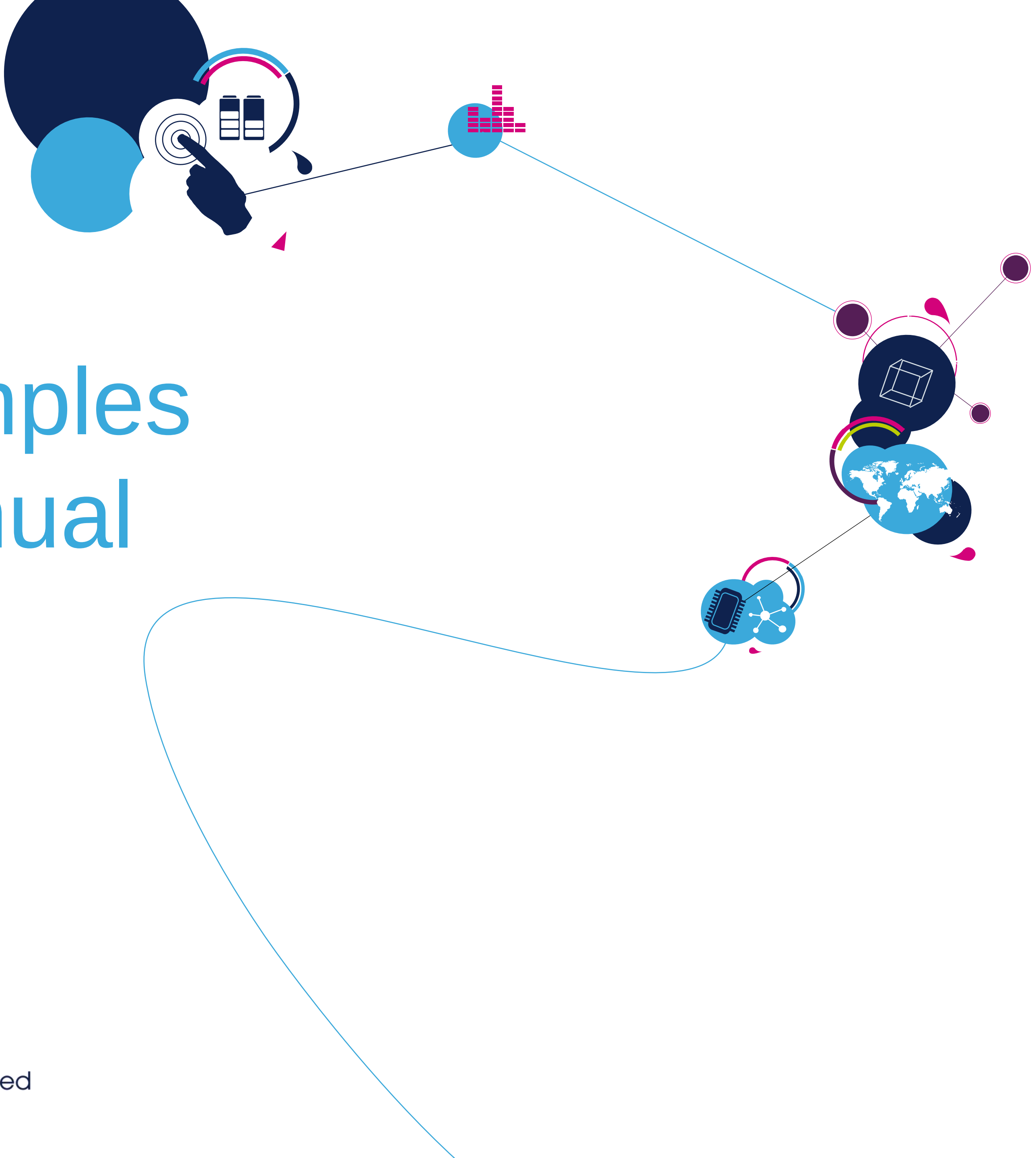
CodeSamples User Manual



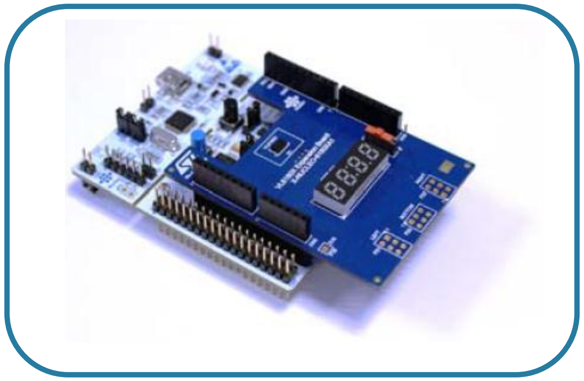
life.augmented

History

Version	Date	Comments
1.0	17 June 2015	Initial

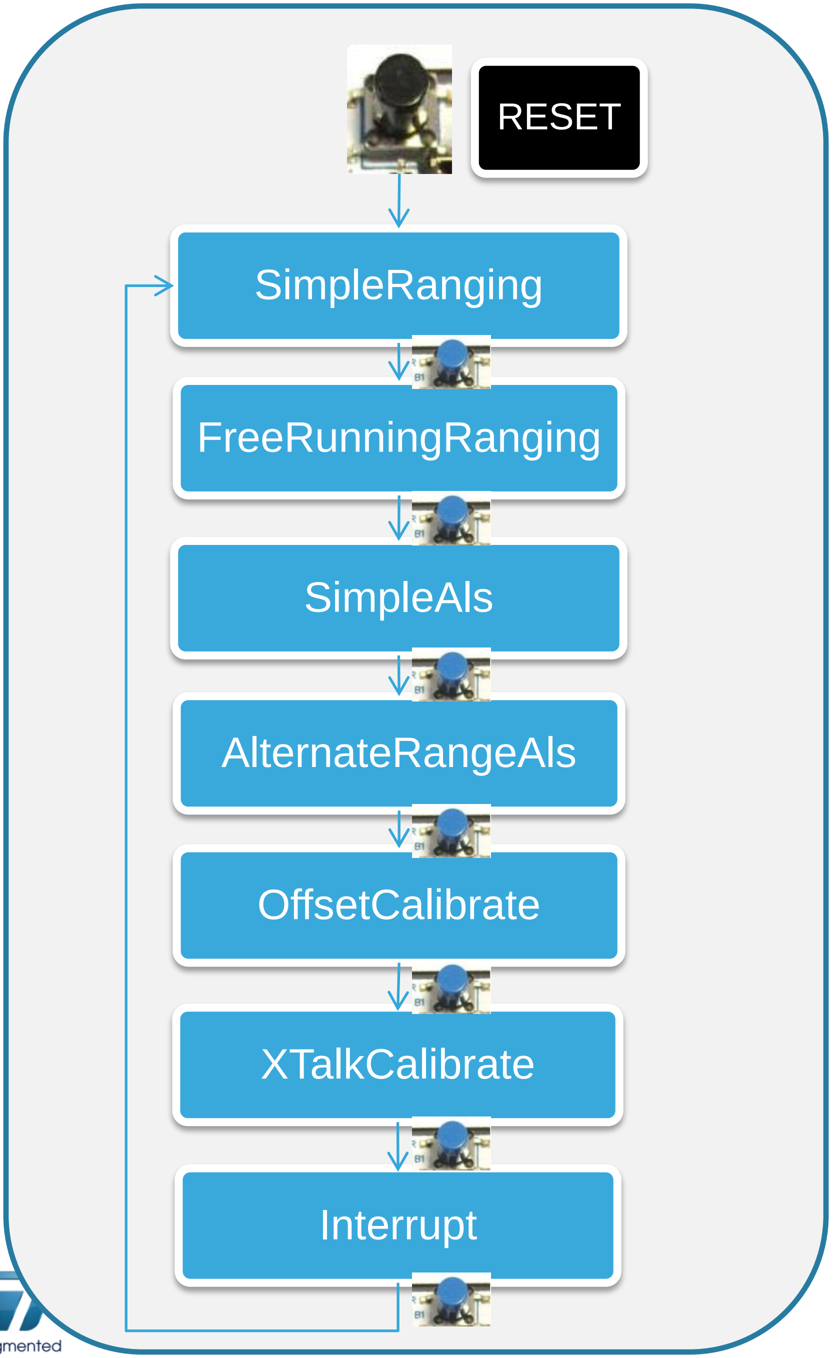
Introduction

- All sample codes provided in API package (under example/code_samples directory) gathered in a single demo
  - To help getting more familiar with API usage
  - To serve as reference platform for API porting
  - To serve as reference for device offset & xTalk calibration procedure
- Use blue button to go from one sample code to the next
- Hardware (validated on F401 Nucleo)



Single VL6180X device

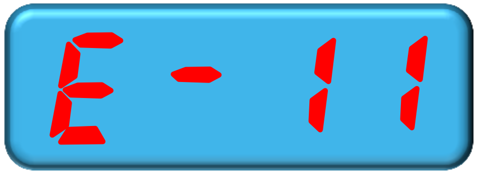
Demo modes


RESET
SimpleRanging
FreeRunningRanging
SimpleAls
AlternateRangeAls
OffsetCalibrate
XTalkCalibrate
Interrupt

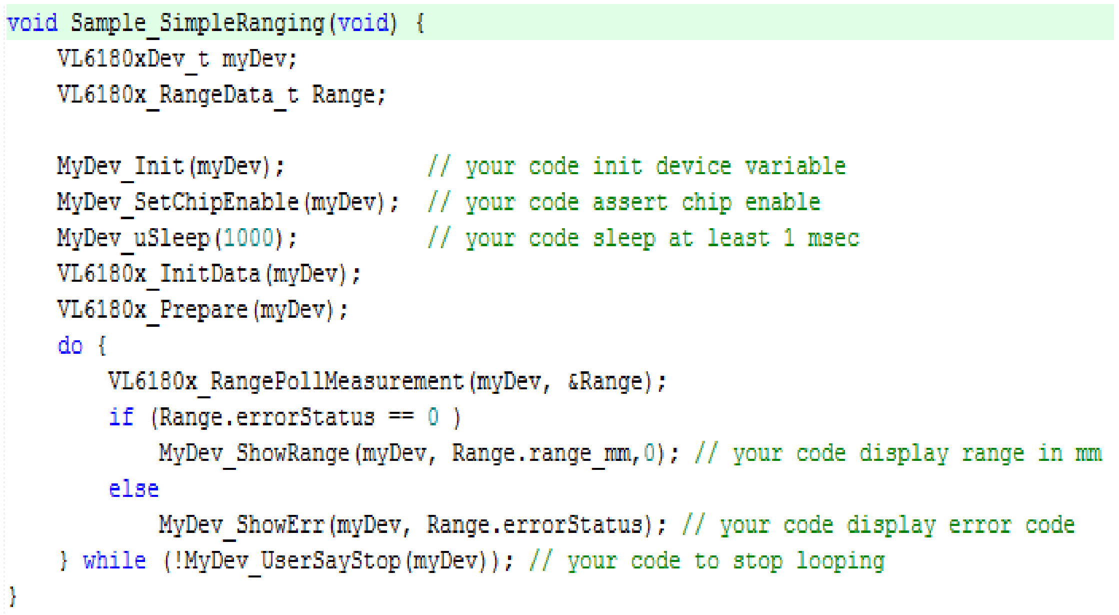
SimpleRanging

- Display distance (in mm) when a target (hand) is detected by the device.
- Scaling factor is x1 so ranging is limited to ~200 mm
- When no target is detected, API error code is displayed
- **VL6180_RangePollMeasurement()** API function used (blocking function)
- Offset & xTalk calibration not done

Target detected at 110 mm



No target detected
(error code – see next slide)

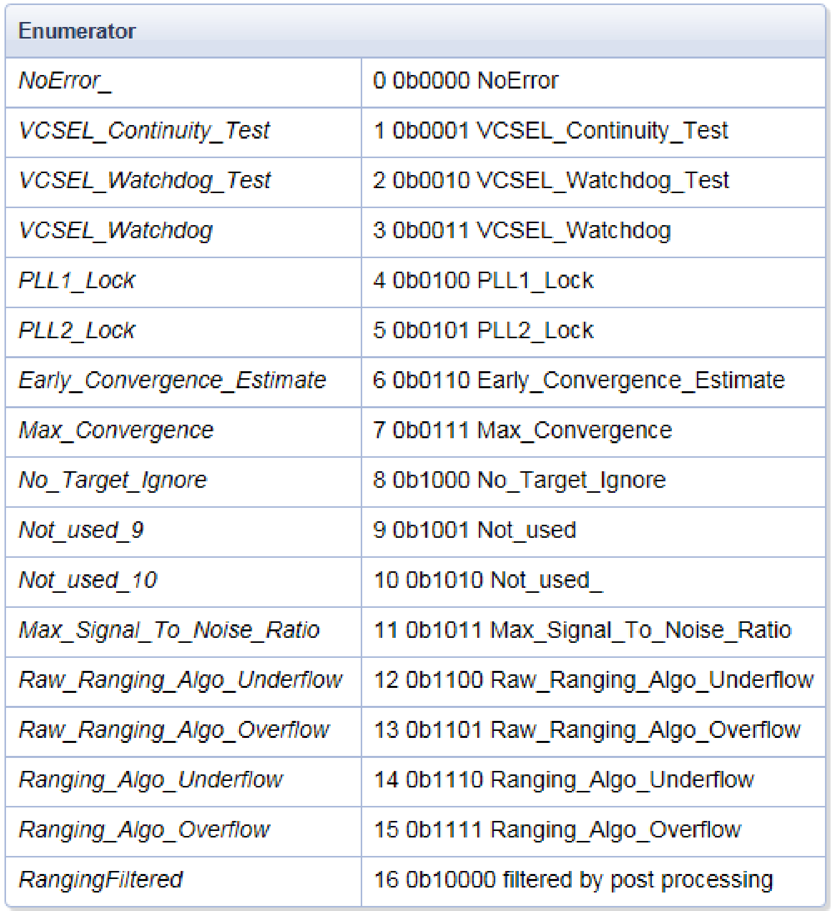


```
void Sample_SimpleRanging(void) {
    VL6180xDev_t myDev;
    VL6180x_RangeData_t Range;

    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
}
```

SimpleRanging

- Error code returned by the ranging functions of the API in case no target is detected

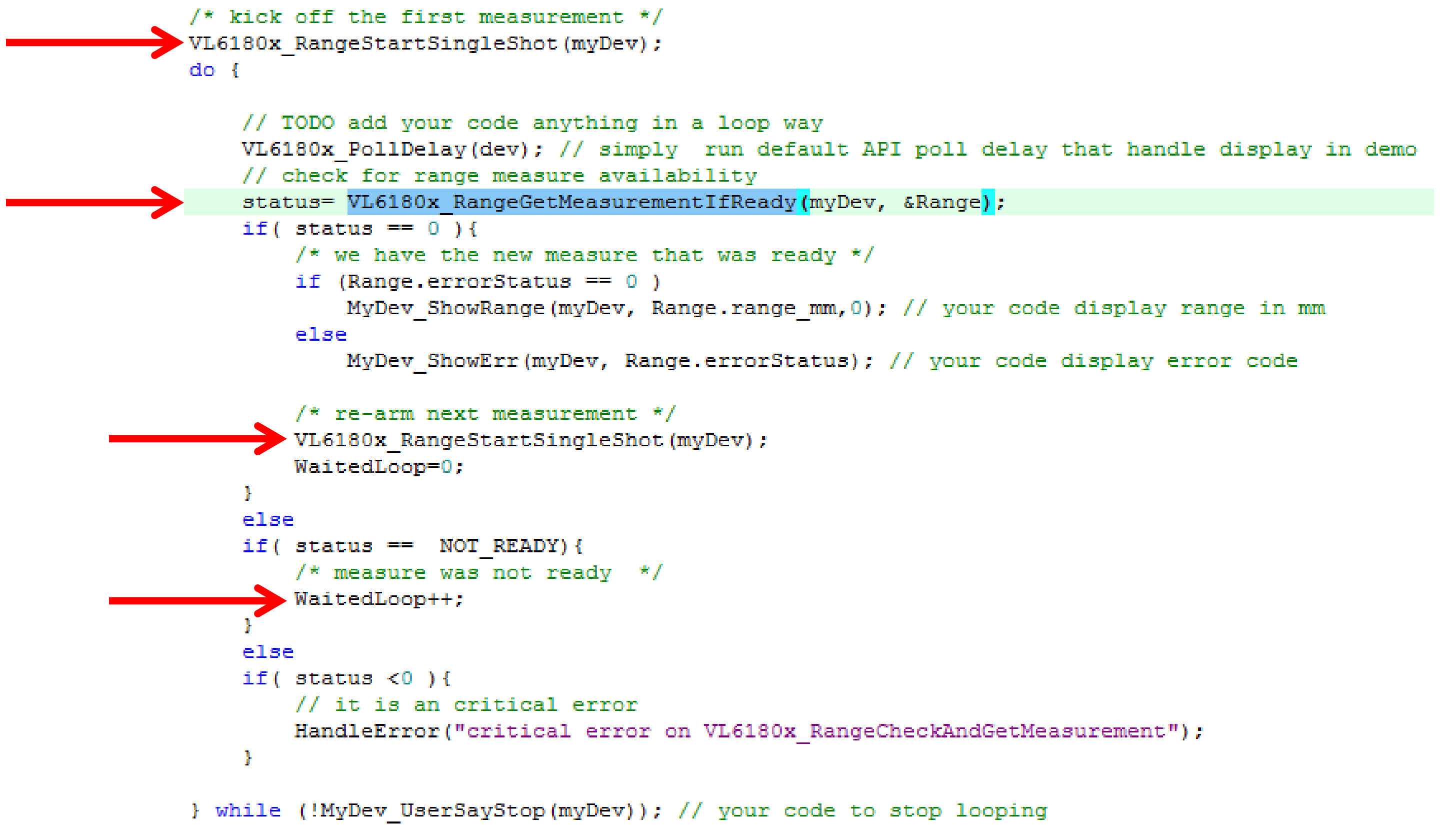


Enumerator	
*NoError_*	0 0b0000 NoError
VCSEL_Continuity_Test	1 0b0001 VCSEL_Continuity_Test
VCSEL_Watchdog_Test	2 0b0010 VCSEL_Watchdog_Test
VCSEL_Watchdog	3 0b0011 VCSEL_Watchdog
PLL1_Lock	4 0b0100 PLL1_Lock
PLL2_Lock	5 0b0101 PLL2_Lock
Early_Convergence_Estimate	6 0b0110 Early_Convergence_Estimate
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

life.augmented

FreeRunningRanging

- Same as SimpleRanging mode except that…
- **VL6180x_RangeGetMeasurementIfReady()** (non-blocking function) API function is used
- CPU is freed during a ranging (in the demo, it refreshes the display only)



```c
/* 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 ){
        /* we have the new measure that was ready */
        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);
        WaitedLoop=0;
    }
    else
    if( status ==  NOT_READY){
        /* measure was not ready  */
        WaitedLoop++;
    }
    else
    if( status <0 ){
        // it is an critical error
        HandleError("critical error on VL6180x_RangeCheckAndGetMeasurement");
    }

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

SimpleAls

- Display light level (in Lux)
- **VL6180x_AlsPollMeasurement()** API function used (blocking function)

Lux level

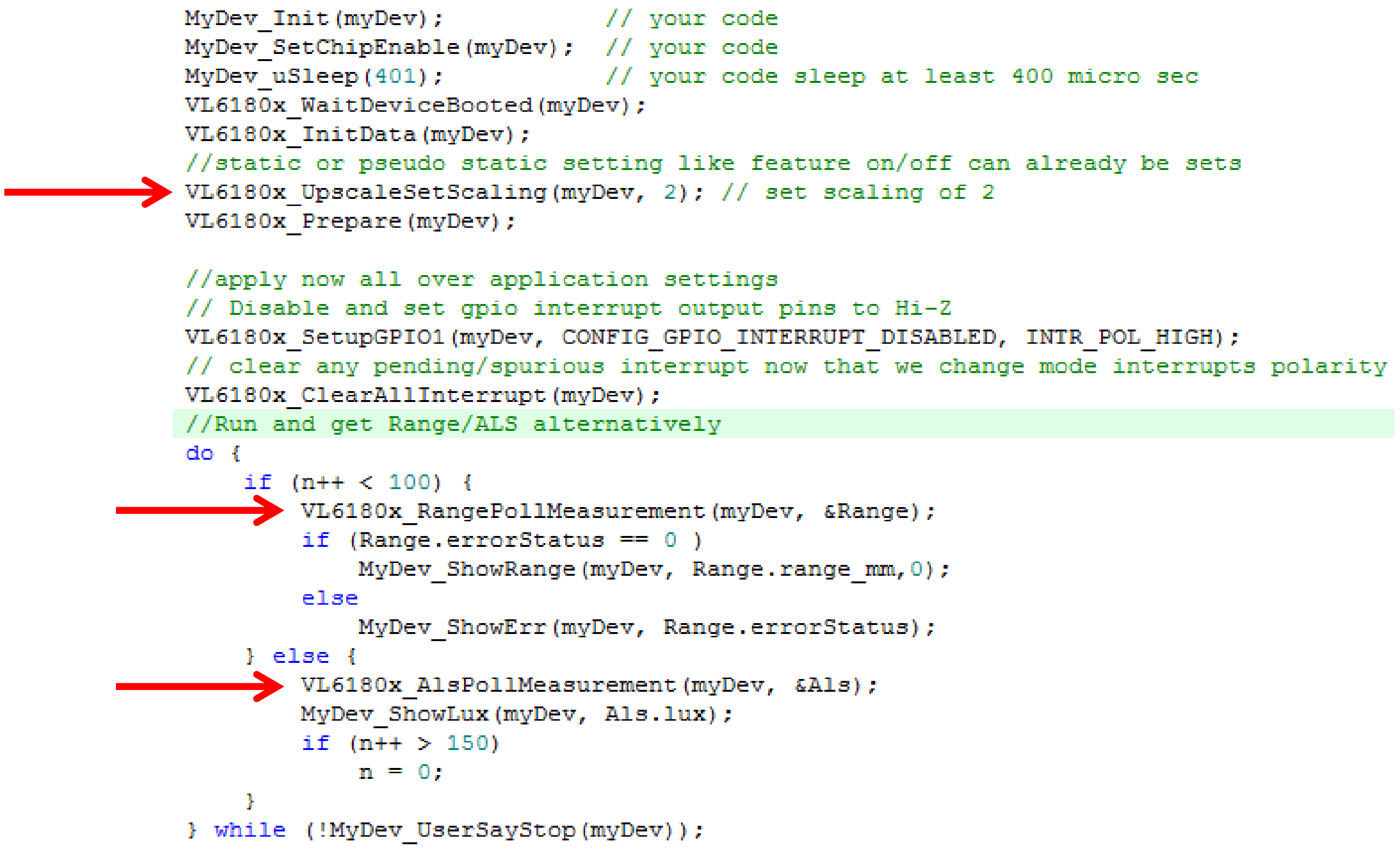
```
void Sample_SimpleAls(void) {
#if VL6180x_ALS_SUPPORT
    VL6180xDev_t myDev;
    VL6180x_AlsData_t Als;

    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_AlsPollMeasurement(myDev, &Als);
        if (Als.errorStatus == 0 )
            MyDev_ShowLux(myDev, Als.lux); // your code display range in mm
        else
            MyDev_ShowErr(myDev, Als.errorStatus); // your code display error code
    } while (!MyDev_UserSayStop(myDev)); // your code to stop looping
#endif
}
```

AlternateRangeAls

- Example showing how to alternate ranging measure (SimpleRanging) and ALS (SimpleAls)

- Scaling factor set to x2, so ranging can go up to ~400 mm



```
MyDev_Init(myDev);              // your code
MyDev_SetChipEnable(myDev);     // your code
MyDev_uSleep(401);              // your code sleep at least 400 micro sec
VL6180x_WaitDeviceBooted(myDev);
VL6180x_InitData(myDev);
//static or pseudo static setting like feature on/off can already be sets
VL6180x_UpscaleSetScaling(myDev, 2); // set scaling of 2
VL6180x_Prepare(myDev);

//apply now all over application settings
// Disable and set gpio interrupt output pins to Hi-Z
VL6180x_SetupGPIO1(myDev, CONFIG_GPIO_INTERRUPT_DISABLED, INTR_POL_HIGH);
// clear any pending/spurious interrupt now that we change mode interrupts polarity
VL6180x_ClearAllInterrupt(myDev);
//Run and get Range/ALS alternatively
do {
    if (n++ < 100) {
        VL6180x_RangePollMeasurement(myDev, &Range);
        if (Range.errorStatus == 0 )
            MyDev_ShowRange(myDev, Range.range_mm,0);
        else
            MyDev_ShowErr(myDev, Range.errorStatus);
    } else {
        VL6180x_AlsPollMeasurement(myDev, &Als);
        MyDev_ShowLux(myDev, Als.lux);
        if (n++ > 150)
            n = 0;
    }
} while (!MyDev_UserSayStop(myDev));
```

OffsetCalibrate

- Run offset calibration procedure and program new offset to the device to get accurate ranging

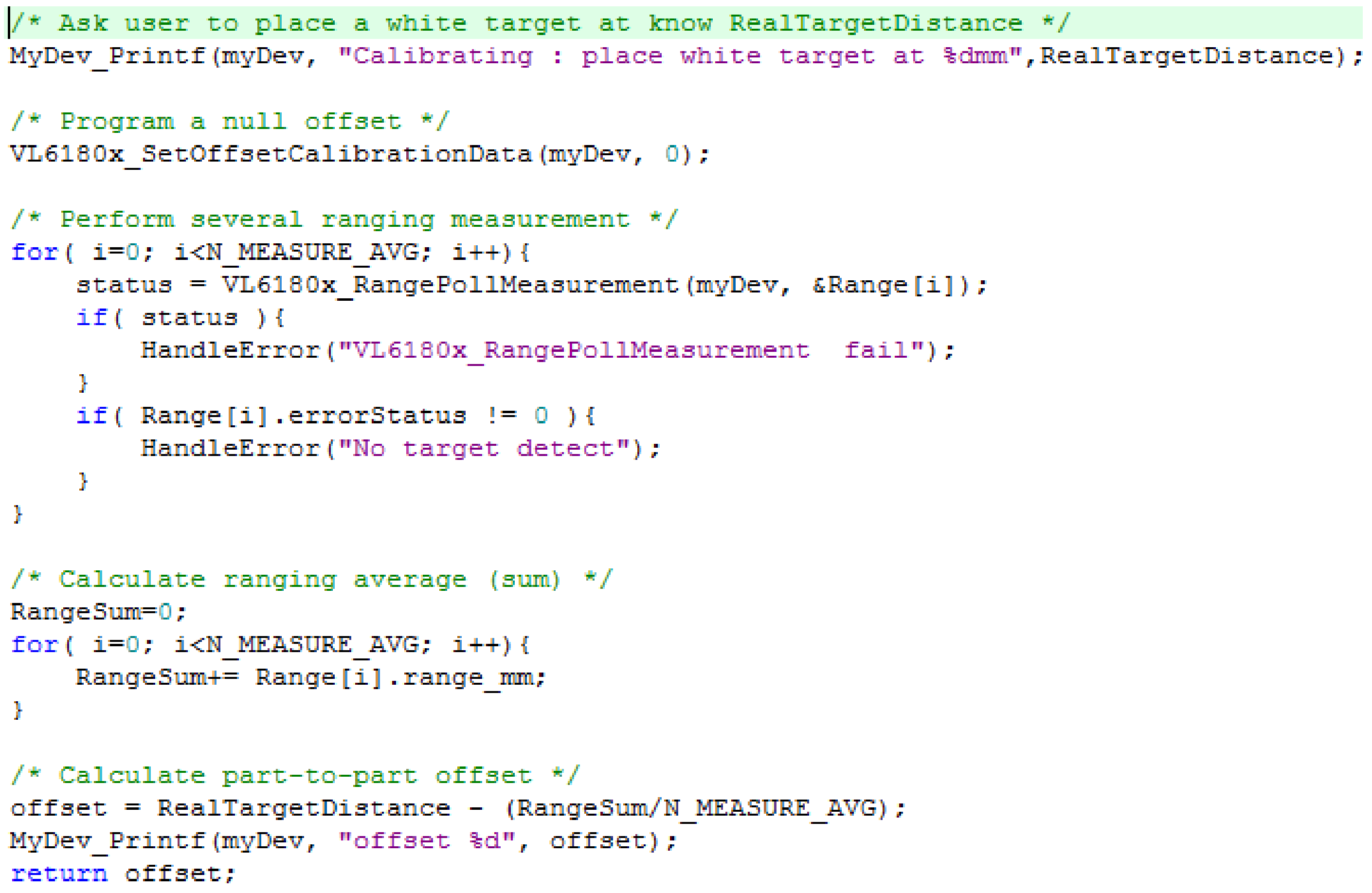
- The procedure requires user interaction through the blue button

- Proceed as follows
  - When starting this mode, the following text is scrolling on the 4-digit display (you'll learn how to read the 7-segments font ☺)
    - "Calibrating : place white target at 50mm"
  - Then, place a white target (88%) at 50mm distance from the device and press the blue button
    - If no target is detected, then procedure fails : press black button to restart the demo from SimpleRanging mode
  - Offset calibration procedure is executed and the new offset is displayed
    - "offset xxx"
  - Press the blue button to program this new offset to the device
  - A SimpleRanging-like mode is now executed with the offset programmed until you press the blue button to go to the next demo mode

- Note that if you reset the device (black button), the programmed offset will be lost and the factory offset will be re-used

- Note that the procedure is slightly different depending on the default scaling factor
  - Scaling x1 : target at 50 mm
  - Scaling x3 : target at 100 mm

OffsetCalibrate



```
/* Ask user to place a white target at know RealTargetDistance */
MyDev_Printf(myDev, "Calibrating : place white target at %dmm",RealTargetDistance);

/* Program a null offset */
VL6180x_SetOffsetCalibrationData(myDev, 0);

/* Perform several ranging measurement */
for( i=0; i<N_MEASURE_AVG; i++){
    status = VL6180x_RangePollMeasurement(myDev, &Range[i]);
    if( status ){
        HandleError("VL6180x_RangePollMeasurement  fail");
    }
    if( Range[i].errorStatus != 0 ){
        HandleError("No target detect");
    }
}

/* Calculate ranging average (sum) */
RangeSum=0;
for( i=0; i<N_MEASURE_AVG; i++){
    RangeSum+= Range[i].range_mm;
}

/* Calculate part-to-part offset */
offset = RealTargetDistance - (RangeSum/N_MEASURE_AVG);
MyDev_Printf(myDev, "offset %d", offset);
return offset;
```

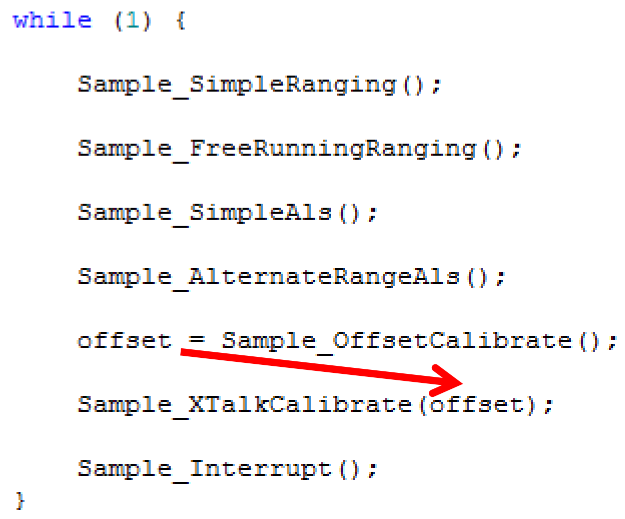
XTalkCalibrate

- Run cross-talk calibration procedure and program xtalk compensation value to the device to get accurate ranging

- Nucleo board is not adapted for putting a cover glass on top of the device (no cover glass holder) so this mode may not be relevant. However, it shows how the procedure should be implemented by the customer when device is integrated into final system (with cover glass and air gap).

- Anyway, for the sake of the demo, it is recommended to place a plastic glass on top of the device (no air gap !) so that the xTalk compensation value that will be calculated and applied will not be null

- Xtalk calibration procedure is meant to be done after the offset calibration procedure (with the right offset programmed to the device). In case a plastic glass is put on top of the device, it is recommended to run the previous **OffsetCalibrate** mode with the same glass : the offset calculated by the Offset calibration procedure will be programmed to the device just before starting the xTalk calibration procedure



```
while (1) {

    Sample_SimpleRanging();

    Sample_FreeRunningRanging();

    Sample_SimpleAls();

    Sample_AlternateRangeAls();

    offset = Sample_OffsetCalibrate();

    Sample_XTalkCalibrate(offset);

    Sample_Interrupt();
}
```

XTalkCalibrate

- The procedure requires user interaction through the blue button
- Proceed as follows
  - When starting this mode, the following text is scrolling on the 4-digit display
    - “Calibrating : place black target at 100mm”
  - Then, place a black target (3%) at 100mm distance from the device and press the blue button
    - If no target is detected, then procedure fails : press black button to restart the demo from SimpleRanging mode
  - Cross-talk calibration procedure is executed and the following message is displayed (scrolling)
    - “range xxx rate yyy comp zzz”
    - xxx is the average range measured during the calibration procedure (before xTalk compensation)
    - yyy is the average return sgnal rate measured during the calibration procedure (before xTalk compensation)
    - zzz is the xTalk compensation value that will be programmed to the device (coded as 9.7 fixed-point)
  - Press the blue button to program the calculated xTalk compensation value to the device
  - A SimpleRanging-like mode is now executed with the until you press the blue button to go to the next demo mode
- Note that if you reset the device (black button), the programmed xTalk compensation value will be lost
- Note that the procedure is slightly different depending on the default scaling factor
  - Scaling x1 : target at 100 mm
  - Scaling x3 : target at 400 mm

life.augmented

XTalkCalibrate

```
/* Ask user to place a black target at know RealTargetDistance from glass */
MyDev_Printf(myDev, "Calibrating : place black target at %dmm",RealTargetDistance);

/* Program a null xTalk compensation value */
status=VL6180x_WrWord(myDev, SYSRANGE_CROSSTALK_COMPENSATION_RATE, 0);

/* Perform several ranging measurement */
for( i=0; i<N_MEASURE_AVG; i++){
    status = VL6180x_RangePollMeasurement(myDev, &Range[i]);
    if( status ){
        HandleError("VL6180x_RangePollMeasurement  fail");
    }
    if( Range[i].errorStatus != 0 ){
        HandleError("No target detect");
    }
}

/* Calculate ranging and signal rate average */
RangeSum=0;
RateSum=0;
for( i=0; i<N_MEASURE_AVG; i++){
    RangeSum+= Range[i].range_mm;
    RateSum+= Range[i].signalRate_mcps;
}

/* Rate sum is 9.7 fixpoint so same for xtlak computed below
 * The order of operation is important to have decent final precision without use of  floating point
 * using a higher real distance and number of point may lead to 32 bit integer overflow in formula below */
XTalkInt = RateSum*(N_MEASURE_AVG*RealTargetDistance-RangeSum)/N_MEASURE_AVG /(RealTargetDistance*N_MEASURE_AVG);
XTalkInt = (XTalkInt>0) ? XTalkInt : 0; // Must be positive of null
MyDev_Printf(myDev, "range %d rate %d comp %d\n", RangeSum/N_MEASURE_AVG, RateSum/N_MEASURE_AVG, XTalkInt);

return XTalkInt;
```

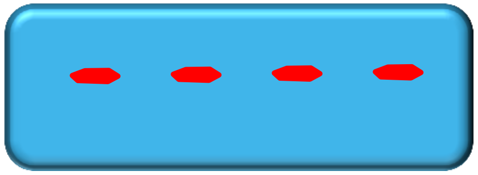
life.augmented

Interrupt

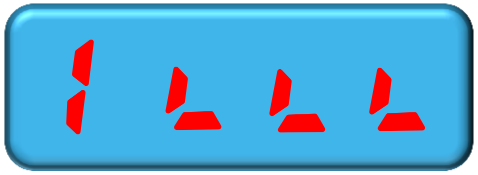
- Program the device in continuous ranging mode so that CPU is waked up by the device only when the measured distance is
  - Lower than threshold (100 mm) : Alarm Low mode
  - Or higher than a threshold (200 mm) : Alarm High mode
  - Or within a given range ([100:200]) : Alarm Window mode
- Pressing the blue button allows to go from Alarm Low mode to Alarm High mode and then to Alarm Window mode (see next slide)
- Scaling factor is set to x2 so that ranging can reach ~ 400 mm
- This demo shows how to program the device so that it can be used for User Detection applications
- In this demo, CPU is never put in a sleep mode as it needs to manage the display refresh even if no interrupt is generated from the device

Interrupt : Alarm Low mode

- Alarm Low mode



Target > 100 mm
(no interrupt generated)

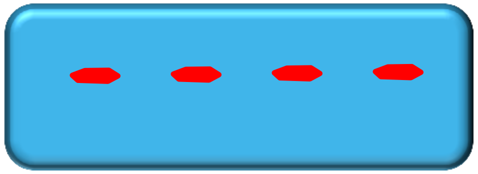


Target <= 100 mm
(interrupt generated)

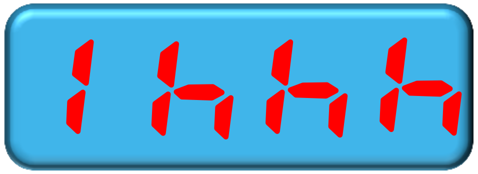
- Press blue button **when target distance is less than 100 mm** to switch to Alarm High mode (otherwise, blue button is not taken into account ! This is just a choice for the sake of the demo)

Interrupt : Alarm High mode

- Alarm High mode



Target < 200 mm
(no interrupt generated)

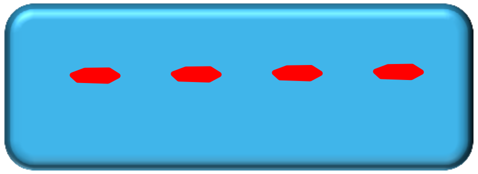


Target >= 200 mm
(interrupt generated)

- Press blue button when target distance is more than 200 mm to switch to Alarm Window mode (otherwise, blue button is not taken into account !)

Interrupt : Alarm Window mode

- Alarm Window mode



200 mm < Target < 400 mm
(no interrupt generated)

otherwise
(interrupt generated)

- Press blue button when target distance is not in the [100:200] window to switch back to **SimpleRanging** mode (otherwise, blue button is not taken into account !)