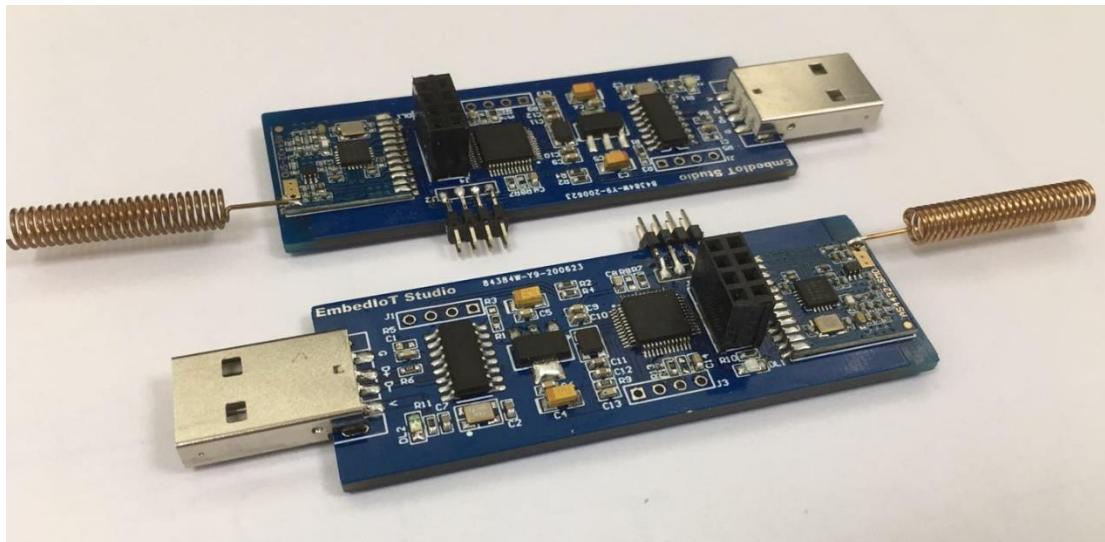


嵌入式物联网应用开发 – ARM 中控扩展模块

为了让 ARM-Linux 开发板具有 2.4GHz 和 433MHz 无线通信功能，微联智控工作室设计了一款通用的 ARM 中控扩展模块，该模块的实物图片，如下图所示。



这款 ARM 中控扩展模块，主要使用了 USB-TTL 芯片和 STM32 单片机进行通信，再通过 SPI 接口外接 si4432 模块和 nRF24L01+PA+LAN 模块。使用这种方式，就可以避免移植 si4432 和 nRF24L01 的驱动到 ARM-Linux 开发板，大大降低了开发工作量和开发难度。

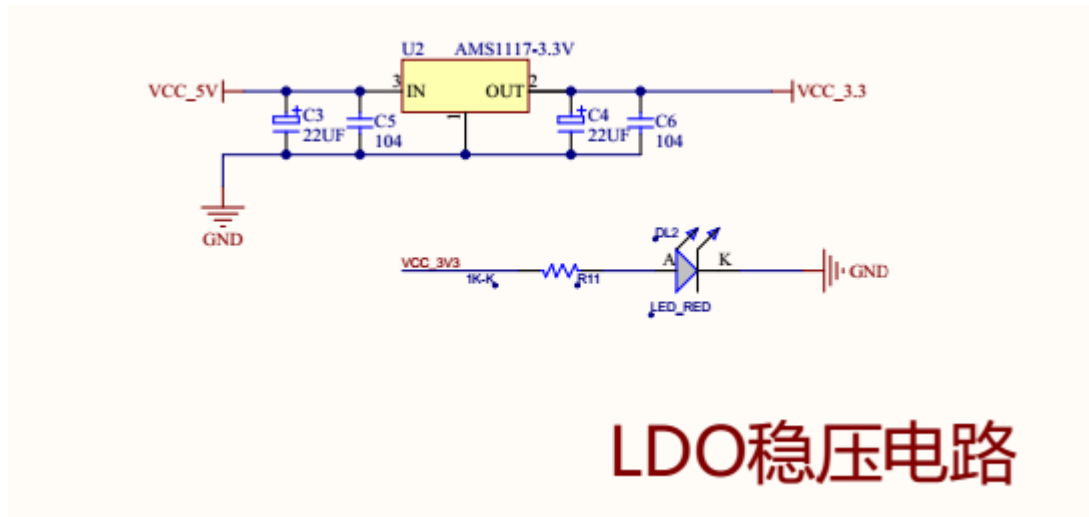
在 ARM-Linux 开发板或者在 PC 电脑上使用这款模块，只需要简单地在开发板移植或在 PC 机上安装 CH340G 的芯片驱动，就可以把这款模块当做串口使用。

ARM-Linux 中控扩展模块具有以下功能：

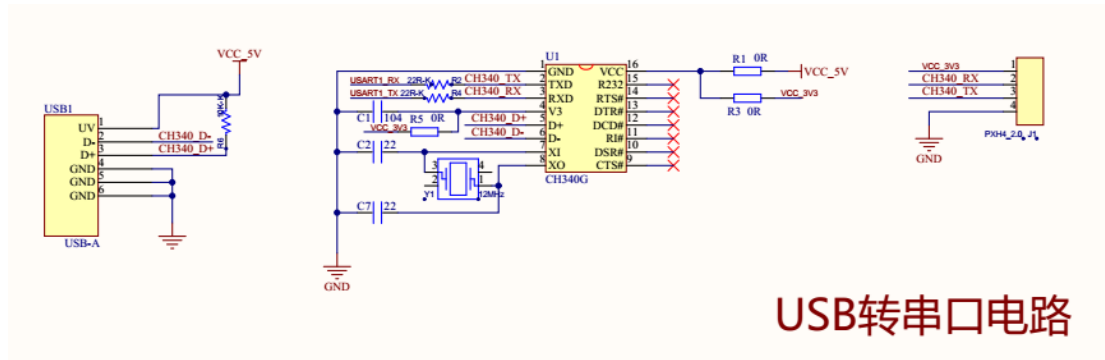
- 1、USB 转串口功能，使用 CH340G 芯片，可以方便对接电脑和嵌入式 Linux 开发板。
- 2、引出 CH340G 的 TTL 接口，可以用作一个普通的 USB 转串口模块（去掉 R2 和 R4 电阻）。
- 3、引出单片机的 UART2 串口，TTL 电平，方便用户扩展（提供源码）。
- 4、板载 433MHz 无线通信模块，与单片机通过 SPI 接口进行通信，用户可编程。
- 5、引出 SPI 接口，对接市面上大部分 SPI 接口的 2.4GHz 通信模块，用户可编程。
- 6、模块尺寸（不含 USB 接口）：70mm * 25mm

硬件设计篇

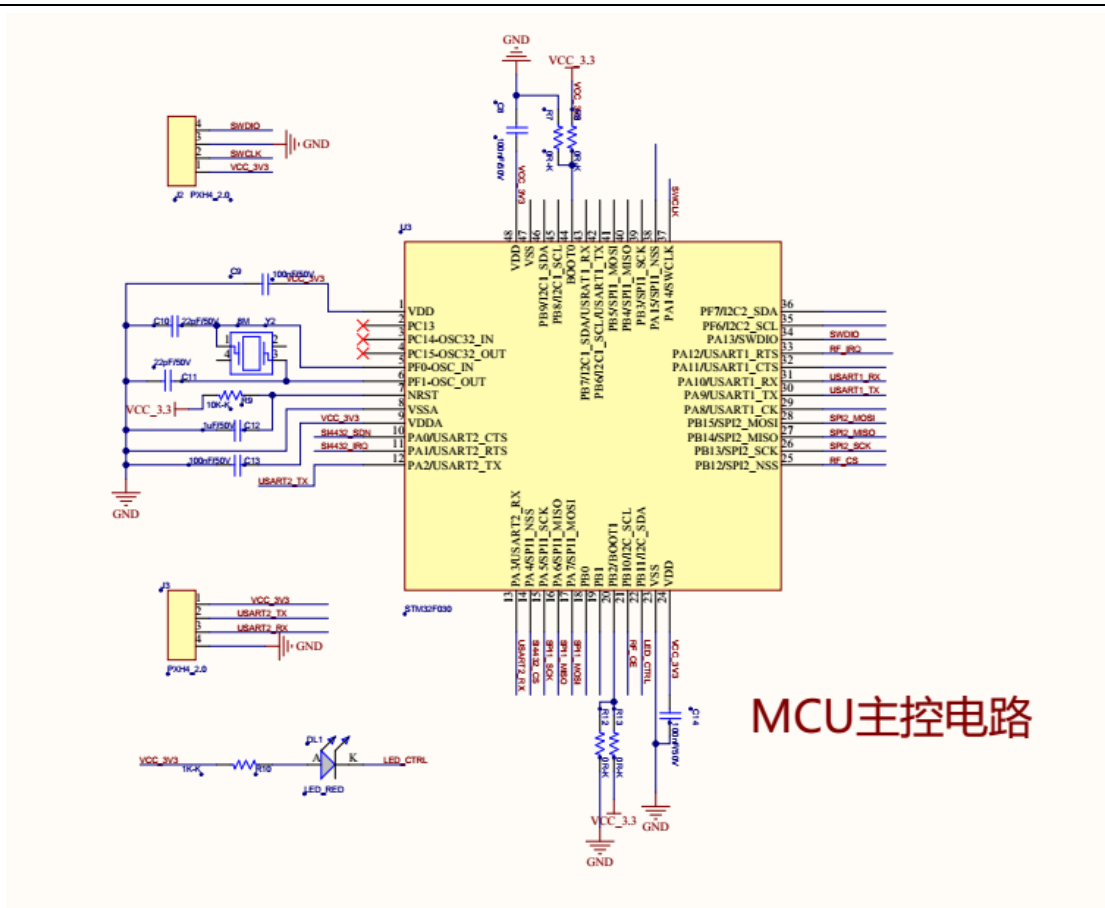
ARM 中控扩展模块使用 USB 接口进行 5V 供电，模块自带稳压芯片，5V 电源通过 LDO 稳压电路，为单片机及其他模块提供 3.3V 电源，模块还自带有红色的电源指示灯用来表示模块的通电情况，如下图所示。



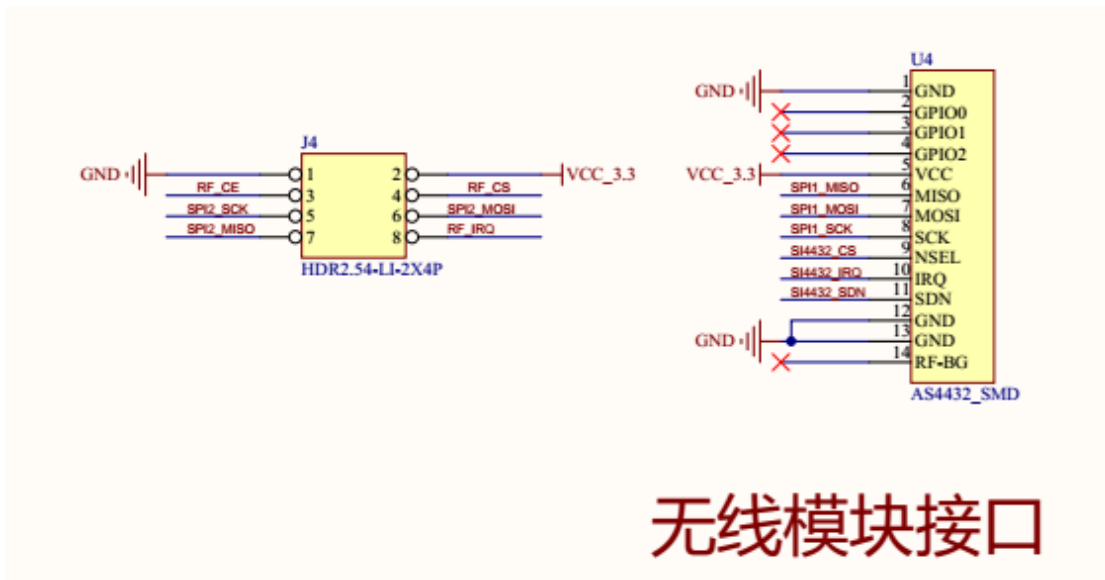
USB 转串口电路使用沁恒科技的 CH340G 作为转换芯片，该芯片可以通过 3.3V 或 5V 电源供电，对外引出芯片的 TTL 接口（使用时需要去掉 R2 和 R4 电阻），方便开发者扩展，USB 转串口电路如下图所示。



ARM 中控扩展模块采用 STM32F030C8T6 作为主控芯片，这款 MCU 芯片是 Cortex-M0 内核，主频 48MHz，芯片自带 64KB 可编程 Flash 和 8KB RAM，很适合物联网终端设备的控制器开发。模块的 MCU 控制电路如下图所示。



ARM 中控扩展模块使用 SPI1 和 SPI2 接口跟无线模块进行通信，其中，SPI1 接口连接 SI4432 这款 433MHz 无线模块，SI4432 模块使用焊接方式固定在中控扩展模块上。SPI2 接口通过提供接口的方式外引出来，可以用来对接 nRF24L01 这款 2.4GHz 无线模块，如下图所示。



以上就是 ARM 中控扩展模块的原理图描述，各个模块的原理比较简单，具体原理图和 PCB 的硬件工程可以到 GitHub 或 Gitee 上下载。

软件设计篇

ARM 中控扩展模块的软件框架，是基于任务和事件的 OSAL 调度器来进行开发，有关 OSAL 调度器的介绍，可以查看以下文章。

[开源 | 嵌入式物联网应用开发 - 基于任务和事件的 OSAL 调度器](#)

ARM 中控扩展模块，具有串口通信功能，2.4G 和 433 无线通信功能，看门狗系统监测功能，LED 指示灯功能。因此，在软件设计上，把这些功能划分为 6 个任务，在 `osal.h` 中定义好各个任务的 ID，并创建各个任务的源代码文件，如下图所示。

```

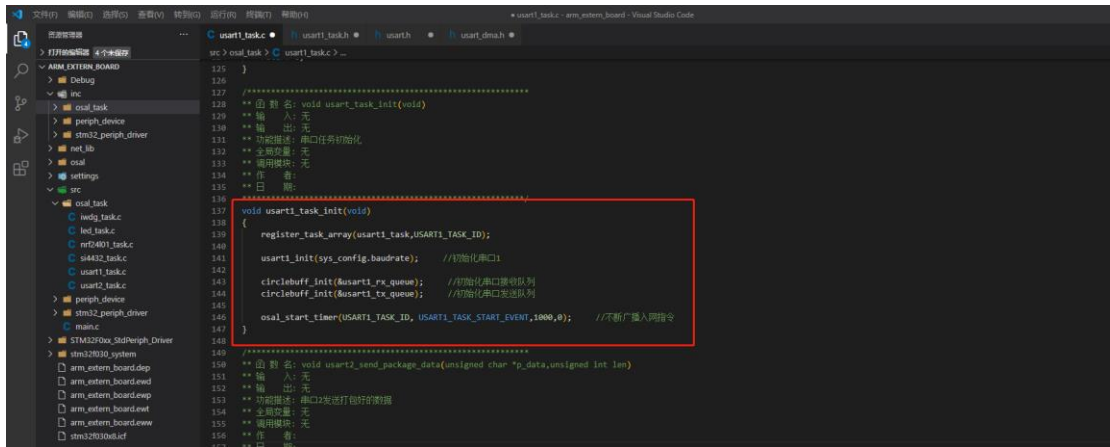
1  #ifndef __OSAL_H__
2  #define __OSAL_H__
3
4  #define TASK_NO_TASK      0x00
5
6  #define TASKSCNT  10
7
8  struct
9  {
10     #ifdef USART1_ON
11         USART1_TASK_ID,           //串口1任务
12     #endif
13     #ifdef USART2_ON
14         USART2_TASK_ID,           //串口2任务
15     #endif
16     RF24181_TASK_ID,             //RF24181任务
17     SI4432_TASK_ID,             //SI4432任务
18     WDG_TASK_ID,                //看门狗任务
19     LED_TASK_ID,                //LED指示灯任务
20 };
21
22 typedef unsigned short (*taskEventHandler)( unsigned char task_id, unsigned short event );
23
24 //=====
25 ** 函数名: unsigned char osal_set_event(unsigned char task_id, unsigned short event_flag)
26 ** 输入: task_id-任务id, event_flag-事件标志
27 ** 输出: 无
28 ** 功能描述: 立即事件设置函数
29 ** 返回值: 无
30 ** 调用模块: 无
31 ** 注意: 无
32 ** 日期:
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
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
    
```

在 `main.c` 主函数里面，进行 OSAL 调度器和各个任务事件的初始化，由于有些设备不一定用到所有串口，因此，通过 `USART1_ON` 和 `USART2_ON` 这两个宏定义作为串口模块的开关，主函数的内容如下图所示。

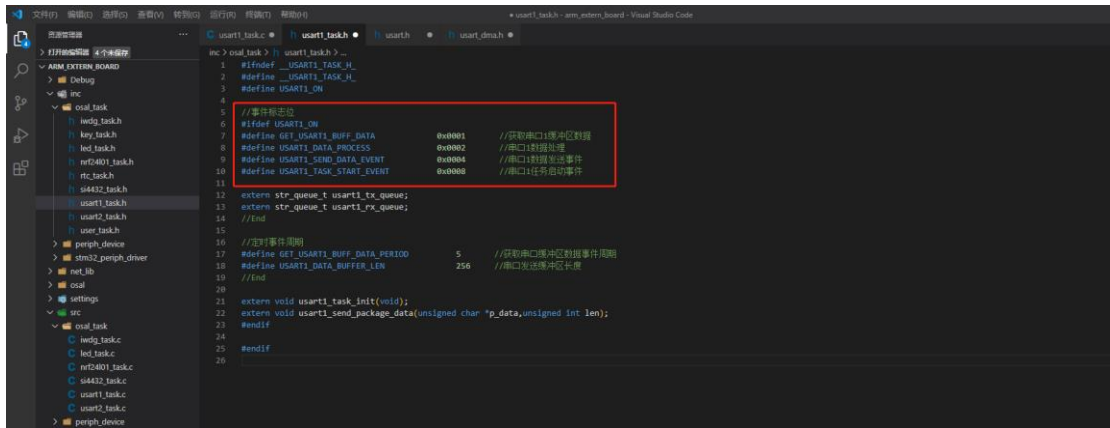
```

1  #include "main.h"
2
3
4
5
6
7
8  void main(void)
9  {
10     systick_init();           //初始化滴答时钟
11     osal_timer_init();        //初始化OSAL定时器
12     check_sys_config();       //检测系统配置
13
14     #ifdef USART1_ON
15         usart1_task_init();    //串口1任务初始化
16     #endif
17     #ifdef USART2_ON
18         usart2_task_init();    //串口2任务初始化
19     #endif
20     rf24181_task_init();       //rf24181任务初始化
21     si4432_task_init();        //SI4432任务初始化
22     wdg_task_init();           //看门狗任务初始化
23     led_task_init();           //led任务初始化
24
25     while(1)
26     {
27         run_system();
28     }
29 }
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
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
    
```

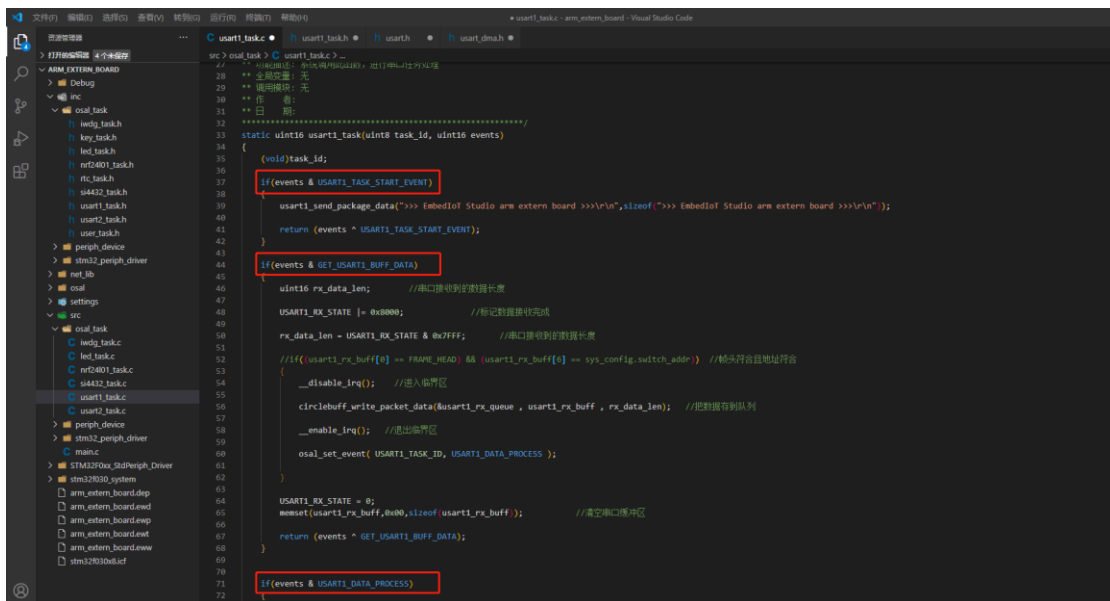
先来看看串口任务处理，ARM 中控扩展模块主要用到了串口 1 以及 DMA 通信。所以，需要在工程的宏定义那里定义 `USART1_ON` 宏和 `USART_DMA` 宏，以使能串口 1 的 DMA 通信方式。在源文件 `usart1_task.c` 中，对串口任务进行了注册和串口外设初始化，并且同时初始化了两个队列数组，用来缓冲串口的发送数据和接收数据，如下图所示。



在头文件 usart1_task.h 中，定义串口 1 任务相关的事件，主要有串口 1 数据收发和处理，串口任务启动事件，如下图所示。



然后在源文件 usart1_task.c 的 usart1_task()函数中，处理各种串口任务事件，OSAL 调度器在收到串口的各种事件后，都会在这个函数里面处理各种事件，如下图所示。



(限于篇幅，这里只列出部分源码)

对于 SI4432 和 nRF24L01 这两款无线模块的任务处理，其任务事件的处理框架都是大同小异的，这里以 SI4432 为例进行说明。

先来看看源文件 si4432_task.c 里面的任务初始化函数，如下图所示。

```

137
138 //*****
139 ** 函 数 名: void si4432_task_init(void)
140 ** 输 入 无
141 ** 输 出: 无
142 ** 功 能 描 述: si4432任务初始化
143 ** 参 数 描 述: 无
144 ** 调用模块: 无
145 ** 注 意:
146 ** 日 期:
147 //*****
148 void si4432_task_init(void)
149 {
150     register_task_array(si4432_task,SI4432_TASK_ID);
151
152     si4432_init(); //si4432 芯片初始化
153
154     circInbuff_init(&si4432_rx_queue); //初始化si4432接收队列
155     circOutbuff_init(&si4432_tx_queue); //初始化si4432发送队列
156
157     si4432_convert_channel(sys_config.channel - 1); //设置当前信道
158
159
160
    
```

在初始化函数 si4432_task_init()中，先对 si4432 芯片的连接引脚和芯片寄存器进行初始化，然后再初始化 si4432 的数据发送和接收队列，然后再根据系统设置的参数，进行无线通信信道设置。

在头文件 si4432_task.h 中，定义了各种任务事件，主要有数据发送和接收，超时处理，以及中断事件，如下图所示。

```

1
2 #ifndef __SI4432_TASK_H_
3 #define __SI4432_TASK_H_
4 #define SI4432_SEND_DATA_EVENT 0x0001 //数据发送事件
5 #define SI4432_SEND_TIMEOUT_EVENT 0x0002 //发送超时事件
6 #define SI4432_RECV_DATA_EVENT 0x0004 //数据接收事件
7 #define SI4432_IRQ_PROCESS_EVENT 0x0008 //中断处理事件
8
9 extern str_queue_t si4432_rx_queue; //si4432接收队列
10 extern str_queue_t si4432_tx_queue; //si4432发送队列
11 extern uint32_t tx_fifo16k_flags;
12 extern uint16_t current_brightness;
13
14 extern unsigned char si4432_send_data(unsigned char *p_data,unsigned int len);
15 extern void si4432_task_init(void);
16
17 #endif
18
19
20
    
```

各种任务事件是在 si4432_task()这个函数中进行处理的，对于 SI4432 无线模块接收到的数据，目前都是通过串口透传出去，交给上位机进行处理，各个任务事件的函数内容，如下图所示。

```

static void terminal_si4432_data_process(unsigned char *p_data, unsigned int datalen)
{
    usart1_send_data(p_data, datalen); //串口发送数据
}

//*****
** 函数名: ui4432_irq_task(ui4432_task_id, uint32 events)
** 输入: task_id-任务id, event-事件id
** 输出: events-正在处理的事件
** 功能描述: 驱动网络设备的, 进行si4432任务处理
** 全局变量: 无
** 调用模块: 无
** 作者:
** 日期:
*****
static uint16_t si4432_irq_task(uint8_t task_id, uint32_t events)
{
    (void)task_id;

    if (events & SI4432_IRQ_PROCESS_EVENT)
    {
        si4432_irq_process(); //si4432中断处理函数
        return (events ^ SI4432_IRQ_PROCESS_EVENT);
    }

    if (events & SI4432_SEND_TIMEOUT_EVENT) //数据发送超时事件
    {
        si4432_init(); //重新初始化设备
        events ^= SI4432_SEND_DATA_EVENT;
        return (events ^ SI4432_SEND_TIMEOUT_EVENT);
    }

    if (events & SI4432_RECV_DATA_EVENT) //数据接收事件
    {
        if (si4432_rx_queue.sem_value) //接收队列不为空
        {
            unsigned int datalen = 0;
            uint8_t rx_buff[SI4432_RX_BUFFER_LEN];

            datalen = circlebuff_read_packet_data(&si4432_rx_queue, rx_buff); //从队列读取数据
            terminal_si4432_data_process(rx_buff, datalen); //处理si4432接收的数据
        }
    }
}
    
```

(限于篇幅, 这里只列出部分源码)

由硬件原理图可知, 不管是 SI4432 或 nRF24L01 模块, 一旦接收到无线数据, 都是通过引脚中断的方式来通知 STM32 进行处理的, 因此, 需要在 STM32 的外部中断引脚处理函数中, 取出无线模块接收到的数据, 如下图所示。

```

//外部中断处理函数
void EXTI0_IRQ_Handler(void)
{
    if (EXTI_GetITStatus(SI4432_EXTI_LINE) != RESET) //si4432中断
    {
        si4432_irq_process();
        //osal_set_event(SI4432_TASK_ID, SI4432_IRQ_PROCESS_EVENT); //中断处理事件
        EXTI_ClearITPendingBit(SI4432_EXTI_LINE);
    }
}

//外部中断处理函数
void EXTI15_IRQ_Handler(void)
{
    if (EXTI_GetITStatus(NRF24L01_EXTI_LINE) != RESET) //nrf24l01中断
    {
        nrf24l01_irq_process();
        //osal_set_event(NRF24L01_TASK_ID, NRF24L01_IRQ_PROCESS_EVENT); //中断处理事件
        EXTI_ClearITPendingBit(NRF24L01_EXTI_LINE);
    }
}
    
```

看门狗任务是用来监测整个系统的运行情况的, 一旦发现系统运行异常, 看门狗由于喂狗不及时, 系统就会进行超时重启。因此, 看门狗任务比较简单, 主要是初始化 STM32 的看门狗外设, 并设置一个定时喂狗任务, 定时进行看门狗计数器复位操作, 防止系统复位。与此同时, 也可以利用看门狗超时的特性, 来定义一个复位任务, 方便一些系统重启操作, 如下图所示。

```

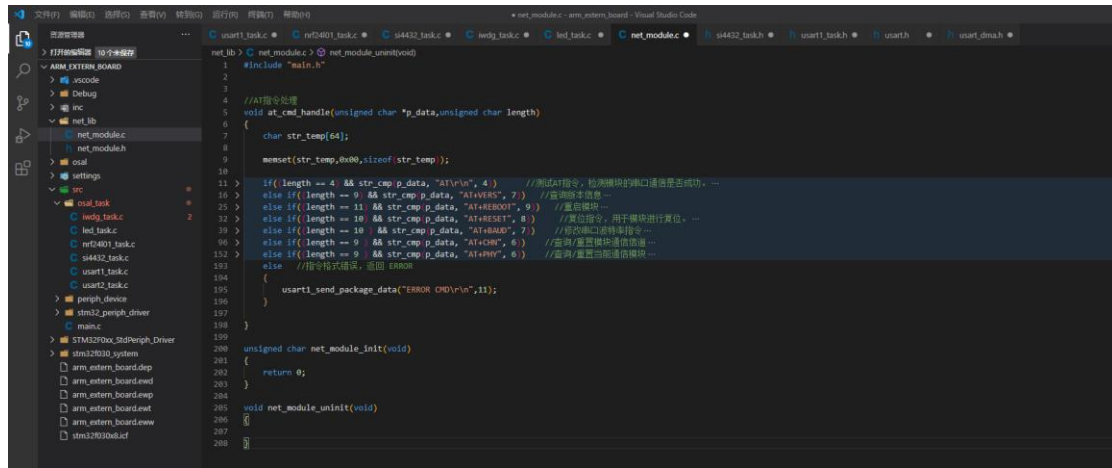
38
39 ** 函数名: uint16 imdg_task(uint8 task_id, uint16 events)
40 ** 输入: task_id-任务id, event-事件id
41 ** 输出: events-未处理的事件
42 ** 功能描述: 看门狗任务处理
43 ** 全局变量: 无
44 ** 调用顺序: 无
45 ** 作者:
46 ** 日期:
47
48 static uint16 imdg_task(uint8 task_id, uint16 events)
49 {
50     (void)task_id;
51
52     if (events & IMDG_FEED_EVENT)
53     {
54         imdg_feed(); //定时喂狗
55         return (events ^ IMDG_FEED_EVENT);
56     }
57
58     if (events & IMDG_RESET_EVENT)
59     {
60         while(1){delay_ms(10);}
61     }
62
63     return 0;
64 }
65
66 //*****
67 ** 函数名: void imdg_task_init(void)
68 ** 输入: 无
69 ** 输出: 无
70 ** 功能描述: 看门狗任务初始化
71 ** 全局变量: 无
72 ** 调用顺序: 无
73 ** 作者:
74 ** 日期:
75
76 void imdg_task_init(void)
77 {
78     register_task_array(imdg_task,IMDG_TASK_ID);
79
80     imdg_init(IMDG_Prescaler_65,645);
81     osal_start_timer(IMDG_TASK_ID, IMDG_FEED_EVENT, IMDG_FEED_PERIOD, IMDG_FEED_PERIOD);
82 }
83
    
```

指示灯任务是用来指示系统无线网络状态的，当无线网络处于连接成功/连接断开/数据广播等不同状态的时候，指示灯会以不同的频率进行闪烁。网络状态指示灯的任务处理，如下图所示。

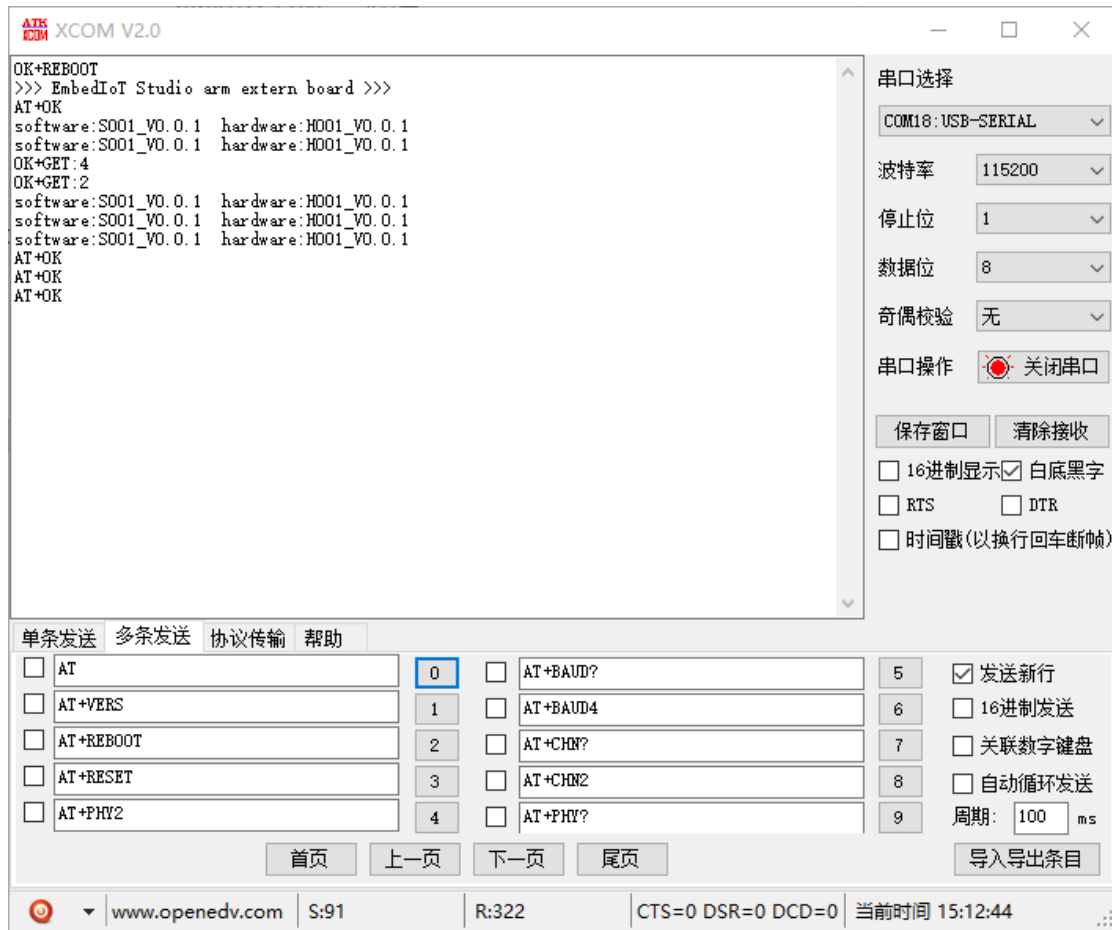
```

17
18 static uint16 led_task(uint8 task_id, uint16 events)
19 {
20     (void)task_id;
21
22     if (events & LED_TASK_START_EVENT)
23     {
24         hal_led_set(LED_RED, HAL_LED_MODE_ON); //开机上电时,红灯亮
25         return (events ^ LED_TASK_START_EVENT);
26     }
27
28     if (events & LED_TASK_BLINK_EVENT) //led闪烁事件
29     {
30         hal_led_update(); //更新led信号
31         return (events ^ LED_TASK_BLINK_EVENT);
32     }
33
34     if (events & RED_LED_TOGGLE_EVENT) //红色LED翻转事件
35     {
36         hal_led_set(LED_RED, HAL_LED_MODE_TOGGLE);
37         return (events ^ RED_LED_TOGGLE_EVENT);
38     }
39
40     return 0;
41 }
42
43 //*****
44 ** 函数名: void led_task_init(void)
45 ** 输入: 无
46 ** 输出: 无
47 ** 功能描述: led任务初始化
48 ** 全局变量: 无
49 ** 调用顺序: 无
50 ** 作者:
51 ** 日期:
52
53 void led_task_init(void)
54 {
55     register_task_array(led_task,LED_TASK_ID);
56
57     led_gpio_init();
58
59     hal_led_set(LED_RED, HAL_LED_MODE_OFF); //初始化时,红灯灭
60
61     osal_start_timer(LED_TASK_ID, RED_LED_TOGGLE_EVENT, 500,500);
62 }
63
    
```

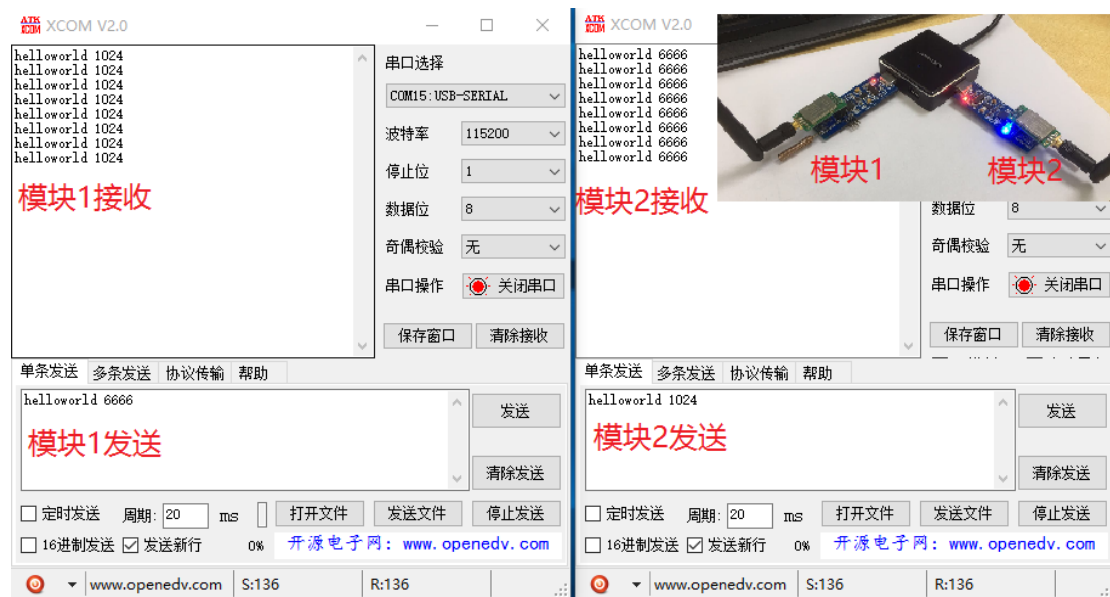
为了方便开发者对 ARM 中控扩展模块进行设置，模块还支持 AT 指令扩展，开发者可以通过 AT 指令对模块进行波特率参数，信道，恢复出厂设置，查询软硬件版本信息等操作，也可以根据实际需要，修改源码扩展自己的 AT 指令集，目前模块支持的 AT 指令如下图所示。



以上就是 ARM 中控模块的软件开发内容，编译源码并下载到模块中运行，可以使用串口工具测试模块的 AT 指令是否工作正常，如下图所示。



使用两个 ARM 中控扩展模块，并同时接到 PC 电脑上，用串口调试工具可以在这两个模块直接互相透传数据，如下图所示。



项目的开源地址:

<https://github.com/embediot/Embedded-IoT-Project>

<https://gitee.com/embediot/Embedded-IoT-Project>

感谢阅读!

[点击这里，访问作者博客](#)

欢迎关注【微联智控工作室】

