code samples

- Have a look at the various examples from the API documentation
  - Click on the file to open source code
  - Source code is located in example/code_samples directory



life.augmented

Discover

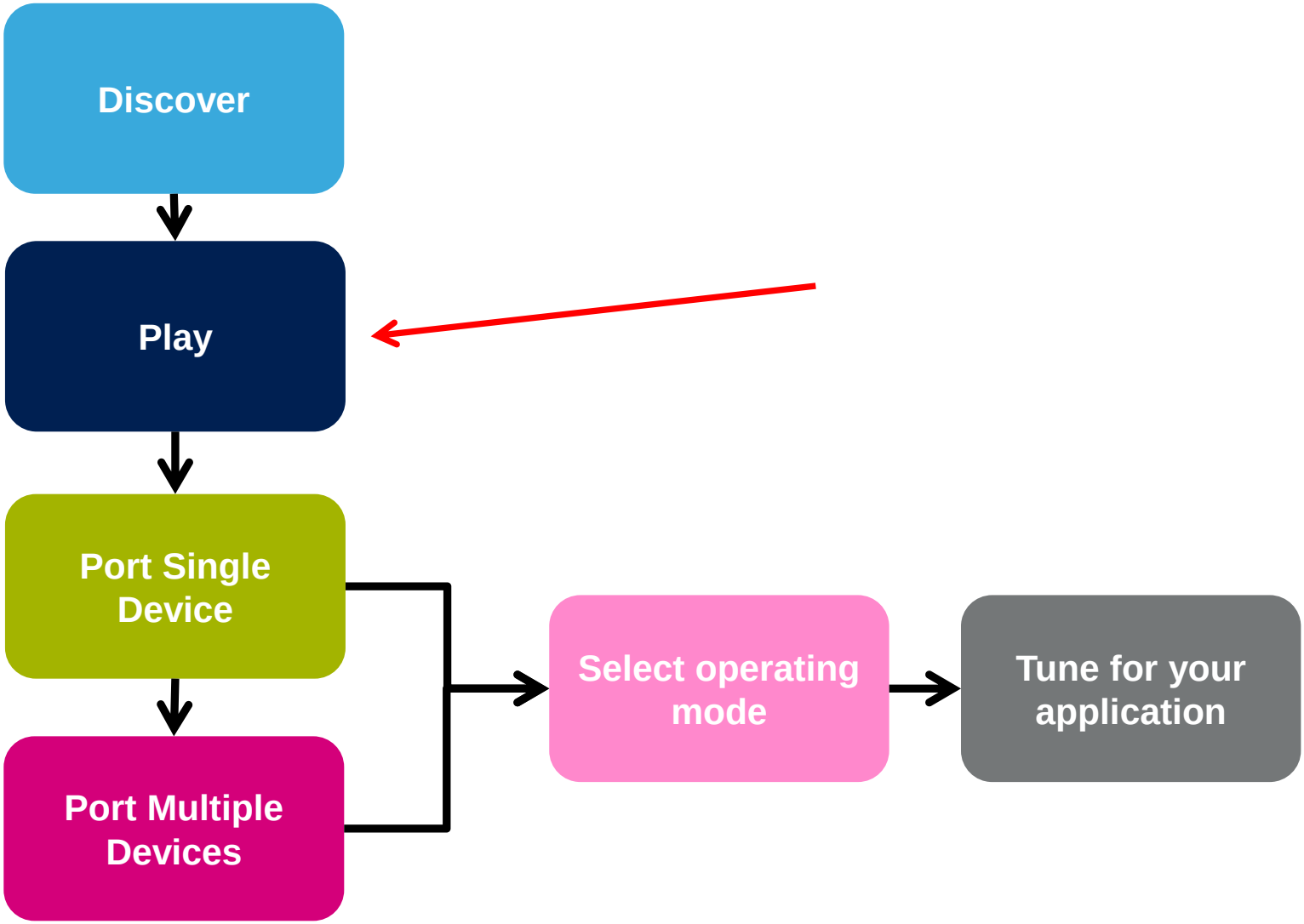
Simple ranging

- Simple Ranging : this is the code that will run in customer application !

```
#include "vl6180x_api.h"
#include "vl6180x_sample_plat.h" /* contain all device/platform specific code */

void Sample_SimpleRanging(void) {
    VL6180xDev_t myDev;
    VL6180x_RangeData_t Range;

    MyDev_Init(myDev);            // your code init device variable
    MyDev_SetChipEnable(myDev);   // your code assert chip enable
    MyDev_uSleep(1000);           // your code sleep at least 1 msec
    VL6180x_InitData(myDev);
    VL6180x_Prepare(myDev);
    do {
        VL6180x_RangePollMeasurement(myDev, &Range);
        if (Range.errorStatus == 0 )
            MyDev_ShowRange(myDev, Range.range_mm,0); // your code display range in mm
        else
            MyDev_ShowErr(myDev, Range.errorStatus); // your code display error code
    } while (!MyDev_UserSayStop(myDev)); // your code to stop looping
}
```

life.augmented

API Integration Procedure

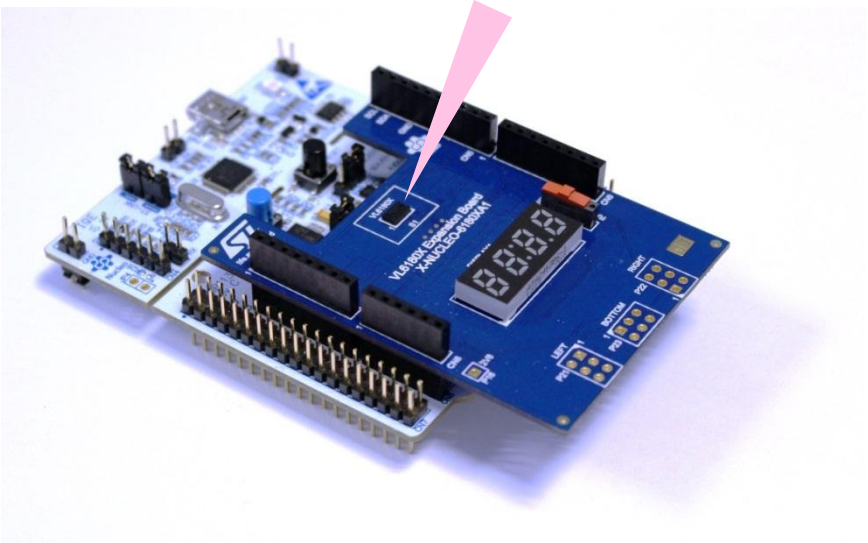


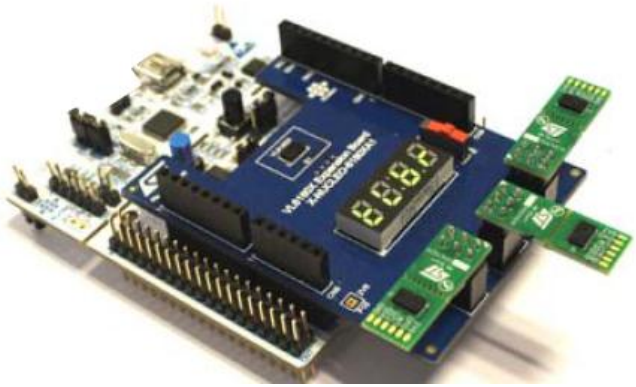
life.augmented

Play

Nucleo examples

- API package contains several “applications” examples which can be compiled and executed on a Nucleo F401 board equipped with X-NUCLEO-6180XA1 expansion board (P-NUCLEO-6180X1 pack)
  - See in example\Nucleo\Projects\Multi\Examples\VL6180X
    - RangingAndALS
      - Simple demo displaying on the 4-digit display the distance (in mm) from a target (Ranging) or the Lux level (ALS)
    - CodeSamples
      - This is a single demo gathering all sample codes execution. Pressing the blue button allows to go from one sample code to the next one
    - RangingWithSatellites
      - Demo showing how to get ranging from up to 4 devices (satellites) on the same I2C bus





life.augmented

Play

RangingAndALS Nucleo example

- Documentation is in the docs directory
  - P-NUCLEO-6180X1*.pdf
  - Focus on **VL6180X Nucleo pack software installation** chapter
- The demo is already pre-compiled so you simply need to connect Nucleo board to your PC through USB and drag&drop the demo binary
  - From example\Nucleo\Projects\Multi\Examples\VL6180X\RangingAndALS\Bin
  - To Nucleo image
  - Demo will start !
- Read documentation to get more details about the various modes of the demo
  - Ranging with various scaling factors : x1, x2, x3, Automatic
  - Ranging with alarm mode
  - ALS

Play

CodeSamples Nucleo example

- All sample codes are gathered in a single demo/project
  - example\Nucleo\Projects\Multi\Examples\VL6180X\CodeSamples
- Following slides describe in details how to
  - Compile this project
  - Flash Nucleo board
  - Execute and debug (breakpoint) each sample code

life.augmented

Play

CodeSamples : Install Keil

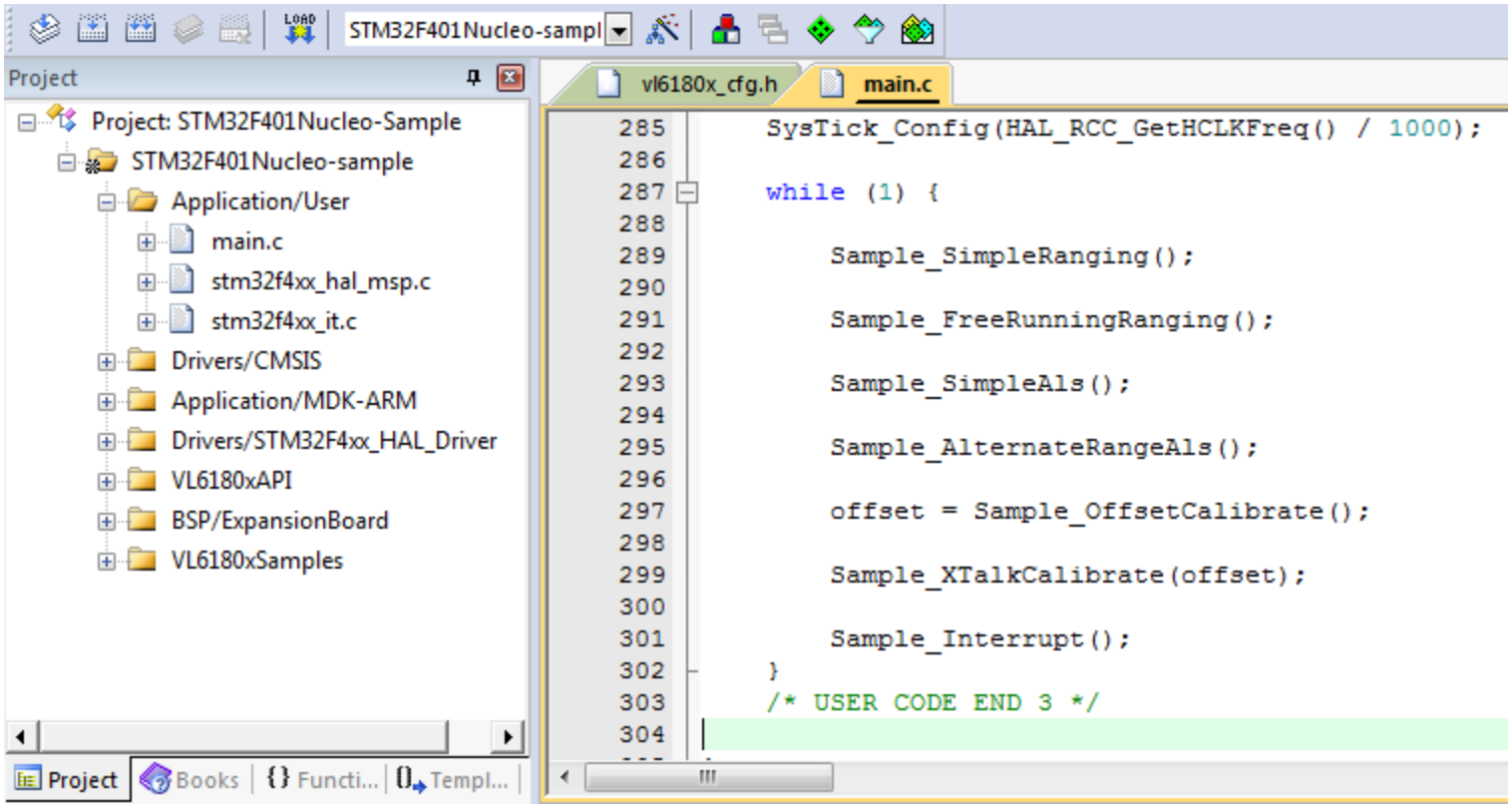
- Install latest version (5.x) of Keil uVision IDE from ARM (www.keil.com)
  - Download MDK-ARM
  - The free Lite Edition is enough (code and data size limited to 32 KB)
- Once installed, double-click on this file to open the project in Keil
  - example\Nucleo\Projects\Multi\Examples\VL6180X\CodeSamples\MDK-ARM\STM32F401RE-Nucleo\STM32F401Nucleo-Sample.uvprojx

life.augmented

Play

CodeSamples : Open Project

- The project is already configured and is ready to be compiled

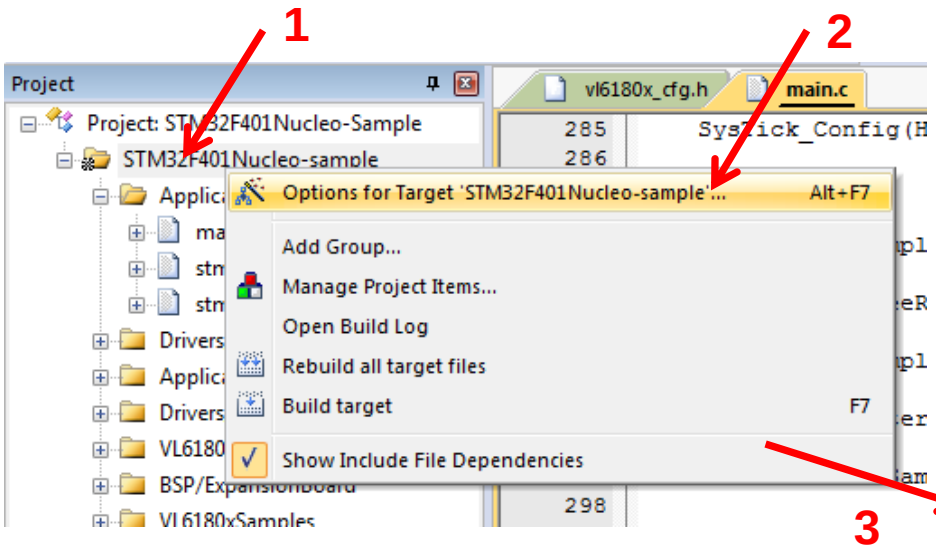


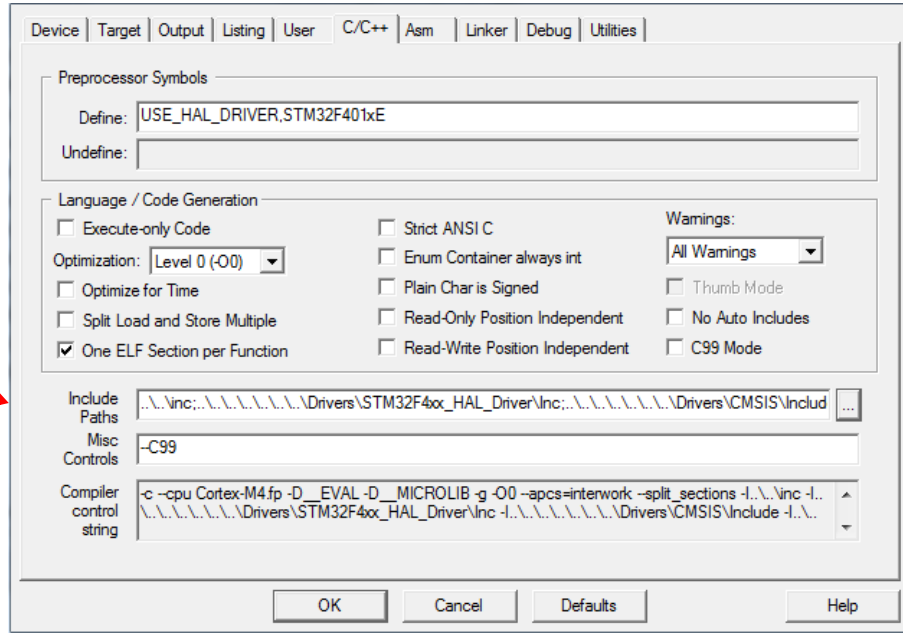
life.augmented

Play

CodeSamples : See Project config

- Project configuration can be seen as follows
  1. Right-click on the project
  2. Options for Target ‘xxx’…
  3. See project configuration window

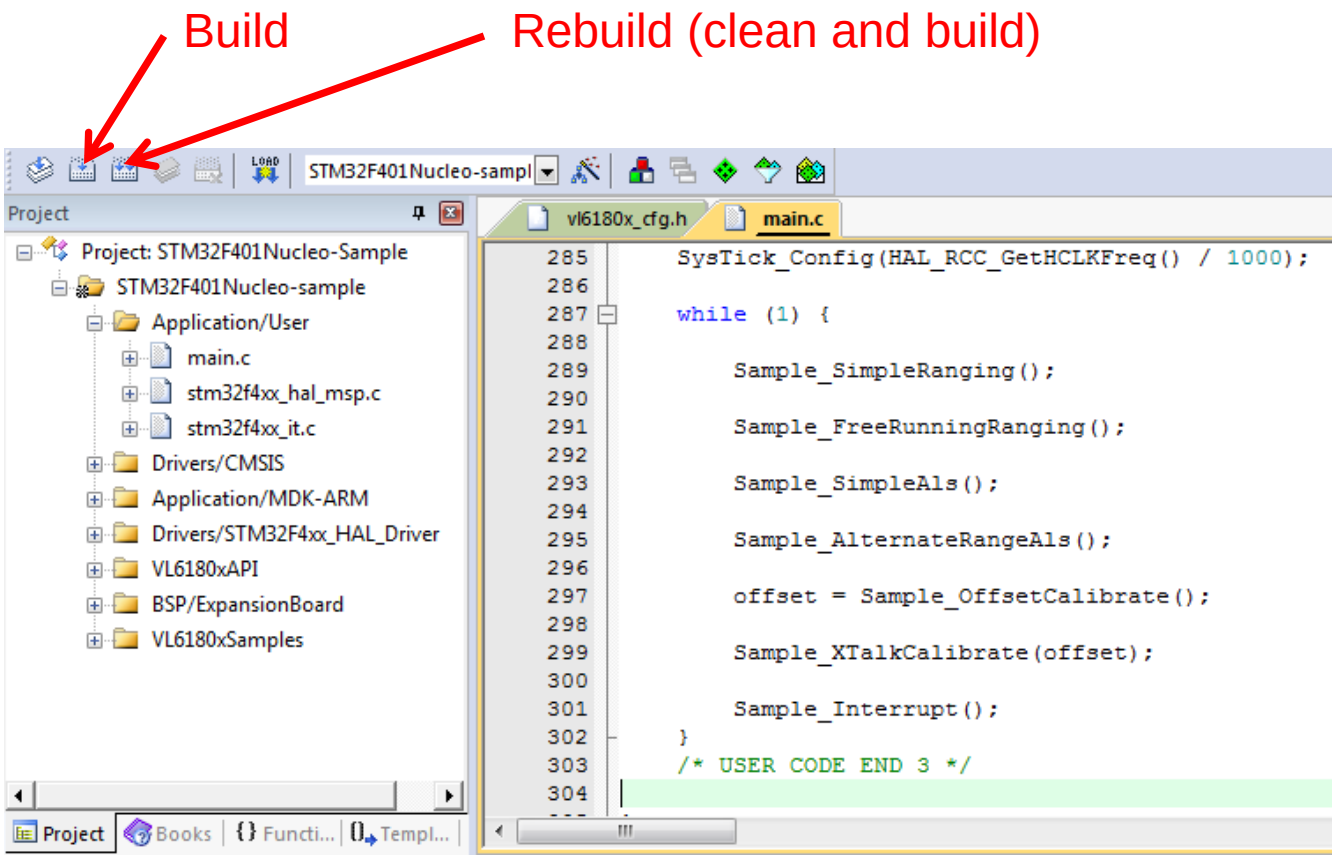




Play

CodeSamples : Compile

- Compile the project by clicking on one of the **Build** buttons

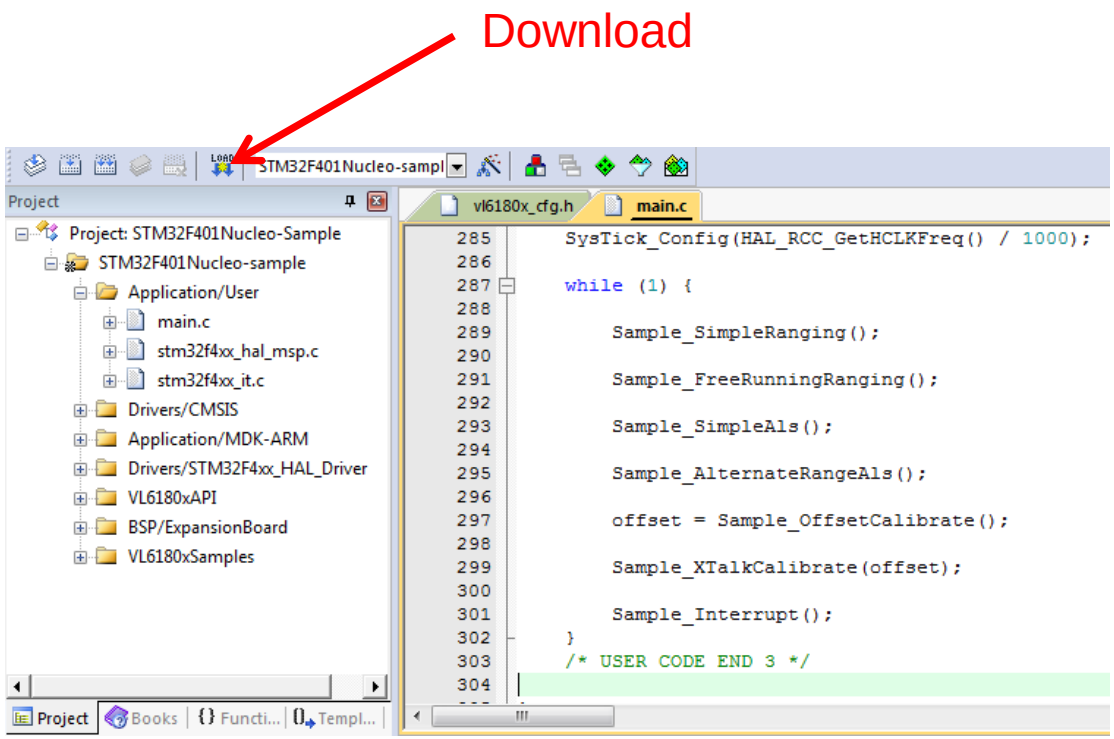


life.augmented

Play

CodeSamples : Flash the board

- Connect Nucleo board to PC through USB cable
- Flash the board with the compiled code by clicking on the **Download** button



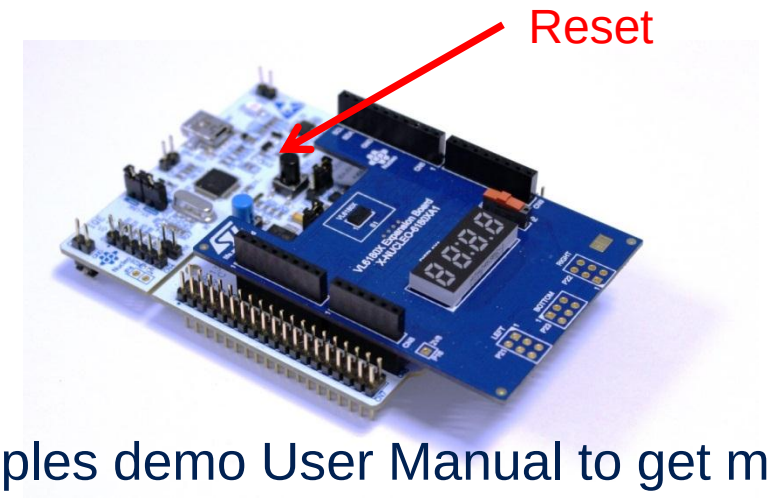
life.augmented

Play

CodeSamples : Start the demo

- Once code is downloaded, press the reset black button to start the demo

```
Build Output
Build target 'STM32F401Nucleo-sample'
"STM32F401Nucleo-VL6180X Configuration\STM32F401Nucleo-VL6180X Configuration.axf" - 0 Error(s), 0 Warning(s).
Load "STM32F401Nucleo-VL6180X Configuration\\STM32F401Nucleo-VL6180X Configuration.axf"
Erase Done.
Programming Done.
Verify OK.
```



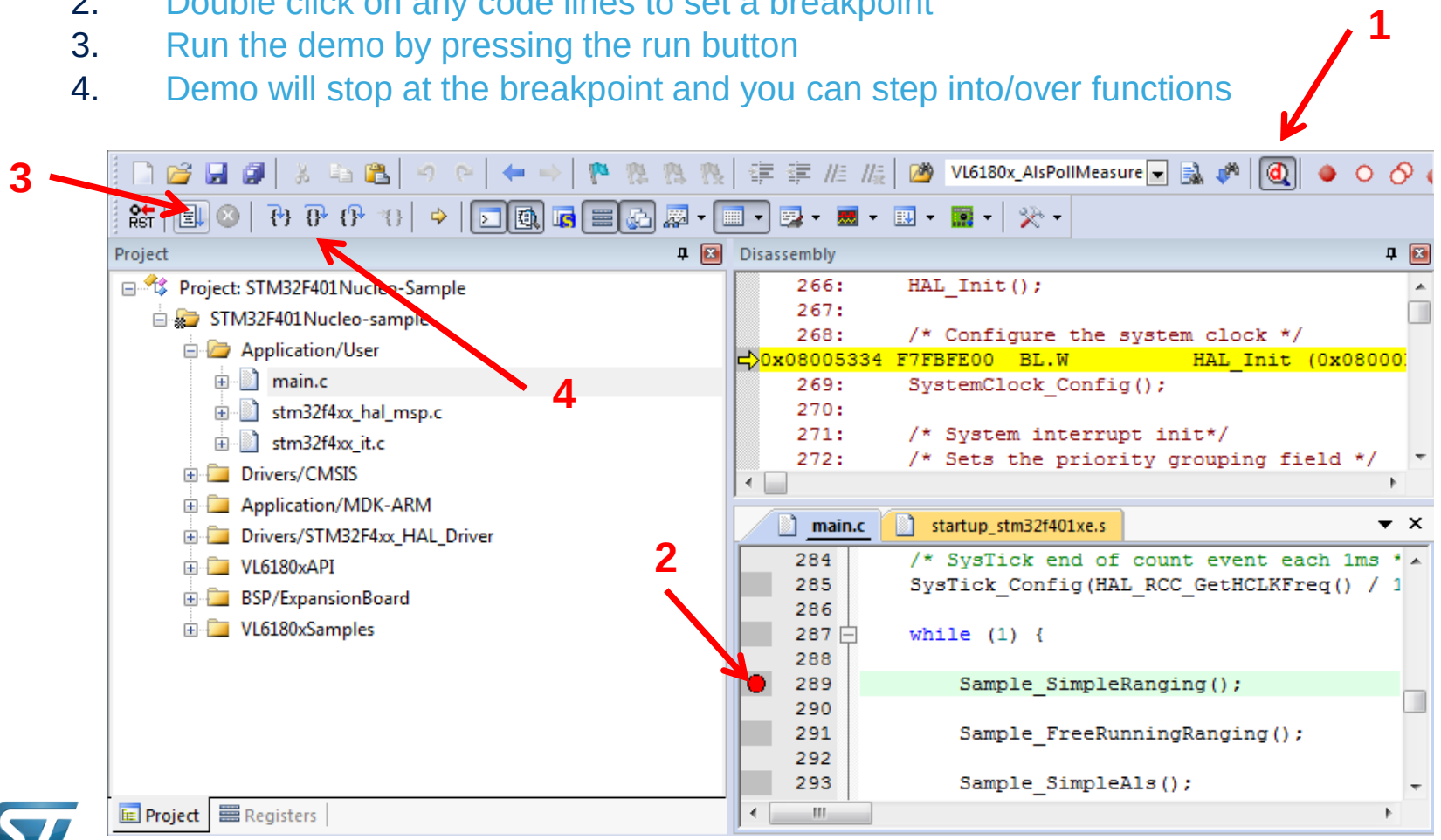
- Read CodeSamples demo User Manual to get more details on various modes

life.augmented

Play

CodeSamples : Debug the demo

- Alternatively, run the demo with the debugger
  1. Press Start/Stop debug button
  2. Double click on any code lines to set a breakpoint
  3. Run the demo by pressing the run button
  4. Demo will stop at the breakpoint and you can step into/over functions

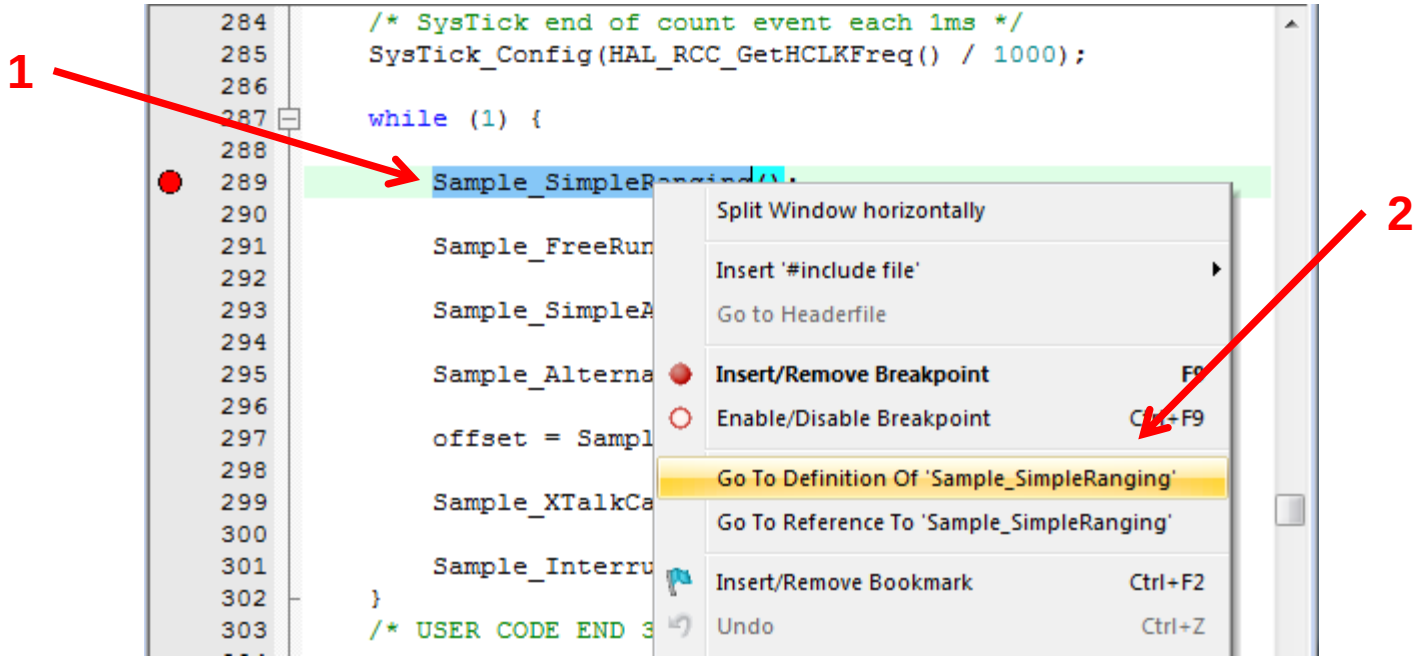


life.augmented

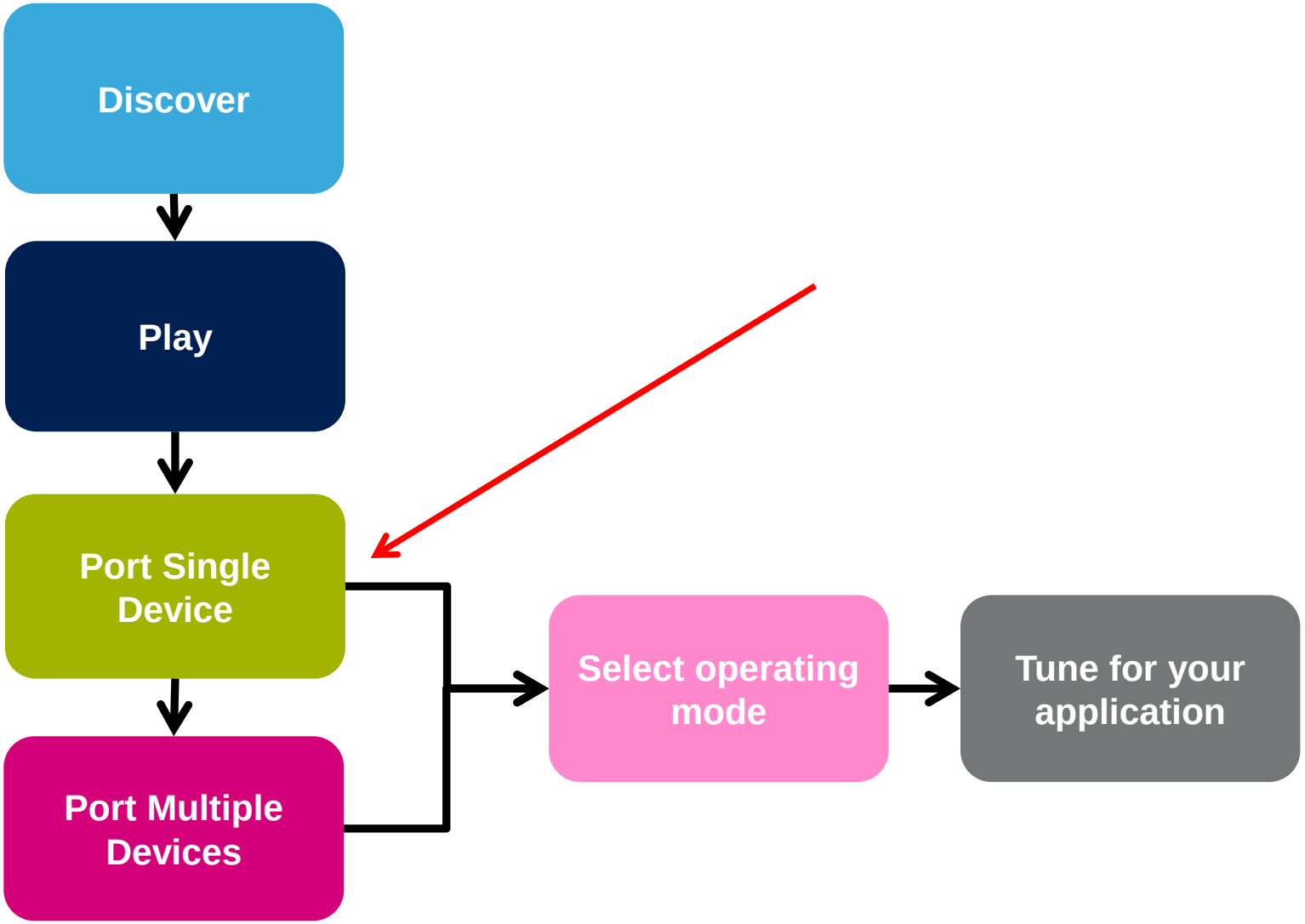
Play

CodeSamples : Analyze the code

- Using Keil (or any other IDE) is great to analyze the code and jump from functions to functions
  1. Select a function with the mouse
  2. Right-Click and select Go To Definition Of…
  3. This will open the function implementation



API Integration Procedure



life.augmented

Port API

- Following slides describe step-by-step the procedure to port the API code onto customer platform

- Pre-requisites/Assumptions
  - Customer CPU supports standard C code compiling (C99)
  - Porting is done by a SW engineer familiar with targeted CPU, compiler and toolchain
  - I2C low-level C driver (read/write) is already available and validated for the targeted paltform
  - A dummy ranging project is already created by customer and is able to do simple I2C read/write access through the I2C low-level driver

- Purpose is to upgrade the customer ranging project so that it can run a basic ranging operation (SimpleRaning sample code) with a single VL6180 device connected on I2C bus of the CPU

```
void Sample_SimpleRanging(void) {
    VL6180xDev_t myDev;
    VL6180x_RangeData_t Range;

    MyDev_Init(myDev);           // your code init device variable
    MyDev_SetChipEnable(myDev);  // your code assert chip enable
    MyDev_uSleep(1000);          // your code sleep at least 1 msec
    VL6180x_InitData(myDev);
    VL6180x_Prepare(myDev);
    do {
        VL6180x_RangePollMeasurement(myDev, &Range);
        if (Range.errorStatus == 0 )
            MyDev_ShowRange(myDev, Range.range_mm,0); // your code display range in mm
        else
            MyDev_ShowErr(myDev, Range.errorStatus); // your code display error code
    } while (!MyDev_UserSayStop(myDev)); // your code to stop looping
}
```

Port Single Device

Port API : Step 1

- Unzip VL6180_API_x.x.x_Extended_Range.zip
- Copy following files into the customer ranging project (most probably in some Include and Src directories)
  - API core (must never be modified by customer)
    - core/inc/vl6180x_api.h
    - core/inc/vl6180x_def.h
    - Core/src/vl6180x_api.c
  - API static configuration (should not be modified by customer)
    - config/extended_range/vl6180x_cfg.h
  - API platform-dependent files (must be modified by customer)
    - platform/template/vl6180x_platform.h
    - platform/template/vl6180x_types.h
    - platform/cci-i2c/vl6180x_i2c.h and platform/cci-i2c/vl6180x_i2c.c
- Modify the customer ranging project so that new files are compiled when project is built
- Add this line in main.c file to include API definitions
  - #include “vl6180x_api.h”
- Compile the customer ranging project
  - There will be some compiling errors
  - This is expected and fixing these errors is part of the API porting flow
  - See next slides…

Port Single Device

Port API : Step 2

- vl6180x_platform.h
  - Some includes like <unistd.h> or <pthread.h> could not be found in the targeted platform. If this is the case, remove them (needed only for advance usage of the API)
  - By default, API is configured/optimized to handle a single VL6180 device on the bus
    - #define VL6180x_SINGLE_DEVICE_DRIVER 1
  - **VL6180xDev_t** is a generic device type which does the link between API and platform abstraction layer. All API functions take as input a variable of this type which is passed from functions to functions down to customer I2C low-level driver
    - By default, **VL6180xDev_t** is defined as **uint8_t** and is meant to host the device I2C address
  - API polling functions (such as VL6180x_RangePollMeasurement()) rely on the VL6180x_PollDelay() function defined in vl6180x_platform.h. Depending on customer platform constraints (multi-thread, RTOS, …), this function/macro can be adapted by customer
    - By default, VL6180x_PollDelay() does nothing
  - Most of the API functions have a basic logging mechanism that can be enabled for debugging purpose (function entry/exit). This logging mechanism is implemented through several macros that should be defined in vl6180x_platform.h
    - By default VL6180X_LOG_ENABLE macro is set to 0, meaning logging is disabled so no need to concentrate on the logging macros now
  - [New in API 3.1.0] Two macros called VL6180x_HAVE_MULTI_READ and VL6180x_CACHED_REG are defined and set to 1 by default. This allows to improve speed performance of post-processing operations made in the API by using multi read I2C transfers

Port API : Step 3

- vl6180x_types.h
  - This file holds basic type definitions that may require modifications
  - By default it relies on
    - <stdint.h> for signed and unsigned 8/16/32 bits types
    - <stddef.h> for NULL definition
  - If the 2 above files are provided by the compiler, it is enough to include them, otherwise, customer must define the types for his platform
    - #include <linux/types.h> for instance for Linux

- At this stage, all compiling errors should be fixed. Let's focus now on remaining linking errors
  - Undefined symbol VL6180x_I2CRead (referred from vl6180x_i2c.o)
  - Undefined symbol VL6180x_I2CWrite (referred from vl6180x_i2c.o).

Port Single Device

Port API : Step 4

- All API functions rely on a set of 8 read/write functions (CCI-like interface)
  - VL6180x_WrByte()
  - VL6180x_WrWord()
  - VL6180x_WrDWord()
  - VL6180x_UpdateByte()
  - VL6180x_RdByte()
  - VL6180x_RdWord()
  - VL6180x_RdDWord()
  - VL6180x_RdMulti() only if VL6180x_HAVE_MULTI_READ is set to 1 in vl6180x_platform.h [New in API 3.1.0]
- In case customer I2C driver is CCI compliant, customer can simply call its CCI functions from the 7 functions above
- In case customer I2C driver exposes low level/raw I2C commands, a CCI to I2C translation layer is proposed by ST in the vl6180x_i2c.c file. This implementation relies on two I2C raw access functions that must be implemented by customer (to make the link with customer I2C driver)
  - VL6180x_I2CRead(VL6180xDev_t dev, uint8_t* buff, uint8_t len)
  - VL6180x_I2CWrite(VL6180xDev_t dev, uint8_t* buff, uint8_t len)
- If the proposed implementation of the 8 above functions is not adapted to customer I2C driver, customer can adapt them
- At this stage, all compiling and linking errors should be fixed : API is ported !
  - Time to start using it (see next slide)

Port Single Device

Port API : Step 5

- Insert this code (from example/code_samples/vl6180x_simple_ranging.c) in the project and call the Sample_SimpleRanging() function from main.c file

```
void Sample_SimpleRanging(void) {
    VL6180xDev_t myDev;
    VL6180x_RangeData_t Range;

    MyDev_Init(myDev);           // your code init device variable
    MyDev_SetChipEnable(myDev);  // your code assert chip enable
    MyDev_uSleep(1000);          // your code sleep at least 1 msec
    VL6180x_InitData(myDev);
    VL6180x_Prepare(myDev);
    do {
        VL6180x_RangePollMeasurement(myDev, &Range);
        if (Range.errorStatus == 0 )
            MyDev_ShowRange(myDev, Range.range_mm,0); // your code display range in mm
        else
            MyDev_ShowErr(myDev, Range.errorStatus); // your code display error code
    } while (!MyDev_UserSayStop(myDev)); // your code to stop looping
}
```

- Implement following functions (see examples for Nucleo)
  - MyDev_Init()
  - MyDev_SetChipEnable()
  - MyDev_uSleep()
  - MyDev_ShowRange()
  - MyDev_ShowErr()

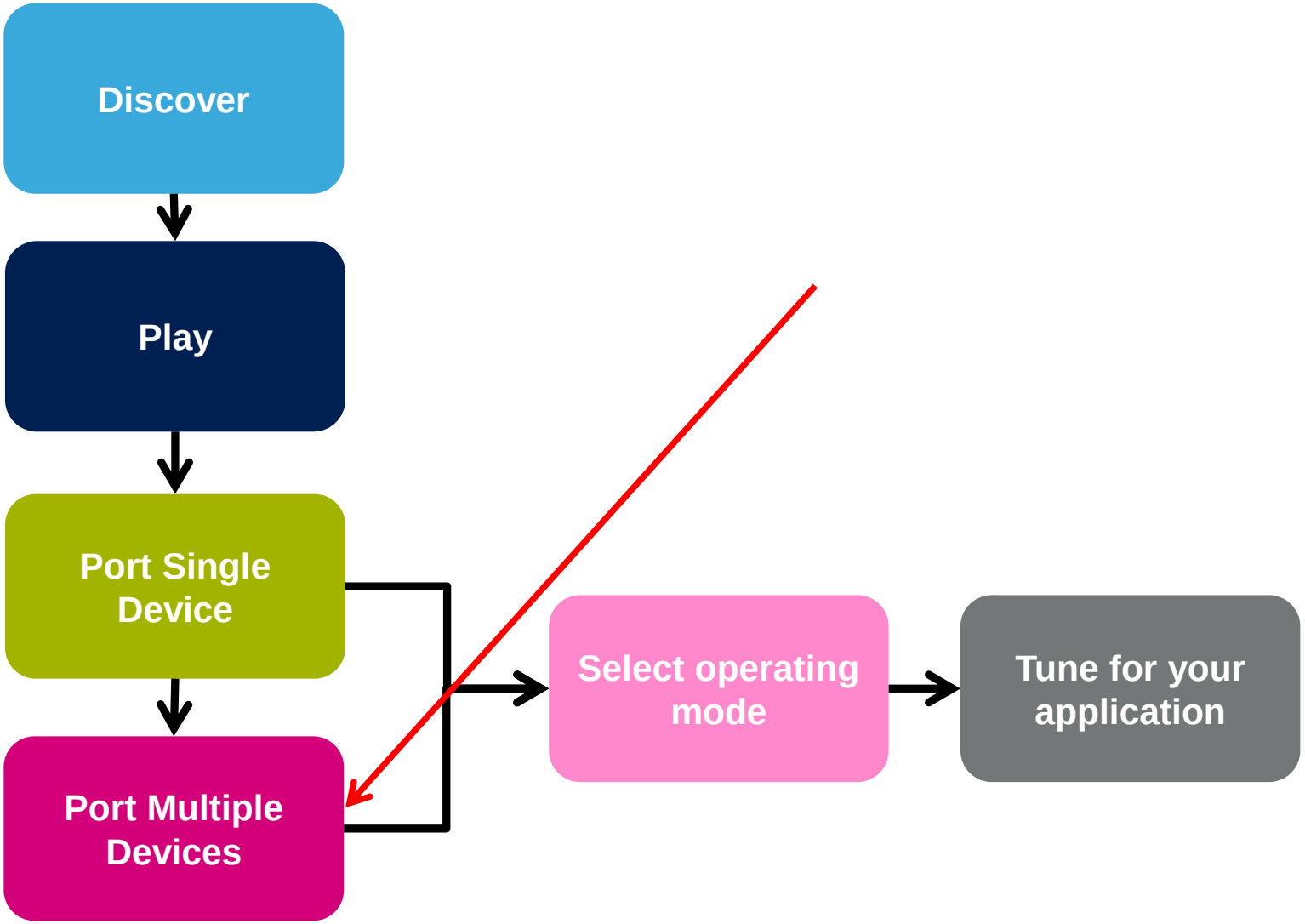
```
#define MyDev_Init(dev)  (dev=0x52)
```

```
static inline void MyDev_SetChipEnable(VL6180xDev_t dev) {
        XNUCLEO6180XA1_Reset(0);
        XNUCLEO6180XA1_WaitMilliSec(5);
        XNUCLEO6180XA1_Reset(1);
        XNUCLEO6180XA1_WaitMilliSec(5);
}
```

```
#define MyDev_uSleep(x) XNUCLEO6180XA1_WaitMilliSec((x+999)/1000)
```

life.augmented

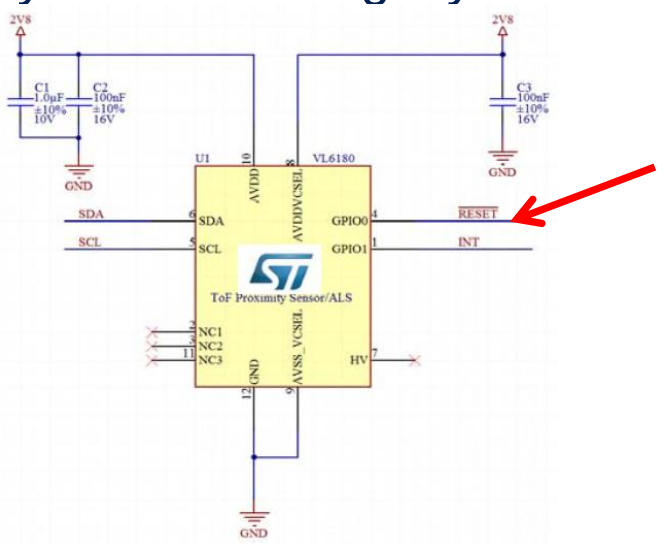
API Integration Procedure



life.augmented

How to control several devices ?

- Assumptions
  - API has already been ported following the 5 steps described above (single device)
  - Several devices are connected to the same I2C bus of host CPU
  - Each VL6180 device can be put out of reset separately by the CPU
- The procedure consists in initializing each device one by one and for each to change its I2C address so that it can be accessed by the CPU with a dedicated address
- Before applying this procedure, API platform layer must be slightly adapted for multi-device support
  - See next slides…



Adapt API platform layer for multi-devices

- In vl6180x_platform.h API file
  - Set VL6180x_SINGLE_DEVICE_DRIVER macro to 0 so that API implementation will be automatically adapted to a multi-device context
    - #define VL6180x_SINGLE_DEVICE_DRIVER 0
  - Define **VL6180xDev_t** type as a structure pointer holding any data required for multi-device management. A mandatory field is an instance of VL6180xDevData containing ST API private data
    - Here is an example

```
struct MyVL6180Dev_t {
    struct VL6180xDevData_t Data; // Mandatory (API private data)
    uint8_t I2cAddr;              // Recommended : I2C address
    uint8_t DevID;                // Optional : Device ID
    uint8_t DevPresent;           // Optional : is device present ?
};
typedef struct MyVL6180Dev_t *VL6180xDev_t;
```

Multi device init sequence

- Pseudo-code shown here is extracted from RangingWithSatellites Nucleo main.c file
- First, reset of devices:

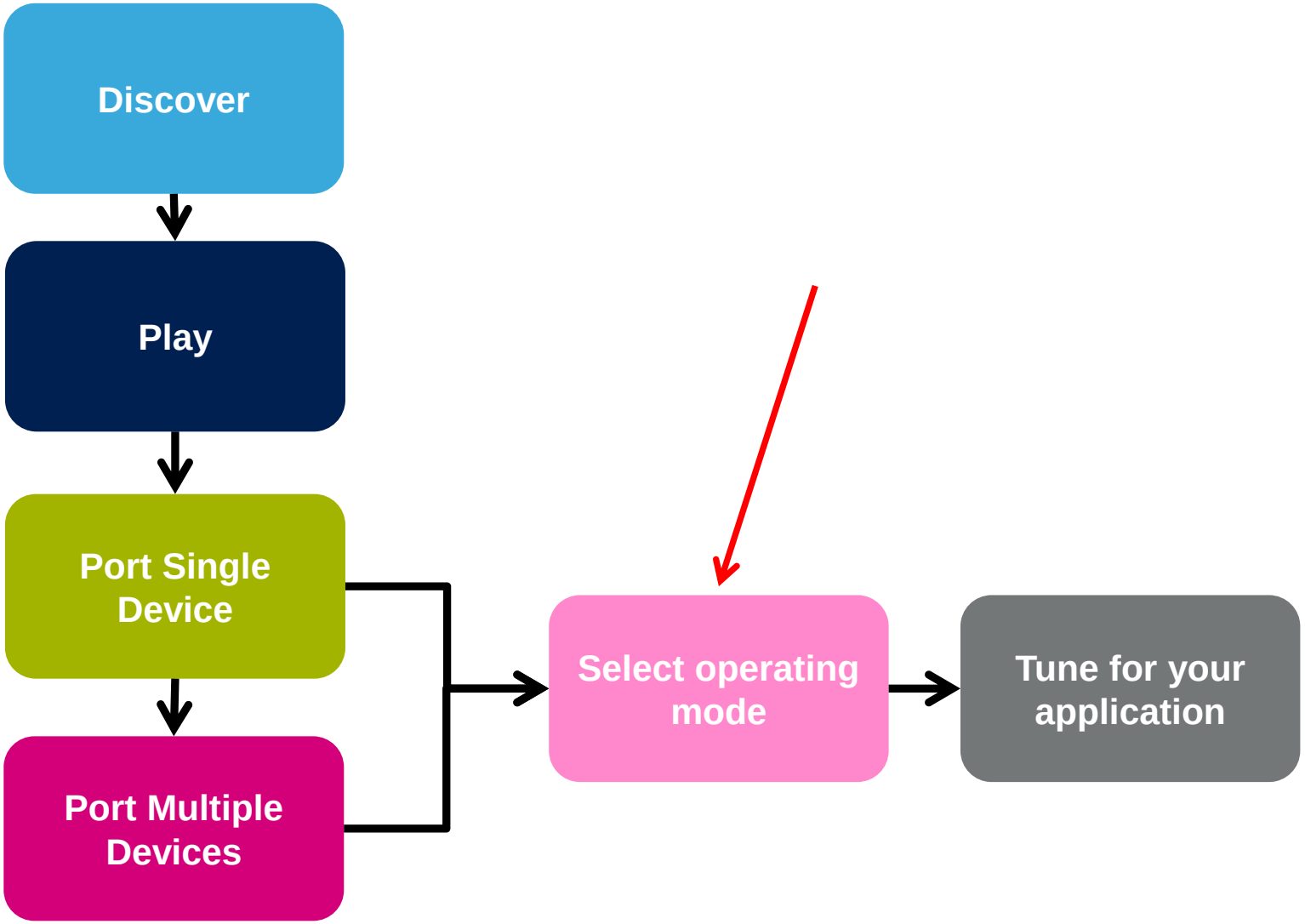
```
#define MAX_DEV 4
/* One instance of MyVL6180Dev_t per device */
struct MyVL6180Dev_t BoardDevs[4];

int status;
int i;
VL6180x_RangeData_t Range[MAX_DEV]; /* One instance of RangeData structure per device */

/* Put all devices under reset */
for (i = 0; i < MAX_DEV; i++) {
    /* put all under reset */
    MyDev_Reset(&BoardDevs[i]);
}
```

Multi device init sequence

```
/* Detect presence and initialize devices i2c address  */
/* Set device i2c address for dev[i] = 0x52+(i+1)*2 */
for (i=0;i<n_dev;i++) {
    int FinalI2cAddr;
    uint8_t id;

    /* Unreset device that wake up at default i2c addres 0x52 */
    MyDev_UnReset(&BoardDevs[i]);
    MyDev_uSleep(2000); /* at least 400usec before accessing the device */
    BoardDevs[i].I2cAddr = 0x52;

    /* To detect device presence try to read its dev id */
    status = VL6180x_RdByte(&BoardDevs[i], IDENTIFICATION_MODEL_ID, &id);
    if (status) {
        /* This device is not present => skip init */
        BoardDevs[i].Present = 0;
        continue;
    }

    /* device present only */
    BoardDevs[i].Present = 1;

    /* Init device before changing its I2C address */
    status = VL6180x_InitData(&BoardDevs[i]);

    /* Change I2C address */
    FinalI2cAddr = 0x52 + ((i+1) * 2);
    if (FinalI2cAddr != 0x52) {
        status = VL6180x_SetI2CAddress(&BoardDevs[i], FinalI2cAddr);
        if( status ){
            HandleError("VL6180x_SetI2CAddress fail");
        }
        BoardDevs[i].I2cAddr = FinalI2cAddr; /* Assign new address to the device structure */
    }

    /* Call any API function now for the selecting device */
    status= VL6180x_Prepare(&BoardDevs[i]);
}
```

Optional

API Integration Procedure



life.augmented

Select operating mode

Select best VL6180 operating mode

- VL6180 device can operate in 2 different modes
  - Single shot measurement
    - RangeReady known from register polling
  - Continuous measurement
    - RangeReady known from interrupt generated by the device
    - Warning : WAF not functional in this mode !
- Based on this 2 modes, VL6180 API exposes 3 different operating modes
  - Polling
  - Interrupt
  - Asynchronous

life.augmented

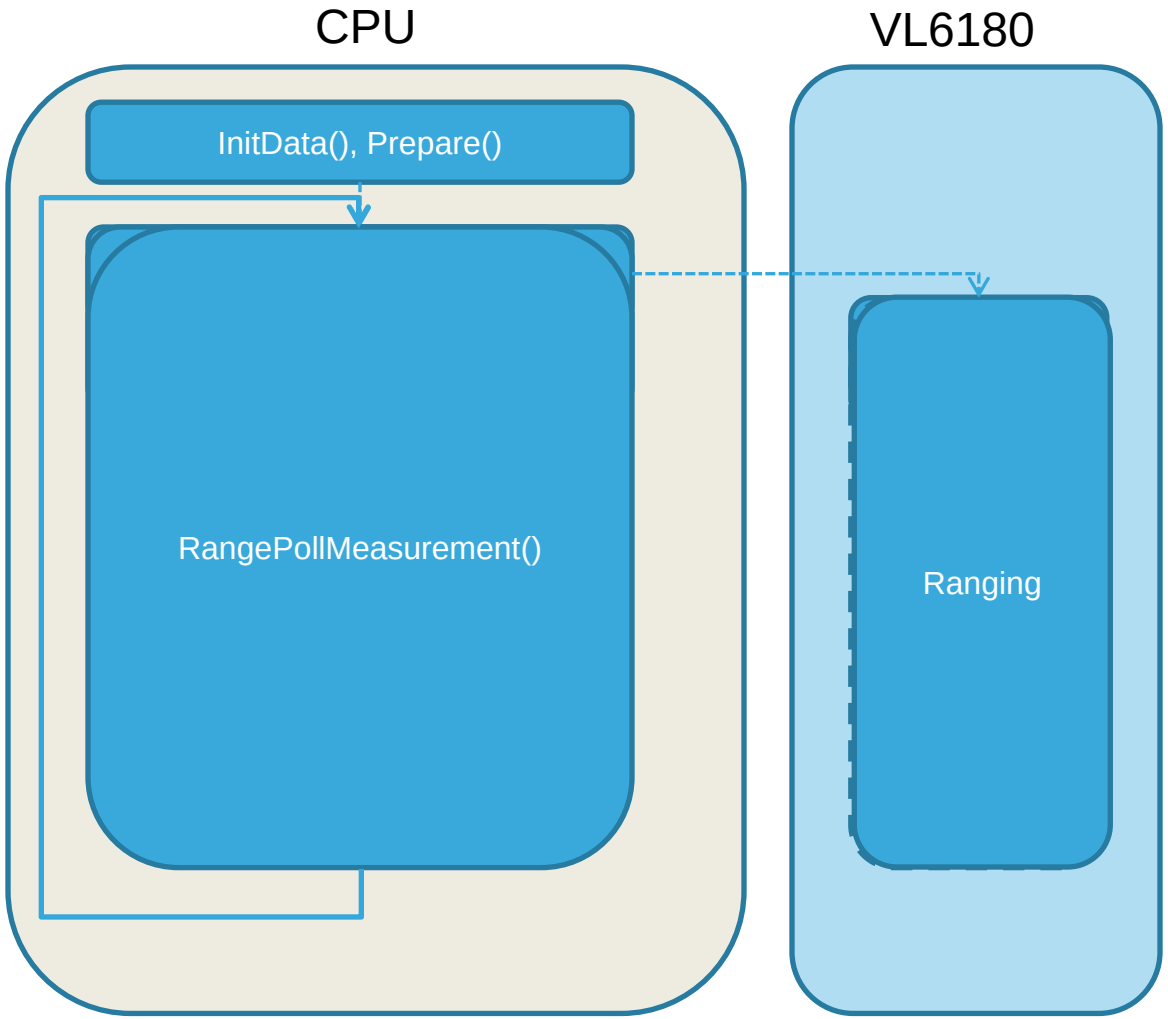
Select operating mode

VL6180 operating modes

API operating mode	Description	API functions	VL6180 mode	Comments
Polling	Host call a blocking API function that requests single shot measurements and waits for the result	VL6180x_RangePollMeasurement	Single Shot	**Recommended for first API porting or debug** : simple but CPU is blocked during ranging
Interrupt	Host programs the device in continuous mode and ranging results are retrieved from interrupts	VL6180x_RangeSetInterMeasPeriod VL6180x_SetupGPIO1 VL6180x_RangeConfigInterrupt VL6180x_RangeStartContinuousMode VL6180x_RangeGetMeasurement VL6180x_ClearAllInterrupt	Continuous	**Recommended for User Detection applications** where CPU is interrupted by VL6180 so can be asleep when no target is detected (power saving)
Asynchronous / FreeRunning	Host requests a single shot measurement and regularly check if result is ready or not	VL6180x_RangeStartSingleShot VL6180x_RangeGetMeasurementIfReady	Single Shot	**Recommended for AF-Assist applications, Android OS-based system** where CPU is synchronized by EOF/SOF from camera or by a timer so that top application controls measurement periods

Select operating mode

Polling mode

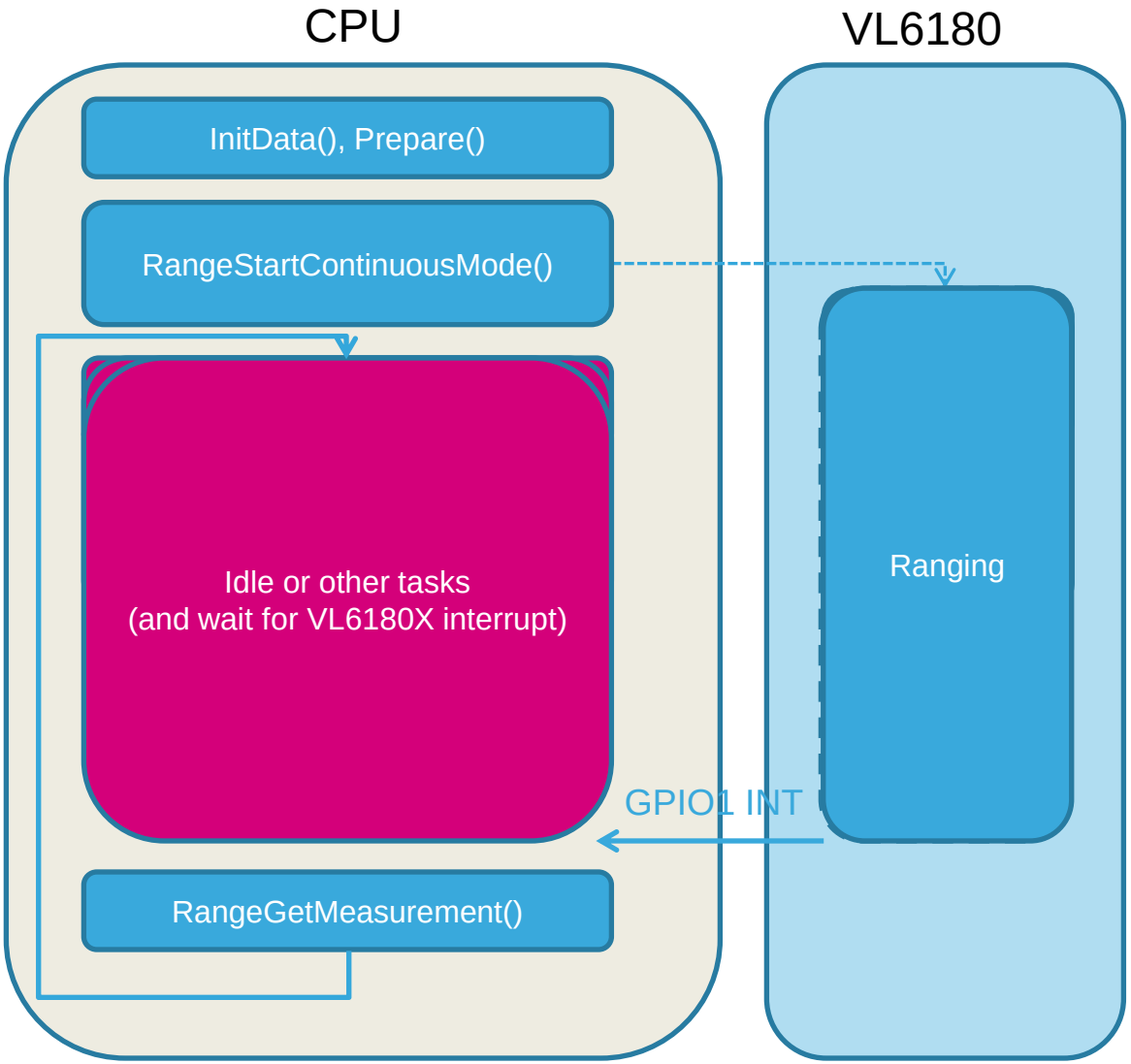


☺ Easy to use
☹ CPU busy

Display in Slide Show mode

Select operating mode

Interrupt mode

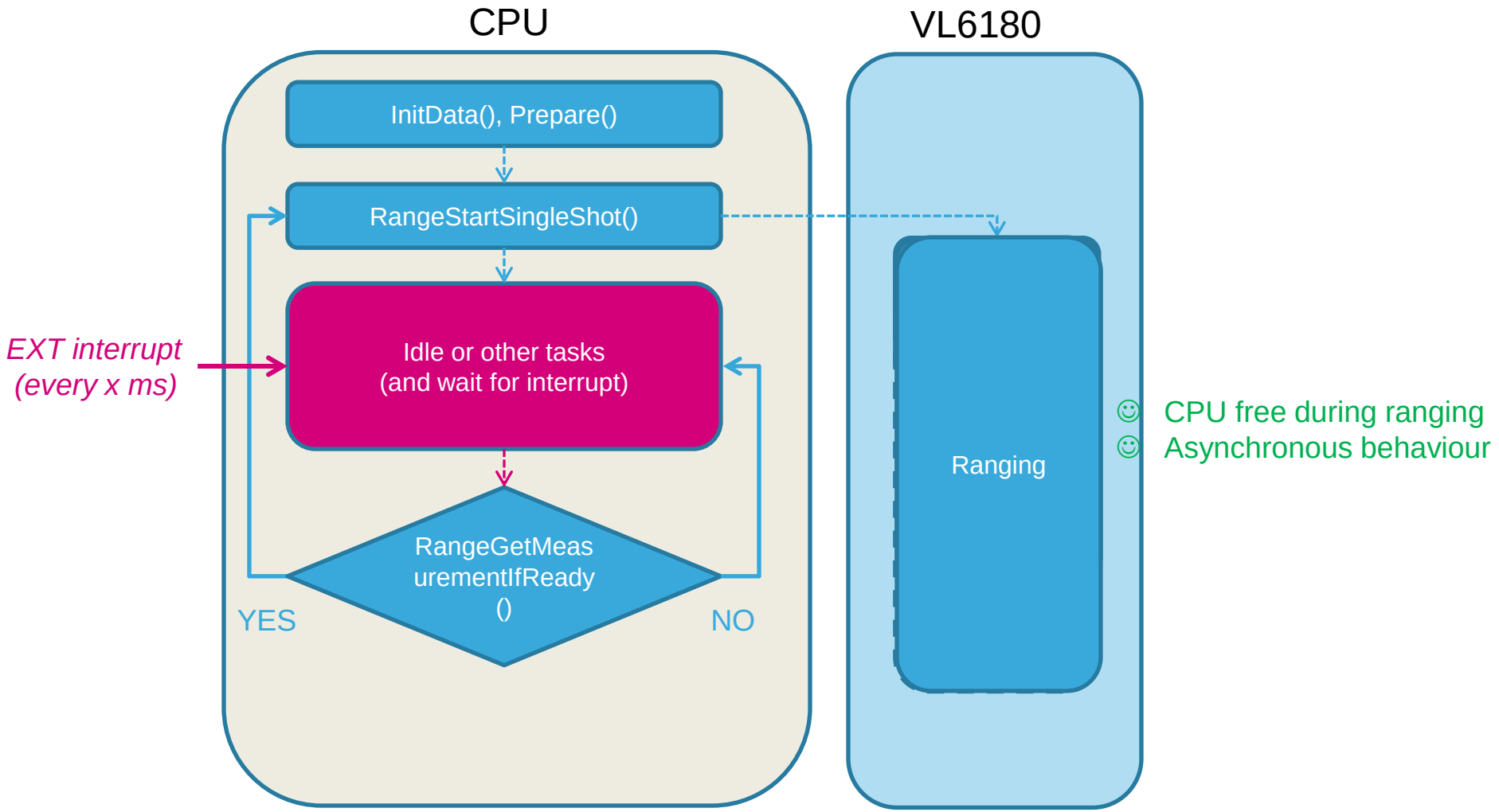


☺ CPU free during ranging
☹ Interrupt to be managed

Display in Slide Show mode

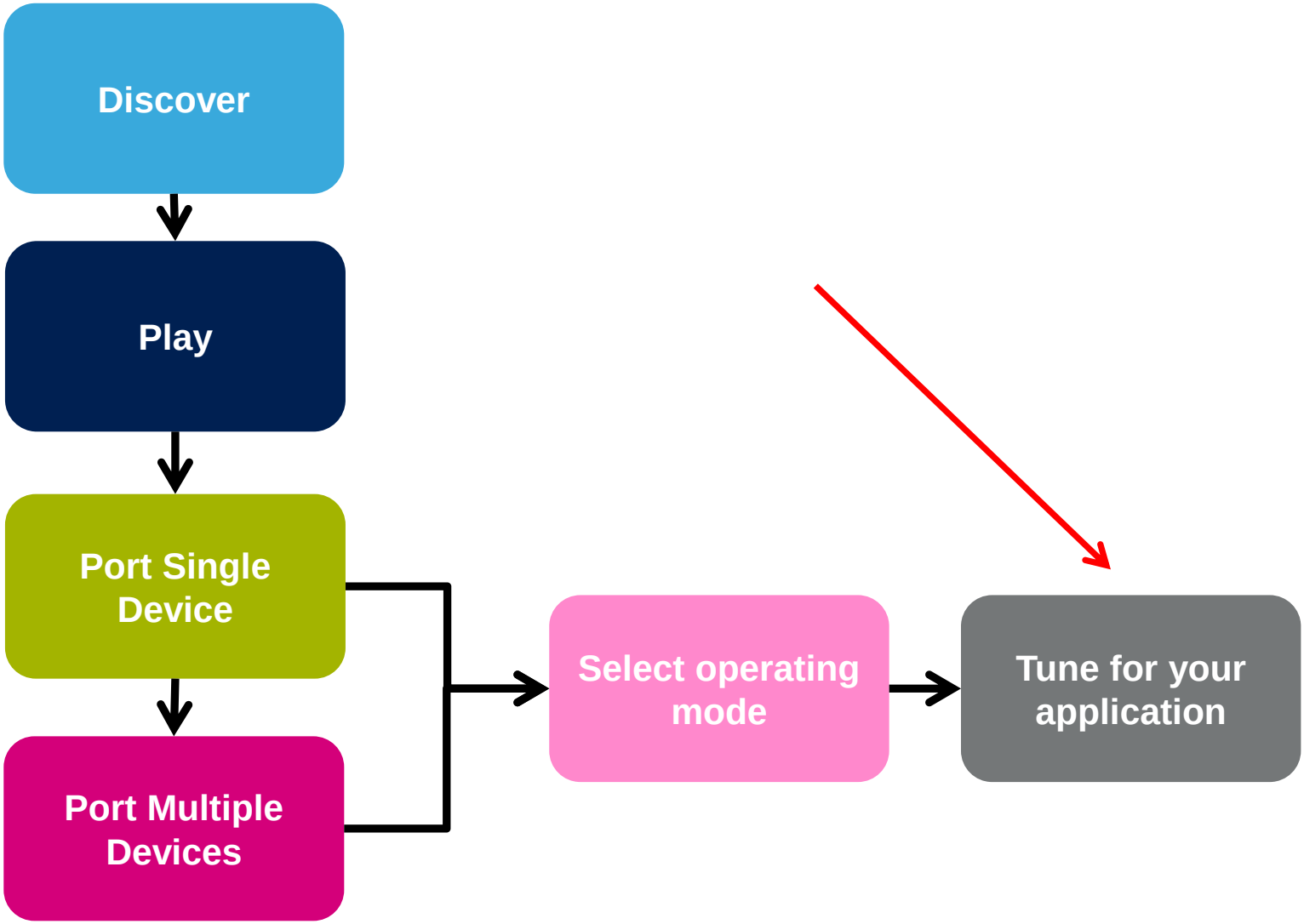
Select operating mode

Asynchronous mode



Display in Slide Show mode

API Integration Procedure



life.augmented

Make use of API error codes

- List of API Ranging measurement functions returning errorStatus in the Range structure (pRangeData):
  - **VL6180x_RangePollMeasurement** (VL6180xDev_t dev, VL6180x_RangeData_t *pRangeData)
  - **VL6180x_RangeGetMeasurementIfReady** (VL6180xDev_t dev, VL6180x_RangeData_t *pRangeData)
  - **VL6180x_RangeGetMeasurement** (VL6180xDev_t dev, VL6180x_RangeData_t *pRangeData)

- When calling one of the 3 “Ranging measurment” API functions, application must check the Range.errorStatus before using the returned distance
  - If Range.errorStatus is null, returned distance is valid
  - Otherwise, application can use the Range.errorStatus value to known the reason for not getting a valid distance and try to “guess” where the target could be

- It is strongly recommended that application calls the VL6180x_RangeGetMeasurementIfReady() API function to get ranging data as this is the function which is able to return all supported errorStatus…

```
/* kick off the first measurement */
VL6180x_RangeStartSingleShot(myDev);
do {

    // TODO add your code anything in a loop way
    VL6180x_PollDelay(dev); // simply  run default API poll delay that handle display in demo
    // check for range measure availability
    status= VL6180x_RangeGetMeasurementIfReady(myDev, &Range);
    if( status == 0 ){
        // Application must check Range.errorStatus before accessing the other data
        //    If Range.errorStatus is DataNotReady, application knows that it has to wait a bit before getting a new data
        //    If Range.errorStatus is 0, application knows it is a valid distance
        //    If Range.errorStatus is not 0, application knows that reported distance is invalid so may take some decisions depending on the errorStatus
        if (Range.errorStatus == DataNotReady)
            continue;

        if (Range.errorStatus == 0)
            MyDev_ShowRange(myDev, Range.range_mm,0); // your code display range in mm
        else
            MyDev_ShowErr(myDev, Range.errorStatus); // your code display error code
        /* re-arm next measurement */
        VL6180x_RangeStartSingleShot(myDev);
    }
    else{
        // it is an critical error
        HandleError("critical error on VL6180x_RangeCheckAndGetMeasurement");
    }

} while (!MyDev_UserSayStop(myDev)); // your code to stop looping
```

Tune for your application

API errorStatus codes

Valid distance

Enumerator	
NoError	0 0b0000 NoError
VCSEL_Continuity_Test	1 0b0001 VCSEL_Continuity_Test
VCSEL_Watchdog_Test	2 0b0010 VCSEL_Watchdog_Test
VCSEL_Watchdog	3 0b0011 VCSEL_Watchdog
PLL1_Lock	4 0b0100 PLL1_Lock
PLL2_Lock	5 0b0101 PLL2_Lock
Early_Convergence_Estimate	6 0b0110 Early_Convergence_Estimate
Max_Convergence	7 0b0111 Max_Convergence
No_Target_Ignore	8 0b1000 No_Target_Ignore
Not_used_9	9 0b1001 Not_used
Not_used_10	10 0b1010 Not_used_
Max_Signal_To_Noise_Ratio	11 0b1011 Max_Signal_To_Noise_Ratio
Raw_Ranging_Algo_Underflow	12 0b1100 Raw_Ranging_Algo_Underflow
Raw_Ranging_Algo_Overflow	13 0b1101 Raw_Ranging_Algo_Overflow
Ranging_Algo_Underflow	14 0b1110 Ranging_Algo_Underflow
Ranging_Algo_Overflow	15 0b1111 Ranging_Algo_Overflow
RangingFiltered	16 0b10000 filtered by post processing
RangingFilteringOnGoing	17 0b10001 ranging filter on going (need few measurements)
DataNotReady	18 0b10011 New data sample not ready

Shall not occur during ranging, can happen only after boot-up if device is defective

Happens only if ECE check is enable, meaning that device detected that no target is present (ECE is off by default in extended-range)

Not valid target detected after maximum convergence time

Too much ambient light detected

Distance filtered by WAF

[Added in API 3.1.0] WAF decision on-going
[Removed in API 3.2.0]

[New in API 3.1.0] Ranging data not ready

life.augmented

Ranging API error status codes (details)

- Error 8 : No Target Ignore
  - This could happen only if RangeIgnore feature is enabled to filter some specific cases with high xTalk compensation values…
- Error 12 or 14 : Ranging underflow
  - This could happen if target is very close (0 to 10 mm) and offset is not correctly set-up, leading to a small negative value (user putting finger on device)
  - This might also happens when a bright target is very far (between 1.2m and 1.5 m) but this will most probably be filtered by WAF => error 16
  - Differentiation between the 2 cases can be done by checking the return signal rate
- Error 13 or 15 : Ranging overflow
  - Theoretically, this error happens when the target is detected by the device but is placed at a high distance resulting in internal variables/registers overflow
  - This means the target in at > 55cm distance (scaling x3)
  - As a side effect, this error can also happen in case of high compensated xTalk and when no target is detected : because of high xTalk, signal rate could be higher than 0.175 Mcps and lower than xTalk : this shall never happen because signaRate=xTalk+targetSIgnalRate, but due to noise introduced by xTalk compensation, this could happen and this error could be generated

0	No Error
1	VCSEL Continuity Test
2	VCSEL Watching Test
3	VCSEL Watchdog
4	PLL1 Lock
5	PLL2 Lock
6	ECE
7	Max Convergence
8	No Target Ignore
11	Max Signal To Noise Ratio
12	Raw Ranging Algo Underflow
13	Raw Ranging Algo Overflow
14	Ranging Algo Underflow
15	Ranging Algo Overflow
16	Measure filtered (WAF)
18	Range Data Not Ready

Ranging API error codes (details)

- Error 16 : Filtered by post-processing (WAF)
  - This error happens when WAF detects a target in a wrap-around condition so instead of returning a wrong (largely under-estimated) distance, this error code is returned
  - When this error is returned, application knows that a bright target is between 60 cm and 120 cm
- Error 18 : ranging data not ready [New in API 3.1.0]
  - This error is returned by the VL6180x_RangeGetMeasurementIfReady() API function when new ranging sample data is not ready
  - Application must simply wait a bit and call again VL6180x_RangeGetMeasurementIfReady()

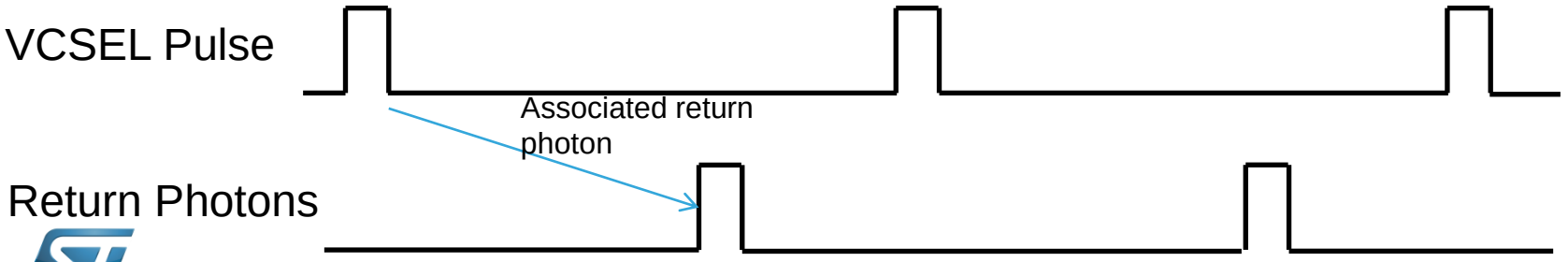
0	No Error
1	VCSEL Continuity Test
2	VCSEL Watching Test
3	VCSEL Watchdog
4	PLL1 Lock
5	PLL2 Lock
6	ECE
7	Max Convergence
8	No Target Ignore
11	Max Signal To Noise Ratio
12	Raw Ranging Algo Underflow
13	Raw Ranging Algo Overflow
14	Ranging Algo Underflow
15	Ranging Algo Overflow
16	Measure filtered (WAF)
18	Range Data Not Ready

Scaling factor

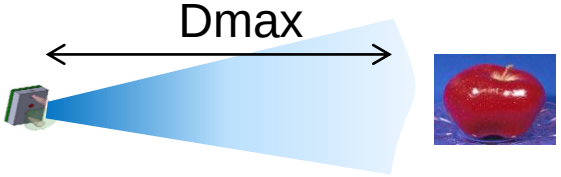
- VL6180 default scaling factor is x3 (defined in API static configuration) which allows max ranging capabilities
- Scaling factor could be decreased to x2 or even x1 for short range applications
  - A single API function allows to change the scaling factor of the device : **VL6180x_UpscaleSetScaling()**
- Offset and xTalk calibration procedures depend on the used scaling factor so API calibration functions are also provided to manage this dependency transparently for the application
  - VL6180x_SetOffsetCalibrationData()
  - VL6180x_SetXTalkCompensationRate()

Wrap-Around Filter (WAF)

- This is an effect linked to the ratio between the VCSEL pulse period and the photon return pulse
- High reflective targets (like mirrors) placed at far distance (>60cm) from VL6180 could still produce enough return signal for VL6180 to declare a valid target and meat the wrap-around condition resulting in a wrong (under-estimated) returned distance
- WAF implemented in API is able to automatically detect a target is in the WA conditions and filters it by returning an invalid distance (error status 16) instead of a wrong under-estimated one
- A dedicated error code is returned by the API
  - 16 : Measure filtered by WAF



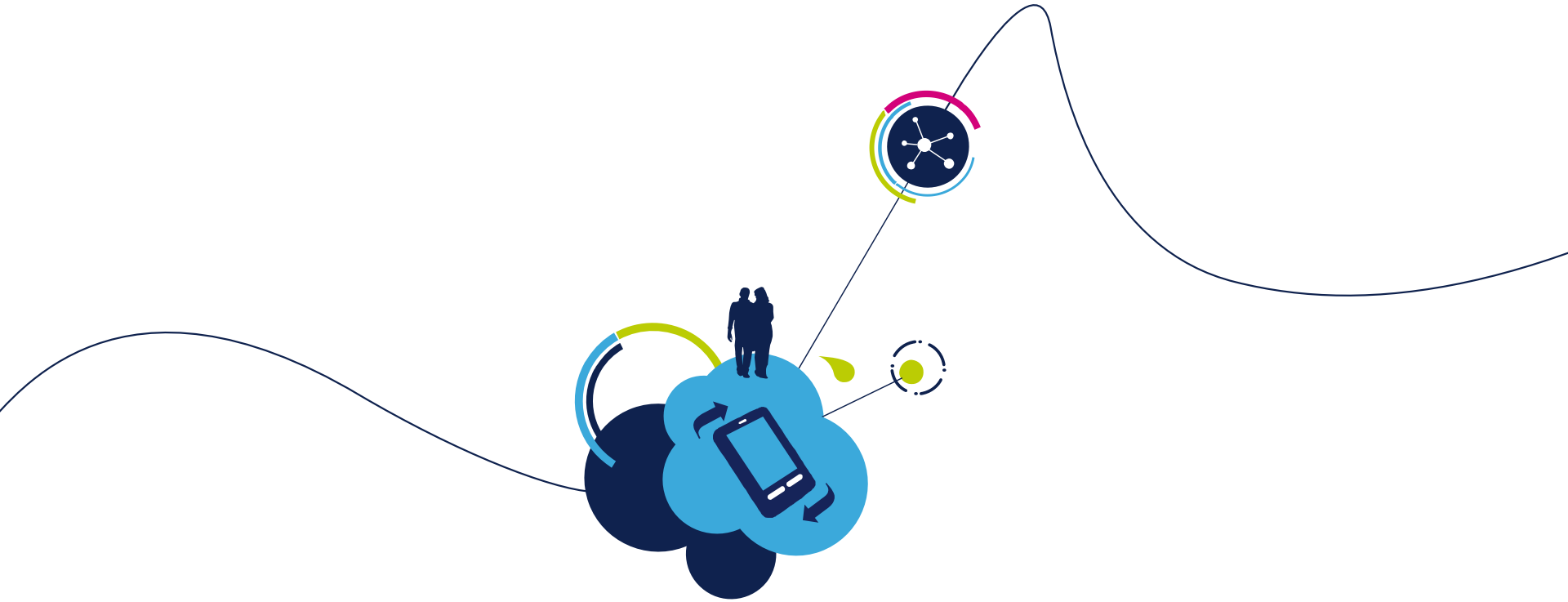
Tune for your application



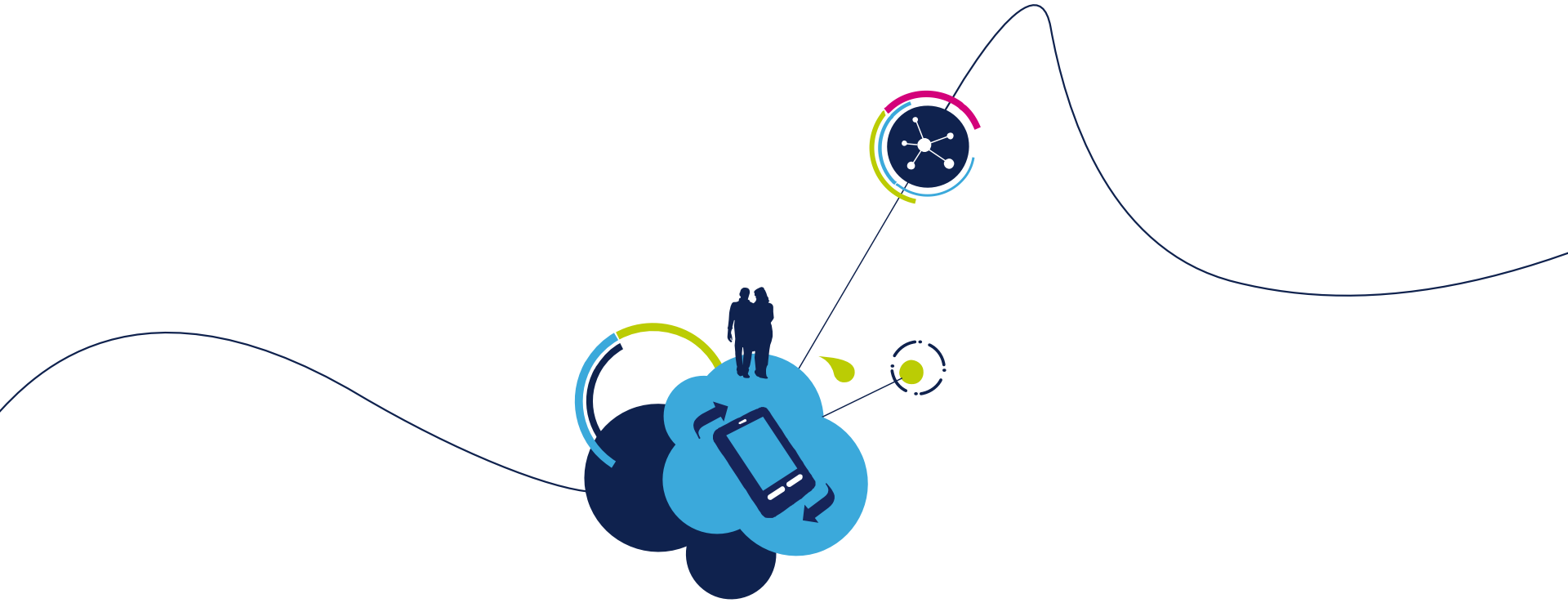
Dmax estimation

- A target placed in front of VL6180 may not be detected because it is too far for the given ambient light conditions (too high)
  - When ambient light level increases, max detection range (Dmax) decreases
- When no target is detected (no valid distance), VL6180 API is able to estimate Dmax being defined as
  - The maximum distance up to which a 17% target would have been detected with the current ambient light level
- When no target is detected by VL6180, the application can interpret the Dmax value as follows
  - No target is detected and there is **no 17% (or above) target between 0 and Dmax mm**
- This could be used by Hybrid AF algorithm to start a standard full search from Dmax (to infinity) to reduce the focus time when target is too far from VL6180
- Dmax calculation constraints & limitations
  - A correction factor is applied to slightly under-estimate the returned Dmax value: since it can be important to ensure Dmax is never over-estimated.
  - Dmax is estimated for a 17% reflectance target. This means that if the real target has a lower reflectance (black target for instance), Dmax calculated by the API could be overestimated.

life.augmented



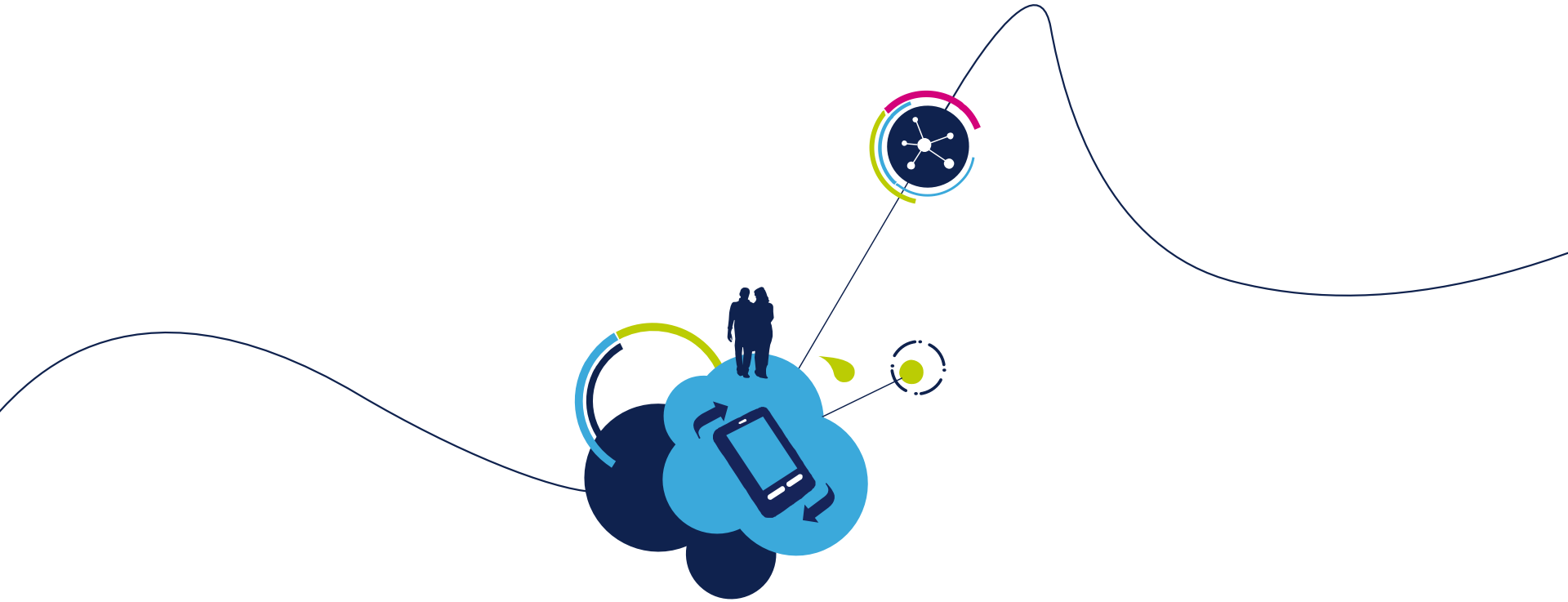
DONE ☺



From API 3.0.2 to API 3.1.0

Porting Guide

- Adapt application code to the new behavior of the VL6180x_RangeGetMeasurementIfReady() API function
  - In API 3.0.2, “Data not Ready” was returned in the function status
  - In API 3.1.0, “Data not Ready” is returned in the Range.errorStatus as 18 code
  - See API sample code : vl6180x_freeruning_ranging.c
- Optionally, add following macros in vl6180x_platform.h file to benefit from the new “Cached Register” feature based on I2C multi read (to speed-up ranging thanks to I2C multi read operations)
  - #define VL6180x_HAVE_MULTI_READ 1
  - #define VL6180x_CACHED_REG 1
- Adapt application to take benefit from new error status returned by the VL6180x_RangeGetMeasurementIfReady() function
  - 17 : WAF decision on-going
    - Application should not take drastic decision
  - 18 : Data not read



From API 3.1.0 to API 3.2.0

Porting Guide

- Adapt application in case errorStatus 17 is used
  - This errorStatus code is not produced anymore by API 3.2.0 so application can not rely on it anymore