

Use of UART Component on Microcontroller

Targeted competences: Use of UART in order to retrieve GPS data at each PPS generation

Hardware: STM32F7 Nucleo board

Framework: Keil μ vision (V5.25.2.0) and CubeMX (V5.1.0) from STMicroelectronics

The aim of this document is to show how to use an UART bus in order to read the data from GPS sensor.

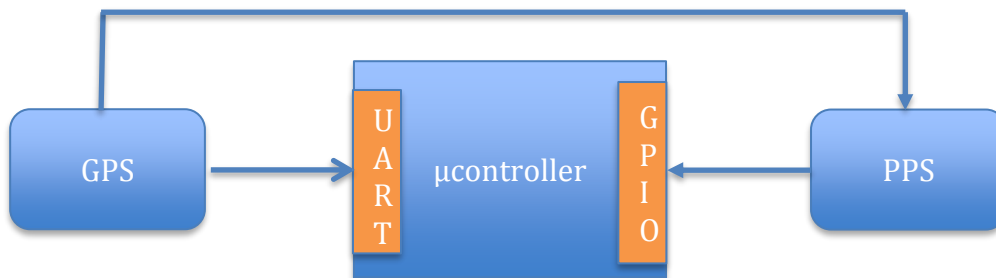


Figure 1: Acquisition Stream

In order to sample the read of the GPS data, we use the PPS (Pulse Per Second) signal comes from the GPS sensor.

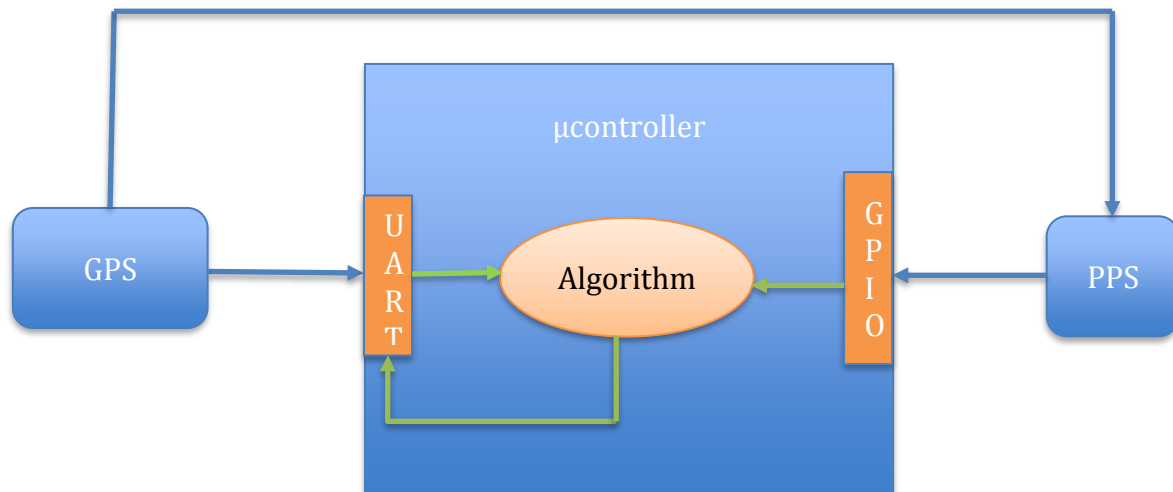


Figure 2: Global Scheme

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.

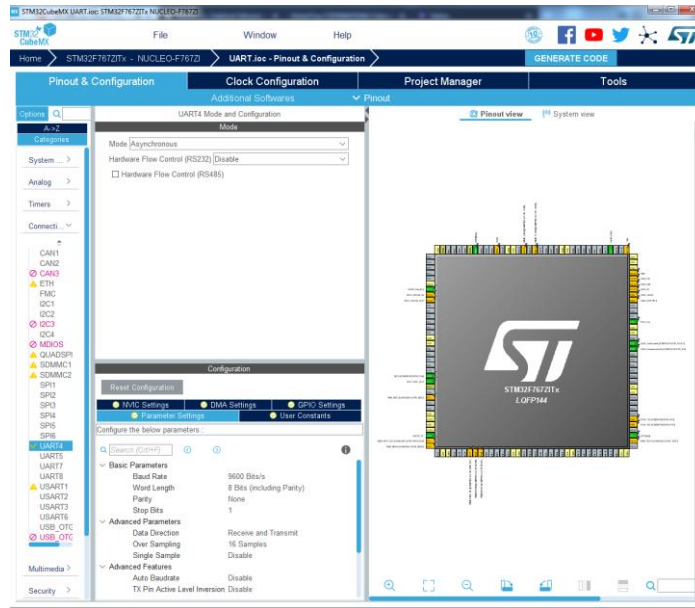


Figure 3: CubeMx configuration for UART

Here, I chose to use the UART bus with the following configuration: 9600bit/s, word length 8 bit, no parity and 1 stop bit. The UART uses no hardware flow control

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.

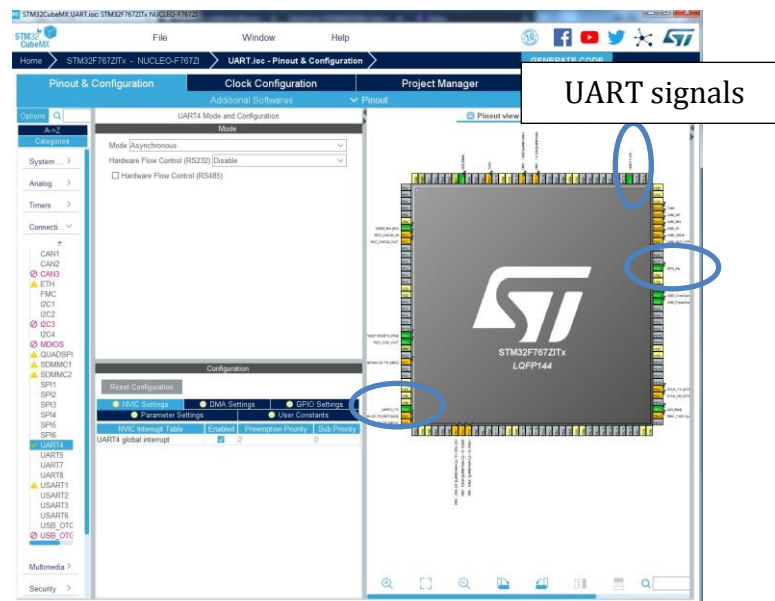


Figure 4: UART signals and NVIC settings

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 μ 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)..

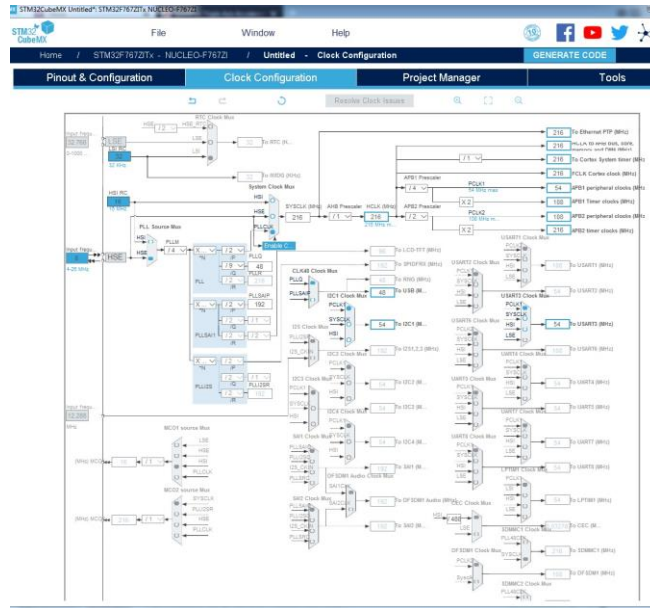


Figure 5: Clock configuration for the board

I have configured the PC6 pin as an external interrupt with a rising edge detection. This GPIO is connected to the PPS signal comes from the GPS board. This external interrupt will be used to start the read of the UART_Rx pin in order to retrieve the NMEA frame.

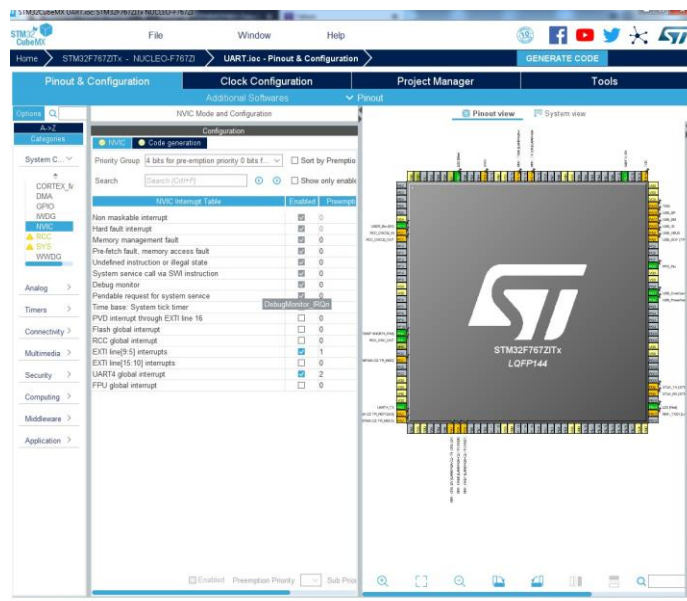


Figure 6: Timer interrupt configuration

We can see in Figure 6 that the EXTI line is enabled and its preemption priority is equal to 1. I have also enabled the UART4 global interrupt with the 2 level priority; this means that the PPS signal (PC6 pin) has a higher interrupt priority level than the uart. Using the UART interrupt avoids us to not realize a blocking pooling on the UART bus in order to know if a data is ready to be read.

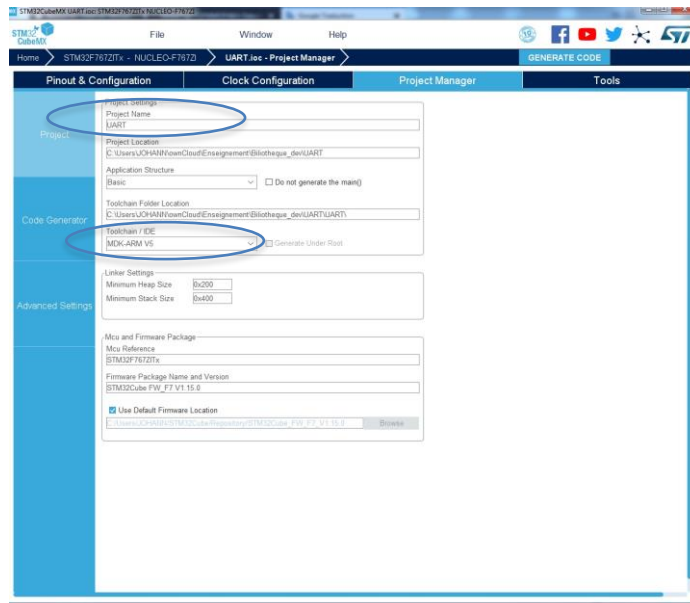


Figure 7: Project manager configuration

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.

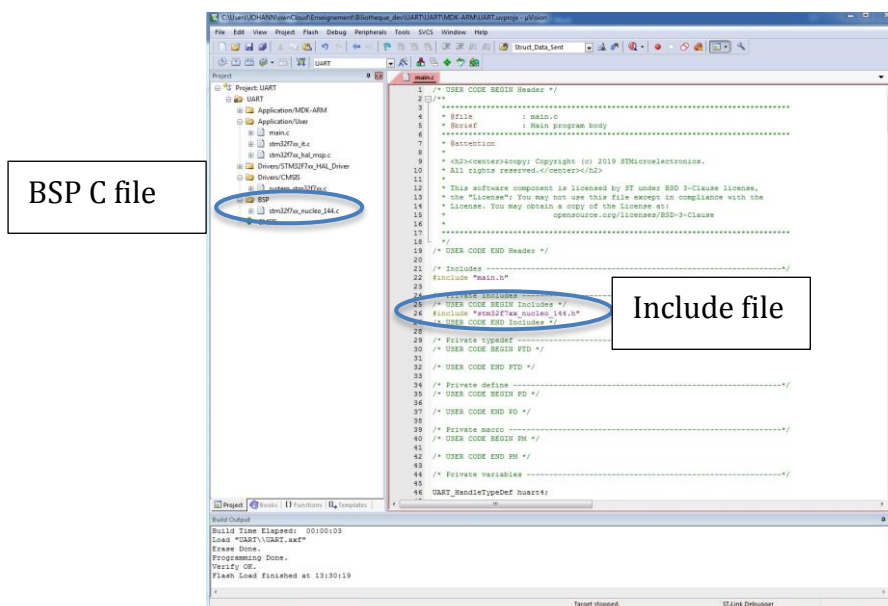


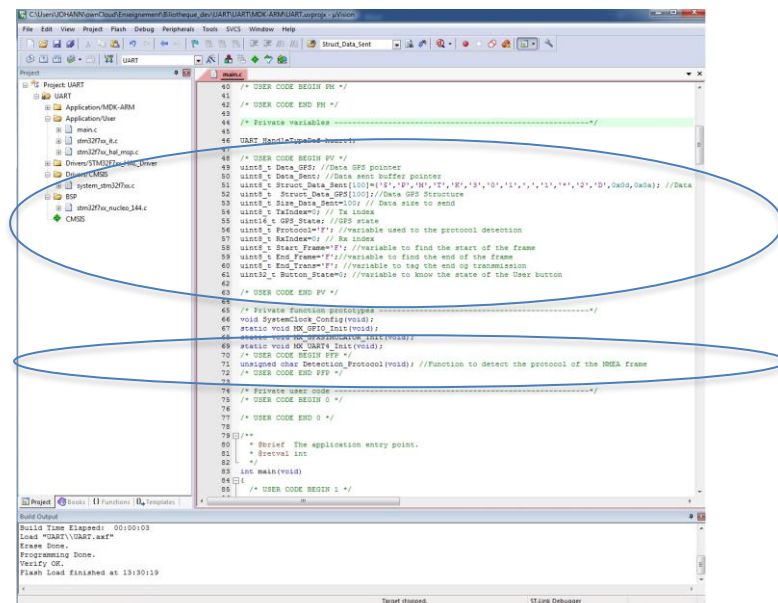
Figure 8: Code generation

As I want to add the BSP function in this project, I include the following header file: stm32f7xx_nucleo_144.h and I add also the associated C file in the project.

In the figure below, you can see all the variable that I have defined to realize the algorithm. Some of them are not used in this code but I created a generic code that you can use if you want to change the configuration of the GPS. The GPS board can receive commands from the μ controller in order to change for instance the baud rate of the transmission, realize a cold start and so on (see the command documentation here: https://cdn-shop.adafruit.com/datasheets/PMTK_A11.pdf).

The datasheet of the GPS module can be found here: <https://cdn-shop.adafruit.com/datasheets/GlobalTop-FGPMMPA6H-Datasheet-V0A.pdf>.

I defined a function named Detection_Protocol allows us to detect the NMEA frame that we want to store.



```
60 /* USER CODE BEGIN RV */
61
62 /* USER CODE END RV */
63
64 /* Private variables
65 -----*/
66
67 UART_HandleTypeDef huart1;
68
69 /* USER CODE BEGIN RV */
70
71 uint8_t Data_GPS; //Data GPS pointer
72 uint8_t Data_Buff; //Data sent buffer pointer
73 uint8_t Struct_Data_GPS[100]; //Data GPS Structure
74 uint8_t Struct_Data_GPS[100]; //Data GPS Structure
75 uint8_t TxIndex; // Tx index
76 uint8_t RxIndex; // Rx index
77 uint8_t GPS_State; //GPS state
78 uint8_t Protocol; //variable used to the protocol detection
79 uint8_t Start_Frame; //variable to find the start of the frame
80 uint8_t End_Frame; //variable to find the end of the frame
81 uint8_t End_Trans; //variable to tag the end of transmission
82 uint8_t Button_State; //variable to know the state of the User Button
83
84 /* USER CODE END RV */
85
86
87 /* Private function prototypes
88 -----*/
89 void SystemClock_Config(void);
90 static void MX_GPIO_Init(void);
91
92 -----
93 void Detection_Protocol(void);
94
95 /* USER CODE BEGIN PV */
96
97 /* USER CODE END PV */
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
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 9: Declaration of all the variables and of the function Detection Protocol

In the code below, I write a part that show you how realize a waiting state driven by a press on the User_Button of the board. This step is not necessary here but it is useful to know how realize it for other application. In order to verify that this step works I switch on a board led.

Note that if the GPS board fixes the satellites before you press the button then the application will begin then even.

When a PPS comes from the GPS board, an external interrupt is generated and the μ controller jumps at the interrupt routine that you can see in the figure below.

```

187 /* USER CODE END SysTick_IRQn 0 */
188 HAL_SysTick_Init();
189 /* USER CODE BEGIN SysTick_IRQn 1 */
190 /* USER CODE END SysTick_IRQn 1 */
191
192
193
194
195
196
197
198
199
200
201
202
203
204 void EXTI9_10_IRQHandler(void)
205 {
206     /* The PPS signal was set so the IT was generated
207     * USER CODE BEGIN EXTI9_10_IRQn 0 */
208     HAL_UART_Receive_IT(&uart4, (uint8_t *) &data_gps, 1); //start the UART receive function
209     BSP_LED_On(LED1); //switch on led1
210     /* USER CODE END EXTI9_10_IRQn 0 */
211     HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_9);
212     /* USER CODE BEGIN EXTI9_10_IRQn 1 */
213     /* USER CODE END EXTI9_10_IRQn 1 */
214 }
215
216
217
218
219
220 void UART4_IRQHandler(void)
221 {
222     /* USER CODE BEGIN UART4_IRQn 0 */
223     /* USER CODE END UART4_IRQn 0 */
224     HAL_UART_IRQHandler(&uart4);
225     /* USER CODE BEGIN UART4_IRQn 1 */
226     /* USER CODE END UART4_IRQn 1 */
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 10: Interrupt routine executed when PPS signal occurs

In this function, I start the UART to receive the GPS data and I switch on the led1 of the board (switch on the led allows knowing if the interrupt routine has been executed).

When a data is received in the UART, an interrupt is generated and then Callback function is called. This callback function first verifies if the UART that generated the interrupt is the uart4 uses by the GPS.

```

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 11: Reception callback function of the UART

Then this function tries to find the start of the frame; for that we test if the first element is a '\$' (you can see also that I switch on the led2; it's just for debugging). If we find this tag then we know that it is the start of the frame and we can store all the GPS frame elements until we find the '\n' element. This one specifies that this is the end of the frame (see the datasheet module). When the end of the frame is found (line 399 in the main.c file) then we call the Detection_Protocol function in order to know if the frame

store is the one that we want to decode. After that, we clear all the elements of the data GPS structure in order to store the next valid frame from the GPS. This step is not essential but we are sure that the data structure has no old value inside. At the end we restart the UART receive function; if you do not do this you will receive only one byte from the UART.

```

406 ReIndex=0; //reset the ReIndex
407
408 Start_Frame='F'; //set the Start Frame variable to 'F' (False)
409 }
410
411 HAL_UART_Receive_IT(&huart1, (uint8_t *) &Data_GPS, 1); //restart the UART receive function
412
413 }
414
415 /* Function to know the NMEA protocol used*/
416 uint8_t Detection_Protocol(void)
417 {
418     for (uint8_t i=0;i<5;i++) //Only the first 5 structure values are needed to know the protocol used
419     {
420         if(Struct_Data_GPS[i+1]== '$' && Struct_Data_GPS[i+2]=='P' && Struct_Data_GPS[i+3]=='R' && Struct
421             Protocol = 'P'; //protocol found
422         else
423             Protocol = 'F'; //protocol not found
424     }
425     return(Protocol); //return the protocol variable
426 }
427
428 /* USER CODE END 4 */
429
430
431 /**
432  * Brief description of the error occurrence.
433  * Return None
434  */
435 void Error_Handler(void)
436 {
437     /* USER CODE BEGIN Error_Handler_Debug */
438     /* User can add his own implementation to report the HAL error return state */
439     /* USER CODE END Error_Handler_Debug */
440 }
441
442
443 #ifndef USE_FULL_ASSERT
444 #define
445     "Error: Please report the name of the source file and the source line number
446     where the error_panic occurs (see comments)
447     = Source file address to the source file name
448     = Source line address panic error line source number
449     = Source name
450     = Source name
451     void assert_failed(uint8_t *file, uint32_t line)
452
453 #endif

```

Build Output

```

Build Time Elapsed: 00:00:03
Load: "UART1_UART.exe"
Erases Done.
Programming Done.
Verify: OK.
Flash Load finished at 13:30:19

```

Figure 12: Detection protocol function

In order to analyze only one NMEA frame between all of them, we must determine what is the type of the frame stored. To do this, we just read the first 5 elements of the GPS data structure since all the frame type can be decoded from these elements. The first one is always the '\$' that tags the start of the frame. Here I directly code the frame type that I want to analyze but you can define this parameter in an include file (this would facilitate the use of this function). You can see that I return only a variable to know if the protocol is the good one (Protocol variable is true 'V' or false 'F'). I have no use this information after to realize some actions but you can do it if you want to complete this code example.

At the end we have to compile the code in order to obtain the executable file for the STM32F7. When the compilation is done without error, we can transfer the code into the board. A demo video can be found here: https://youtu.be/R5l7x_rHsQc