

## Use of I<sup>2</sup>C Component on Microcontroller

**Targeted competences:** Use of I<sup>2</sup>C triple axis accelerometer with a sampling frequency of 100Hz

**Hardware:** STM32F7 Nucleo board

**Framework:** Keil  $\mu$ vision (V5.25.2.0) and CubeMX (V5.0.1) from STMicroelectronics

The aim of this document is to show how to use an I<sup>2</sup>C bus in order to read the data from a 3-Axis accelerometer. In function of values provided by the accelerometer, we will switch on a led.



Figure 1: Acquisition Stream

To obtain the sampling frequency we need to add a timer into this scheme. This timer will avoid making a blocking polling on the I<sup>2</sup>C bus and avoid obtaining too many unnecessary values from the sensor. Here, we choose to sample at 400Hz; this value can be chosen in function of your application (respect the frequency period of your sensors).

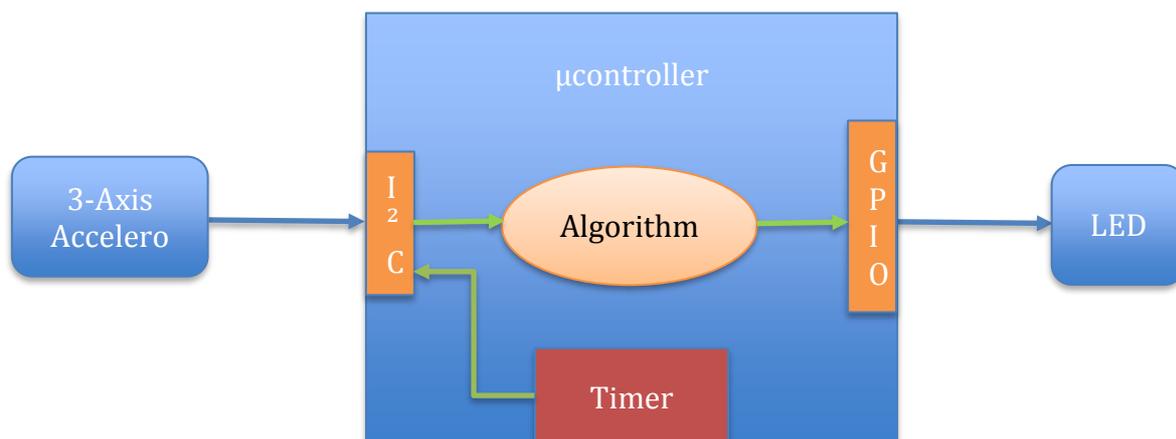


Figure 2: Global Scheme

### 1. Microcontroller configuration

The first step is to configure the microcontroller. In our case, we use the NUCLEO-F767ZI platform based on a STM32F7 architecture. In order to configure this board we

will use the CubeMx software. After the choice of the board (attention the configuration can depend on the board used) you obtain the figure below.

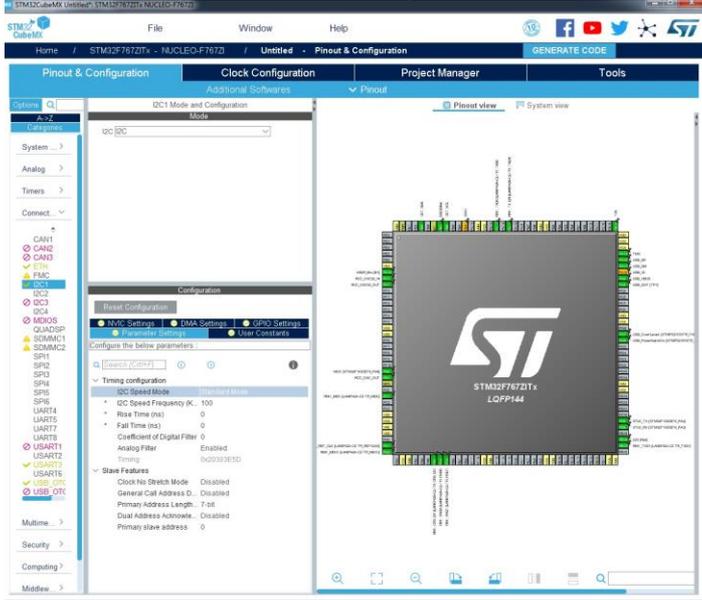


Figure 3: CubeMx configuration for I<sup>2</sup>C

Here, I chose to use the I<sup>2</sup>C bus in standard mode so the frequency of the bus is equal to 400KHz furthermore the address length for the peripherals is in 7-bit mode (see our component datasheet if you use a different peripheral than the one is used). The pin PB6 is used for the SCL signal and the pin PB9 for the SDA signal. Here I cannot explain the behavior of the I<sup>2</sup>C bus; you can find information at the following address: [https://en.wikipedia.org/wiki/I<sup>2</sup>C](https://en.wikipedia.org/wiki/I%C2%B2C)

I configure also the NVIC (enabled the interruption in NVIC settings) in order to interruptions generated when an error or an event appears on the bus.

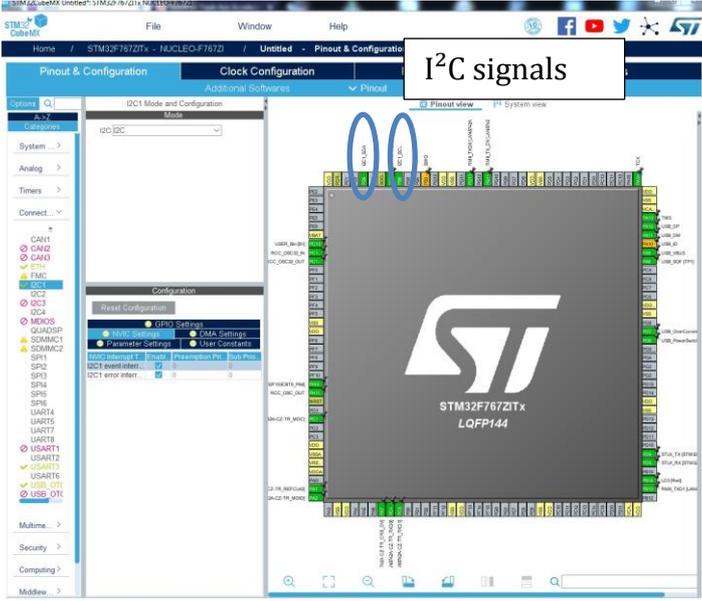


Figure 4: I<sup>2</sup>C signals and NVIC settings

Now we have to configure the clock for the whole board; to do this click on clock configuration item. With this configuration panel, we can choose the frequency of different  $\mu$ controller components as CPU frequency, AHB, APB1 buses and so on. I decide to use the maximum frequency of the SYSCLK that is 216MHz. In this case, the frequency of the APB1 and APB2 timer clocks are respectively 108MHz and 216MHz (see Figure 5). These frequencies will be very important in the next step when we will have to configure the sampling frequency.

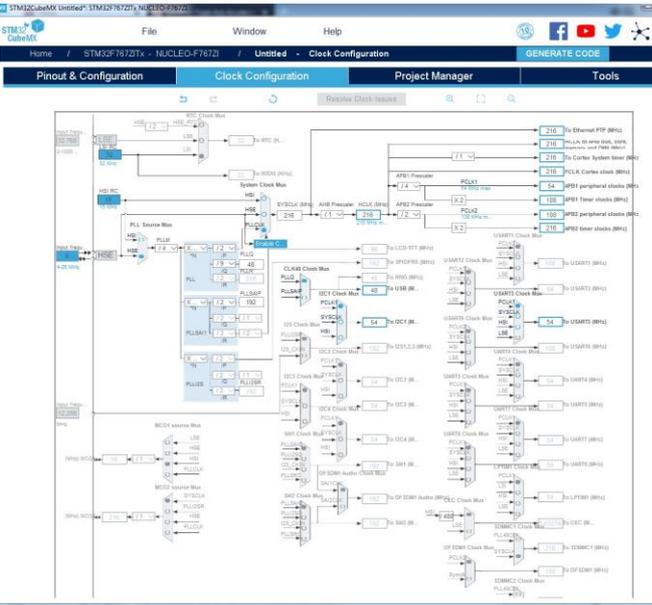


Figure 5: Clock configuration for the board

Now we are going to configure the timer so that generates the sampling frequency. First click and configuration panel then on TIM1 now, we are going to determine the values for the prescaler and the counter period. The prescaler value divides the clock timer frequency in order to obtain a small one; it is useful if you want to have a small sampling frequency. Indeed, as the frequency in our case is 216 MHz and as the maximum count value is 65535 (16-bit timer) the minimum of the frequency scaling can be 3296Hz.

For this example, I would like a sampling frequency of 400Hz so I chose the following values: prescaler value 216 (in this case, the clock frequency of the timer will be 1MHz instead of 216 MHz), the counter period value 2500 (2500 ticks at 1 MHz represents a sampling period of 2.5ms).

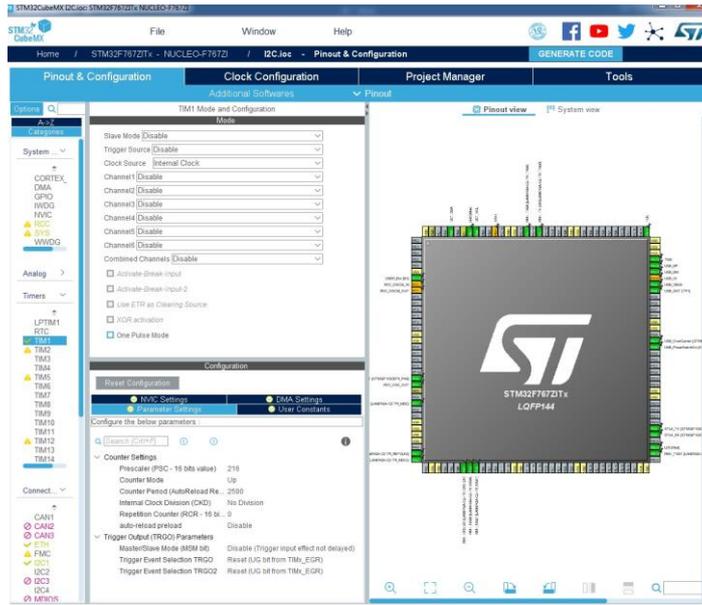


Figure 6: Timer configuration to create the 400Hz sampling frequency

The last thing to do here is to enable the interruption for the timer. To do this, click on the NVIC Settings panel and enables the TIM1 update interrupt and TIM10 global interrupt by checking Enabled.

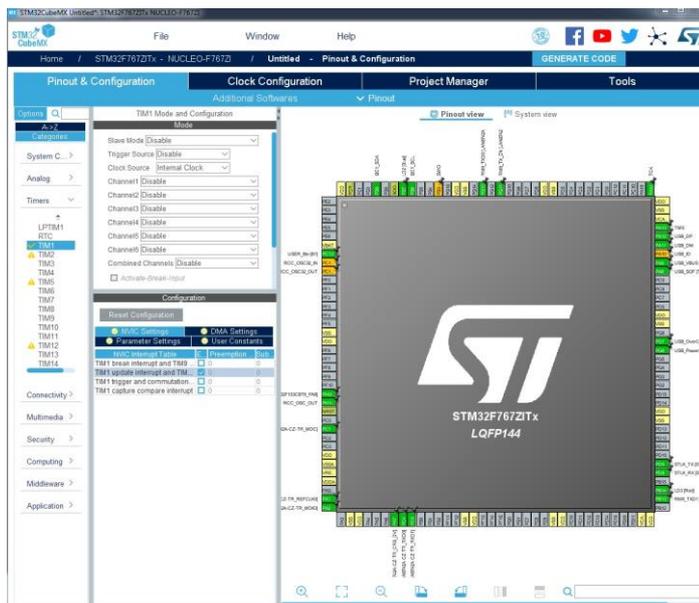
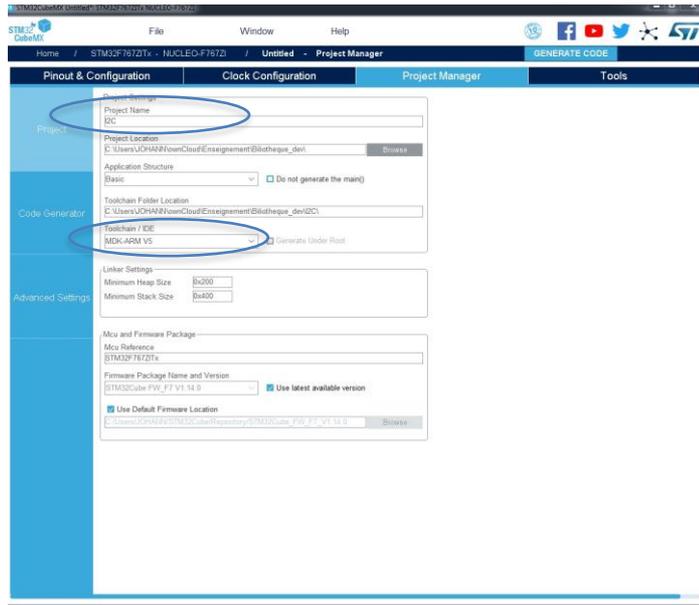


Figure 7: Timer interrupt configuration

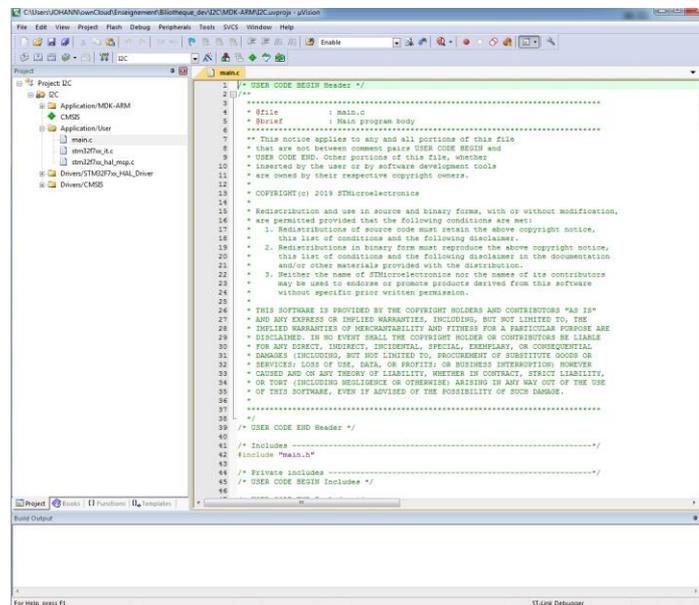
Now, we are going to configure the project manager to generate the different useful files.



**Figure 8: Project manager configuration**

You must give a name for your project and choose the toolchain that you want to use; here I choose to use MDK-ARM toolchain.

After clicking on Generate Code, CubeMx generates the application code and creates the MDK-ARM project as shown in the figure below.



**Figure 9: Code generation**

The next step is to define all the register addresses of the peripheral. In my case, I use a LIS3DH, which is a 3-axis accelerometer. In order to define all the addresses I wrote an include file (accelero.h); you can find below a part of this file.

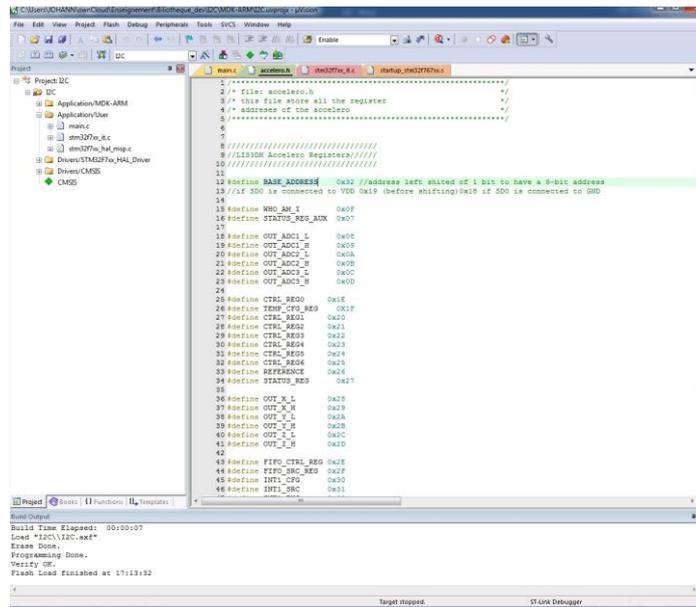


Figure 10: Include file defining the addresses of the LIS3DH component

Now, we must setup the I<sup>2</sup>C sensor and start the timer in order to generate the sampling frequency of 400Hz. To do that we have just to use the HAL driver function for the timer and write a function to setup the sensor.

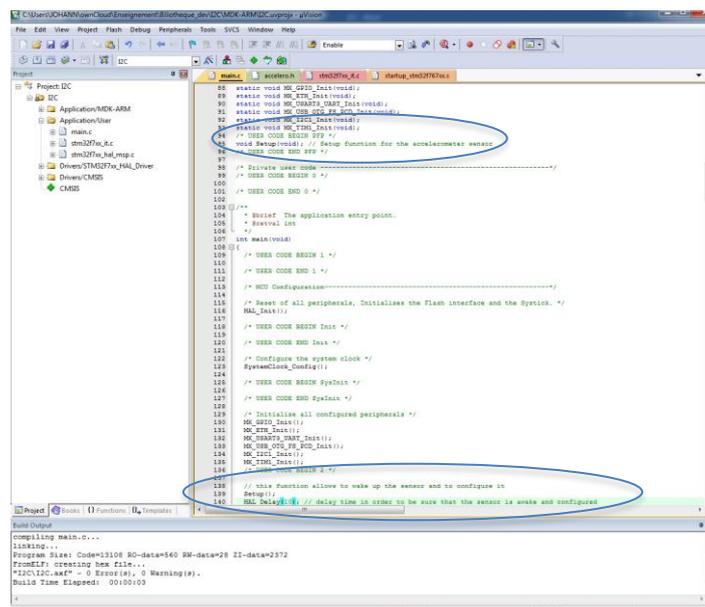


Figure 11: Declaration of the Setup Function and its call

Below, you can find the Setup function code that allows configuring the LIS3DH sensor. I configure it to use a 400Hz sampling frequency and enable all the axis of the accelerometer.

```

460
461 /*Configure GPIO pin ( USB_PowerSwitchOn_Pin */
462 GPIO_InitStruct.Pin = USB_PowerSwitchOn_Pin
463 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP
464 GPIO_InitStruct.Pull = GPIO_NOPULL
465 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW
466 HAL_GPIO_Init(USB_PowerSwitchOn_GPIO_Port, &GPIO_InitStruct);
467
468 /*Configure GPIO pin ( USB_OverCurrent_Pin */
469 GPIO_InitStruct.Pin = USB_OverCurrent_Pin
470 GPIO_InitStruct.Mode = GPIO_MODE_INPUT
471 GPIO_InitStruct.Pull = GPIO_NOPULL
472 HAL_GPIO_Init(USB_OverCurrent_GPIO_Port, &GPIO_InitStruct);
473
474
475
476
477 /* USER CODE BEGIN 4 */
478 void Setup(void)
479 {
480 //autorunne tout les axes du gyro
481 TabBuffer[]=0x7F; // Allow to choose the ctrl reg e destination of the command
482 HAL_I2C_Master_Transmit(&hi2c1, BASE_ADDRESS, (uint8_t *)TabBuffer,1,100); //Allow to store 0x7F in t
483
484 }
485 /* USER CODE END 4 */
486
487
488
489 /* Brief Description of this function is executed in case of error occurrence.
490 * Retval None
491 */
492 void Error_Handler(void)
493 {
494 /* USER CODE BEGIN Error_Handler_Debug */
495 /* You can add your own implementation to report the HAL error return state */
496
497 /* USER CODE END Error_Handler_Debug */
498
499 #ifndef USE_FULL_ASSERT
500 #error "Please enable USE_FULL_ASSERT in the user code."
501 #endif
502 #ifdef __GNUC__
503 #error "Assertion failed in file " __FILE__ " at line " __LINE__
504 #endif
505 #endif
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Figure 12: Setup function code to configure the sensor

Now, the sensor is ready so we can start the timer in IT mode in order to generate the sampling frequency.

```

126
127 /* USER CODE BEGIN SysInit */
128
129 /* USER CODE END SysInit */
130
131 // Initialise all configured peripherals */
132 MX_GPIO_Init();
133 MX_ETH_Init();
134 MX_UART2_USART_Init();
135 MX_USART1_UART_Init();
136 MX_I2C1_Init();
137 MX_TIM1_Init();
138
139 /* USER CODE BEGIN 2 */
140
141
142 // this function allows to wake up the sensor and to configure it
143 Setup();
144
145 //HAL_TIM_Base_Start_IT(&htim1); //start the timer 1 in order to obtain a sampling period of 4000µs
146
147
148 /* USER CODE END 2 */
149
150
151
152
153 /* USER CODE BEGIN 3 */
154
155 /* USER CODE END 3 */
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Figure 13: Starting the sampling timer

To be able to store the 3-axis value of the accelerometer, we declare three variables to store them.

Paid attention, each axis uses two registers to store its value one for the 8-bit LSB and another for the 8-bit MSB. Therefore, to store the complete value I shift the 8-bit MSB to the left (shifted by 8) and then I do a mask (a OR function) between this value and the 8-bit LSB otherwise the MSB value will be modified. I realized that for the three axis of course. This requires realizing two reads one for each register as you can see on the next figure.

```

27 #include "main.h"
28 #include "stm32f7xx_it.h"
29 /* Private includes ----- */
30 /* USER CODE BEGIN Includes */
31 #include "stm32f7xx_hal.h"
32 /* USER CODE END Includes */
33
34 /* Private typedefs ----- */
35 /* USER CODE BEGIN TD */
36 /* USER CODE END TD */
37
38 /* Private defines ----- */
39 /* USER CODE BEGIN PD */
40 /* USER CODE END PD */
41
42 /* Private macros ----- */
43 /* USER CODE BEGIN PM */
44 /* USER CODE END PM */
45
46 /* Private variables ----- */
47 /* USER CODE BEGIN PV */
48 extern uint8_t TxBuffer[];
49 extern uint16_t X_Axis;
50 extern uint16_t Y_Axis;
51 extern uint16_t Z_Axis;
52 /* USER CODE END PV */
53
54 /* Private function prototypes ----- */
55 /* USER CODE BEGIN PFP */
56 /* USER CODE END PFP */
57
58 /* Private user code ----- */
59 /* USER CODE BEGIN UC */
60
61 /* USER CODE END UC */
62
63 /* External variables ----- */
64 extern I2C_HandleTypeDef hi2c1;
65 extern TIM_HandleTypeDef htim1;
66 /* USER CODE BEGIN EV */
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82

```

Figure 14: Variable declarations to store the 3-axis values

The first step is to know if the peripheral is the good one (in our case the LIS3DH). To do that I read the Who\_am\_I register of the peripheral and I compare it with the internal address of the sensor (here the address value is 0x33). If the sensor is the good one then I read the two register of each axis. To do this, I write the address of the desired register in the TxBuffer and write it on the I<sup>2</sup>C bus (by using the HAL\_I2C\_Master\_Transmit function), then I read the returned value on the I<sup>2</sup>C bus (by using the HAL\_I2C\_Master\_Receive function). This entire job is done in the timer interrupt function.

```

211 /* USER CODE END SysTick_Handler */
212
213
214
215
216
217
218
219
220
221
222
223
224 void TIM1_UP_TIM10_IRQHandler(void)
225 {
226 /* USER CODE BEGIN TIM1_UP_TIM10_IRQ_0 */
227 TxBuffer[0] = WHO_AM_I; //Write the Who_Am_I add in the Tx Buffer
228 HAL_I2C_Master_Transmit(hi2c1, BASE_ADDRESS, (uint8_t*)TxBuffer, 1, 100); //Ask the name of the perip
229 HAL_I2C_Master_Receive(hi2c1, BASE_ADDRESS, (uint8_t*)TxBuffer, 1, 100); //read the name of the per
230 HAL_Delay(1);
231
232 if (TxBuffer[0] == 0x33) //verify if the peripheral is the LIS3DH
233 {
234 // X axis value
235 TxBuffer[0] = OUT_X_H; //put the output address of the axis X value
236 HAL_I2C_Master_Transmit(hi2c1, BASE_ADDRESS, (uint8_t*)TxBuffer, 1, 100); // as previous
237 HAL_I2C_Master_Receive(hi2c1, BASE_ADDRESS, (uint8_t*)TxBuffer, 1, 100); // read the LSB value of
238 X_Axis = TxBuffer[0] << 8;
239 HAL_Delay(1);
240
241 TxBuffer[0] = OUT_X_L; //put the output address of the axis X value
242 HAL_I2C_Master_Transmit(hi2c1, BASE_ADDRESS, (uint8_t*)TxBuffer, 1, 100); // as previous
243 HAL_I2C_Master_Receive(hi2c1, BASE_ADDRESS, (uint8_t*)TxBuffer, 1, 100); // read the LSB value of
244 X_Axis = X_Axis | TxBuffer[0];
245 HAL_Delay(1);
246
247 // Y Axis value
248 TxBuffer[0] = OUT_Y_H; //put the output address of the axis Y value
249 HAL_I2C_Master_Transmit(hi2c1, BASE_ADDRESS, (uint8_t*)TxBuffer, 1, 100); // as previous
250 HAL_I2C_Master_Receive(hi2c1, BASE_ADDRESS, (uint8_t*)TxBuffer, 1, 100); // read the MSB value of
251 Y_Axis = TxBuffer[0] << 8;
252 HAL_Delay(1);
253
254 TxBuffer[0] = OUT_Y_L; //put the output address of the axis Y value
255 HAL_I2C_Master_Transmit(hi2c1, BASE_ADDRESS, (uint8_t*)TxBuffer, 1, 100); // as previous
256 HAL_I2C_Master_Receive(hi2c1, BASE_ADDRESS, (uint8_t*)TxBuffer, 1, 100); // read the LSB value of
257 Y_Axis = Y_Axis | TxBuffer[0];
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500

```

Figure 15: Code to read the axis values

To be sure that the code is ok I switch on a led (LD3) if the X-axis value is greater than 10000 (the value of the x-axis can be from 0 to 65535 since it is stored in 16-bit variable).

