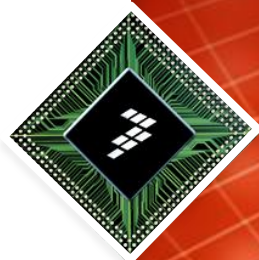
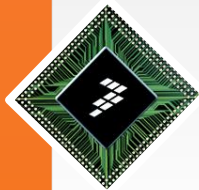




USB CDC Host application

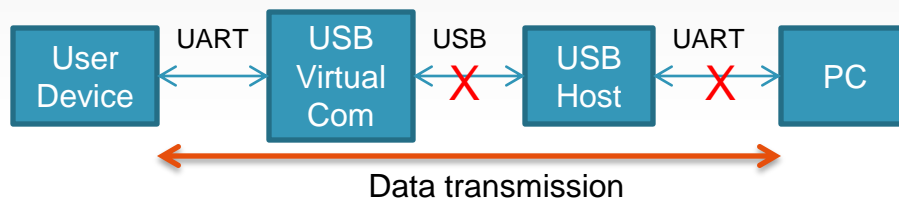
Issue : USB Host not working

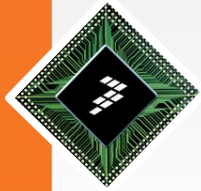




Issue

- Stack revision – 4.0.3
- State
 - Virtual Com USB Device detectable .
 - **Data transfer not working .**
- Expect
 - PC transfer data to Embedded USB Host Device by UART then the data passed to USB Virtual Com Device by embedded USB Host .
 - USB Virtual Com Device transfer data to embedded USB Host then the data passed to PC by UART of embedded USB Host Device .
- **Issue**
 - **USB Host can't transfer data to Virtual Com device from UART**
 - **There is no response when Virtual Com transfer data to USB Host**





Experiment -- Environment

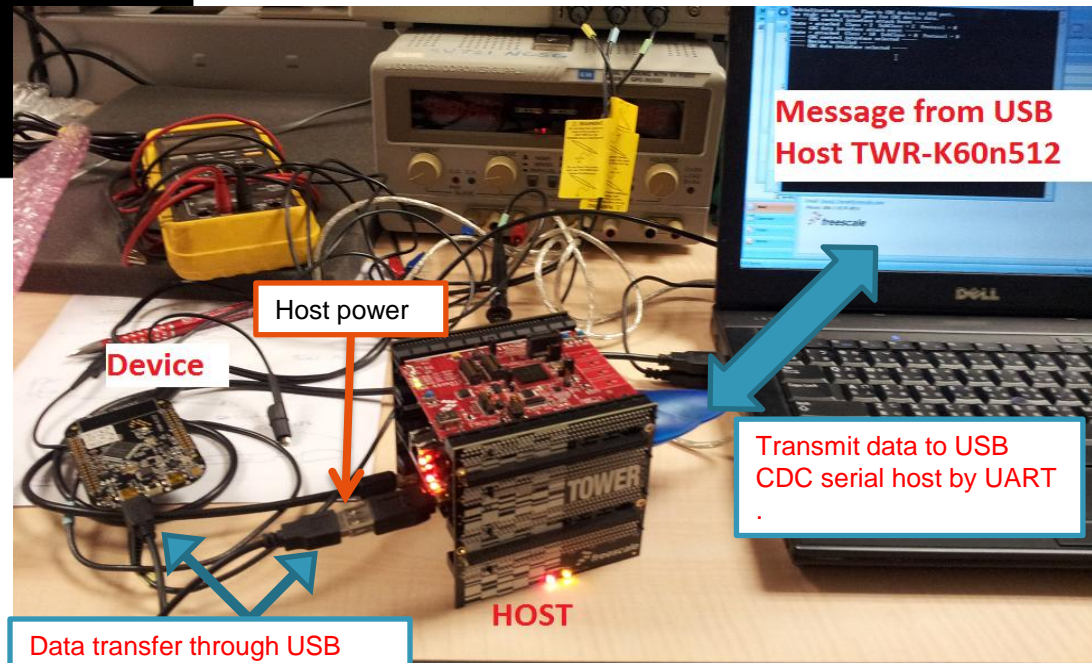
- USB CDC Host :
 - EVB – TWR-K60n512
 - Code -- C:\Freescale\Freescale USB Stack v4.0.3\Source\Host\examples\cdc_serial\cw10\kinetis_k60
- USB virtual Com Device :
 - EVB – FRDM-KL25Z
 - Code -- C:\Freescale\Freescale USB Stack v4.1.1\Source\Device\app\cdc\cw10\kinetis_l2k

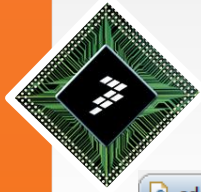
```
COM3:115200baud - Tera Term VT
File Edit Setup Control Window Help

Initialization passed. Plug-in CDC device to USB port.
Use ttyb: as the in/out port for CDC device data.
-----
CDC control interface attach Event -----
State = attached Class = 2 SubClass = 2 Protocol = 0
CDC data interface attach event -----
State = attached Class = 10 SubClass = 0 Protocol = 0
-----
CDC control interface selected -----
Device installed -----
-----
CDC data interface selected -----
```

But no response when we key in any char

Device detected





USB Host information declaration

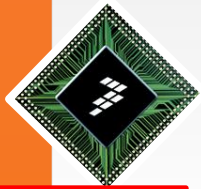
```
cdc_serial.c | sci.c | poll.c | khci_kinetis.c | usb_host_cdc.c | usbms
81 void CDC_Task();
82 extern void _usb_khci_task(void);
83
84 static const USB_HOST_DRIVER_INFO DriverInfoTable[] = {
85     {
86         {0x00,0x00}, /* Vendor ID per USB-IF */
87         {0x00,0x00}, /* Product ID per manufacturer */
88         USB_CLASS_COMMUNICATION, /* Class code */
89         USB_SUBCLASS_COM_ABSTRACT, /* Sub-Class code */
90         0xFF, /* Protocol */
91         0, /* Reserved */
92         usb_host_cdc_acm_event /* Application call back function */
93     },
94     {
95         {0x00,0x00}, /* Vendor ID per USB-IF */
96         {0x00,0x00}, /* Product ID per manufacturer */
97         USB_CLASS_DATA, /* Class code */
98         0xFF, /* Sub-Class code */
99         0xFF, /* Protocol */
100        0, /* Reserved */
101        usb_host_cdc_data_event /* Application call back function */
102    },
103    /* USB 1.1 hub */
104    {
105
106        {0x00,0x00}, /* Vendor ID per USB-IF */
107        {0x00,0x00}, /* Product ID per manufacturer */
108        USB_CLASS_HUB, /* Class code */
109        USB_SUBCLASS_HUB_NONE, /* Sub-Class code */
110        USB_PROTOCOL_HUB_LS, /* Protocol */
111        0, /* Reserved */
112        usb_host_hub_device_event /* Application call back function */
113    },
114    {
115        {0x00,0x00}, /* All-zero entry terminates */
116        {0x00,0x00}, /* driver info list. */
117        0,
118        0,
119        0,
120        0,
121        NULL
122    }
123};
```

• Detail please refer to USBHOSTUG.pdf Chapter 4.2.2 [Define a driver info table].

/* Information for one class or device driver */

typedef struct driver_info

```
{
    uint_8 idVendor[2]; /* Vendor ID per USB-IF */
    uint_8 idProduct[2]; /* Product ID per manufacturer */
    uint_8 bDeviceClass; /* Class code, 0xFF if any */
    uint_8 bDeviceSubClass; /* Sub-Class code, 0xFF if any */
    uint_8 bDeviceProtocol; /* Protocol, 0xFF if any */
    uint_8 reserved; /* Alignm
    event_callback attach_call; /* event callback function*/
*/ } USB_HOST_DRIVER_INFO, PTR_USB_HOST_DRIVER_INFO_PTR; ent padding */
```



Initial USB Host

```

cdc_serial.c sci.c poll.c khci_kinetis.c
129 /*FUNCTION*-----
130 *
131 * Function Name : Main
132 * Returned Value : none
133 * Comments :
134 *
135 *
136 *END*-----
137 #ifdef __GNUC__
138 int main(void)
139 #else
140 void main(void)
141 #endif
142 {
143     USB_STATUS status = USB_OK;
144     _usb_host_handle host_handle;
145
146     /* Initialize the current platform. Call for the
147     _bsp_platform_init();
148     #ifdef MCU_MK70F12
149     sci2_init();
150     #else
151     sci1_init();
152     #endif
153     TimerInit();
154
155     /* Init polling global variable */
156     POLL_init();
157
158     DisableInterrupts;
159     #if (defined_MCF51MM256_H) || (defined_MCF51J1256_H)
160     usb_int_dis();
161     #endif

```

Peripheral initial

```

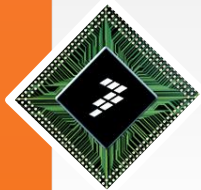
162 /*
163 ** It means that we are going to act like host, so we initialize the
164 ** host stack. This call will allow USB system to allocate memory for
165 ** data structures, it uses later (e.g pipes etc.).
166 */
167 status = _usb_host_init (
168     HOST_CONTROLLER_NUMBER, /* Use value in header file */
169     MAX_FRAME_SIZE, /* Frame size per USB spec */
170     &host_handle); /* Returned pointer */
171 if (status != USB_OK)
172 {
173     printf("\nUSB Host Initialization failed. STATUS: %x", (unsigned int) status);
174     fflush(stdout);
175     exit(3);
176 }
177 status = _usb_host_driver_info_register (
178     host_handle,
179     (void *)DriverInfoTable);
180 if (status != USB_OK)
181 {
182     printf("\nDriver Registration failed. STATUS: %x", (unsigned int) status);
183     fflush(stdout);
184     exit(4);
185 }
186
187 EnableInterrupts;
188 #if (defined_MCF51MM256_H) || (defined_MCF51J1256_H)
189 usb_int_en();
190 #endif
191
192 printf("\fInitialization passed. Plug-in CDC device to USB port. \nUse ttyL
193
194 usb_event_init(&device_registered);
195
196 uart2usb_num = usb2uart_num = 0; /* reset number of bytes in buffers */
197
198 f_usb = (FILE_CDC_PTR)malloc(sizeof(FILE_CDC));
199 memset(f_usb, 0, sizeof(FILE_CDC));
200
201 f_usb->DEV_PTR = (IO_DEVICE_STRUCT_PTR)malloc(sizeof(IO_DEVICE_STRUCT));
202

```

USB Host initial

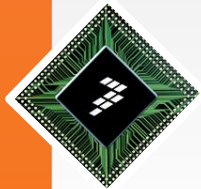
Register USB driver

Initial event



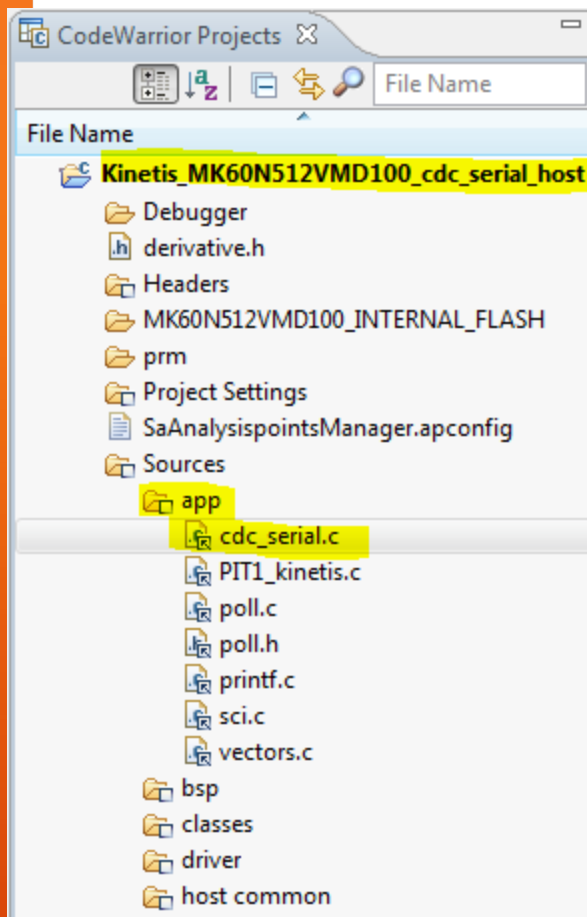
USB Host application function flow

- Please refer to USBHOSTUG.pdf
 - Chapter 4.2.3 [Main application function flow]
 - Initializing hardware
 - Initializing the host controller
 - Registering service
 - Calling tasks in a forever loop



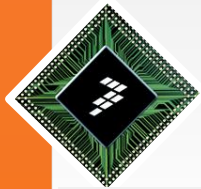
USB CDC Serial Host – main task

- Major task of USB CDC serial Host application



```
195 _usb_event_init(&device_registered);
196 uart2usb_num = usb2uart_num = 0; /* reset number of
197
198 f_usb = (FILE_CDC_PTR)malloc(sizeof(FILE_CDC));
199 memset(f_usb, 0, sizeof(FILE_CDC));
200
201 f_usb->DEV_PTR = (IO_DEVICE_STRUCT_PTR)malloc(sizeo
202
203
204 for(;;)
205 {
206     Poll();
207     CDC_Task();
208     __RESET_WATCHDOG(); /* feeds the dog */
209 } /* loop forever */
210 /* please make sure that you never leave main */
211 #ifdef __GNUC__
212     return 0;
213 #endif
214 }
215
216 /*FUNCTION*-----
217 *
218 * Function Name : CDC_Task
219 * Returned Value : none
220 * Comments      :
221 *     Execution starts here
222 *
```

```
mem_util.h
device_registered : USB_EV
reg_device : _usb_device_in
usb_open_param : const CD
f_usb : FILE_CDC_PTR
uart2usb : char[]
usb2uart : char[]
device_name : char_ptr
uart2usb_num : volatile int
usb2uart_num : volatile int
main_buffer : uchar*
num_done : volatile int_32
buff : char[]
buff_index : int_32
uart_coding : USB_CDC_UA
CDC_Task() : void
_usb_khci_task(void) : void
DriverInfoTable : const USB
check_open : uint_32
Read_USB_Data : uint_8
char_to_recv : uint_32
main(void) : void
CDC_Task() : void
usb_host_cdc_acm_event(
```



Major process of main task – Enumerate Device

```
cdc_serial.c x sci.c poll.c khci_kinetis.c usb_host_cdc.c usbmsgq.c virtual_com.c PIT1_kine
222 /*FUNCTION*-----
223 *
224 * Function Name : CDC_Task
225 * Returned Value : none
226 * Comments      :
227 * Execution starts here
228 *
229 *END*-----*/
230 void CDC_Task ()
231 { /* Body */
232     USB_STATUS      status = USB_OK;
233     uint_32         i = 0;
234     //static uint_32 char_to_recv = 0;
235
236     /* due to the fact that uart driver blocks task, we will check if char is available and then we read it */
237     /* write them to USB */
238     if(USB_EVENT_NOT_SET == _usb_event_wait_ticks(&device_registered, 0x01, TRUE, 0))
239         return;
240
241     if (0 == check_open)
242     {
243         _io_cdc_serial_open(f_usb, device_name, (char *)&usb_open_param);
244         check_open = 1;
245     }
246     else
247     {
248 #ifdef MCU_mcf51jf128
249         /* todo AI: change this for MCU_mcf51jf128 */
250         char temp;
251
252         temp = TERMIO_GetChar();
253         if(temp)
254         {
255             buff_index = 1;
256             buff[0] = temp;
257         }
258 #endif /* MCU_mcf51jf128 */
259
260         /* Read data from UART */

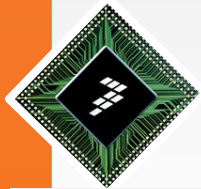
```

Waiting for USB Device attached

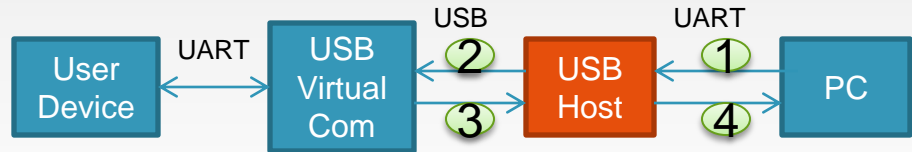
Open USB Device if not yet

Read / Write Data between USB and UART, after USB device attached





USB Data handler



```

240
241 if (0 == check_open)
242 {
243     _io_cdc_serial_open(f_usb, device_name, (char *)&usb_open_param);
244     check_open = 1;
245 }
246 else
247 {
248 |
249     DisableInterrupts;
250     uart2usb_num = buff_index;
251     for (i = 0; i < buff_index; i++)
252         uart2usb[i] = buff[i];
253     buff_index = 0;
254     EnableInterrupts;
255
256     /* Write data to USB */
257     if (uart2usb_num)
258     {
259         num_done = _io_cdc_serial_write(f_usb, uart2usb, (int_32)(sizeof(uart2usb[0]) * uart2usb_num));
260         if(num_done > 0)
261         {
262             for (i = (uint_32)num_done; i < uart2usb_num; i++) /* move buffer data, remove the written ones */
263                 uart2usb[i - num_done] = uart2usb[i];
264             uart2usb_num -= num_done;
265             char_to_recv += num_done;
266         }
267     }
268
269     /* Read data from USB */
270     if(char_to_recv > 0 || Read_USB_Data)
271     {
272         Read_USB_Data = 0;
273         num_done = _io_cdc_serial_read(f_usb, usb2uart + usb2uart_num, (int_32)(sizeof(uart2usb) / sizeof(uart2usb[0]) - usb2uart_num));
274         if(num_done > 0 && usb2uart[usb2uart_num] != 0)
275         {
276             usb2uart_num += num_done;
277             if(char_to_recv >= num_done)
278             {
279                 char to_recv -= num done;
280             }
281             else
282             {
283                 char_to_recv = 0;
284             }
285         }
286         else
287         {
288             char_to_recv = 0;
289         }
290     }
291
292     /* write them to UART */
293     if (usb2uart_num > 0)
294     {
295         for (i = 0; i < usb2uart_num; i++)
296         {
297             #ifdef MCU_mcf51j128
298                 printf(" --received-- %c\n\r",usb2uart[i]);
299             #else
300                 sci2_PutChar(usb2uart[i]);
301             #endif
302         }
303         usb2uart_num = 0;
304     }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 #ifdef MCU_mcf51j128
324     printf(" --received-- %c\n\r",usb2uart[i]);
325 #else
326     sci2_PutChar(usb2uart[i]);
327 #endif
328 #endif
329 }
330 }
331 }
332 }
333 }
334 } /* Endbody */
335

```

1

UART to USB buffer index handler

2

Write data to USB

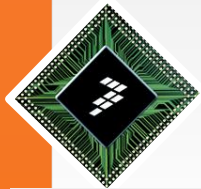
3

Read data from USB

4

Write data to UART

- UART to USB buffer
 - PC terminal data to USB CDC Host
- Write Data to USB
 - USB CDC Host transfer data to CDC Device
- Read data from USB
 - USB CDC Host receive data from CDC device
- Write data to UART
 - USB CDC Host transfer data to PC terminal



Debug 1– Can't receive data from PC terminal

Only handle buffer index but no one to receive data

```

240
241 if (0 == check_open)
242 {
243     _io_cdc_serial_open(f_usb, device_name, (char *)&usb_op
244     check_open = 1;
245 }
246 }
247 else
248 {
249     DisableInterrupts;
250     uart2usb_num = buff_index;
251     for (i = 0; i < buff_index; i++)
252         uart2usb[i] = buff[i];
253     buff_index = 0;
254     EnableInterrupts;
255 }
256
257 /* Write data to USB */
258 if (uart2usb_num)
259 {
260     num_done = _io_cdc_serial_write(f_usb, uart2usb, (int_32)(sizeof(uart2usb[0]) * uart2usb_num));
261     if(num_done > 0)
262     {
263         for (i = (uint_32)num_done; i < uart2usb_num; i++) /* move buffer data, remove the written ones */
264             uart2usb[i - num_done] = uart2usb[i];
265         uart2usb_num -= num_done;
266         char_to_rcv += num_done;
267     }
268 }
269
270 /* Read data from USB */
271 if(char_to_rcv > 0 || Read_USB_Data)
272 {
273     Read_USB_Data = 0 ;
274     num_done = _io_cdc_serial_read(f_usb, usb2uart + usb2uart_num, (int_32)(sizeof(uart2usb) / sizeof(uart2usb[0]) - usb2uart_num));
275     if(num_done > 0 && usb2uart[usb2uart_num] != 0 )
276     {
277         usb2uart_num += num_done;
278         if(char_to_rcv >= num_done)
279         {
280             char to rcv -= num done;
281         }
282         else
283         {
284             char_to_rcv = 0;
285         }
286     }
287     else
288     {
289         char_to_rcv = 0;
290     }
291 }
292
293 /* write them to UART */
294 if (usb2uart_num > 0)
295 {
296     for (i = 0; i < usb2uart_num; i++)
297     {
298 #ifdef MCU_mcf51j128
299         printf(" --received--> %c\n\r",usb2uart[i]);
300 #else
301         sci2_PutChar(usb2uart[i]);
302 #endif
303     }
304     usb2uart_num = 0;
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 }

```

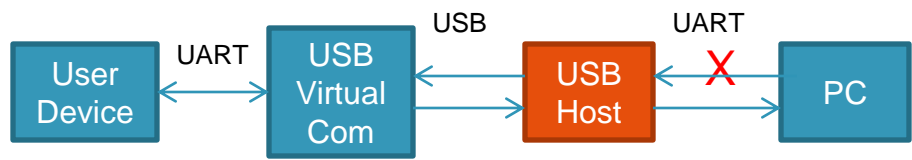
UART to USB buffer index handler

Write data to USB

Read data from USB

Write data to UART

- Issue :
 - Data NOT received from PC terminal
- Root cause :
 - No bubby to receive data from UART
- Workaround :
 - Call `uart_getchar()` to receive data
 - Detail pls refer to next slide .



Debug 1 -- Workaround

– Can't receive data from PC terminal

```
259
260 /* Read data from UART */
261 char temp;
262
263 temp = uart_getchar(); //for CodeWarrior
264 //temp = TERMIO_GetCharNB(); // for IAR
265 if(temp)
266 {
267     buff[0] = temp;
268     buff_index = 1;
269 }
270
271 DisableInterrupts;
272 uart2usb_num = buff_index;
273 for (i = 0; i < buff_index; i++)
274     uart2usb[i] = buff[i];
275 buff_index = 0;
276 EnableInterrupts;
```

Call `uart_getchat()` to receive data from PC terminal

```
*cdc_serial.c *sci.c x PIT1_kinetis.c host_ch9.c
200
201 #ifdef __CC_ARM
202 int getkey(void)
203 #else
204 char uart_getchar (void)
205 #endif
206 {
207     /* wait until character has been received */
208     //while (!(UART3_S1 & UART_S1_RDRF_MASK));
209     if(UART3_S1 & UART_S1_RDRF_MASK)
210     /* Return the 8-bit data from the receiver */
211         return UART3_D;
212     else
213         return 0 ;
214 }
```

A modify required of `uart_getchat()` in `sci.c`
Must change

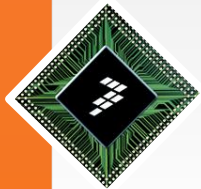
“`while(!(UART3_S1&UART_S1_RDRF_MASK))`”

To

“`if(UART3_S1&UART_S1_RDRF_MASK)`”

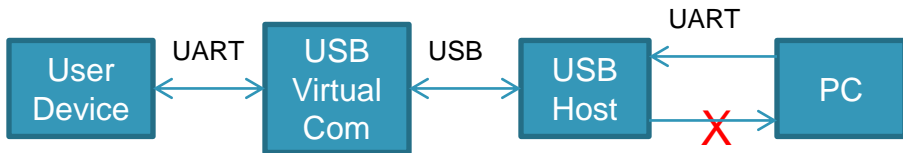
If NOT

Code will be stop here to wait data from PC



Debug 2 – Can't transfer data to PC terminal

- Issue :
 - After bug 1 fixed
 - Data transfer to USB CDC Host from PC terminal is working
 - When USB CDC Device receive data from host that will return the data to Host , but there is nothing display on PC terminal .
- Root cause:
 - Something wrong at “write data to UART” block .
 - Write data to UART by “sci2_PutChar(usb2uart[i])” when usb2uart_num not zero .
 - **But “sci2_PutChar() “ is empty that decelerated in sci.c .**
- Workaround :
 - To call “uart_putchar()” instead .
 - “uart_putchar(usb2uart[i]);” decelerated in sci.c too .
 - Detail please refer to next page .

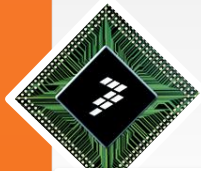


```

240 if (0 == check_open)
241 {
242   _io_cdc_serial_open(f_usb, device_name, (char *)&usb_open_param);
243   check_open = 1;
244 }
245 else
246 {
247   DisableInterrupts;
248   uart2usb_num = buff_index;
249   for (i = 0; i < buff_index; i++)
250     uart2usb[i] = buff[i];
251   buff_index = 0;
252   EnableInterrupts;
253 }
254
255 /*write data to USB */
256 if (usb2usb_num)
257 {
258   num_done = _io_cdc_serial_write(f_usb, uart2usb, (int_32)(sizeof(uart2usb[0]) * uart2usb_num));
259   if(num_done > 0)
260   {
261     for (i = (uint_32)num_done; i < uart2usb_num; i++) /* move buffer data, remove the written ones */
262       uart2usb[i - num_done] = uart2usb[i];
263     uart2usb_num -= num_done;
264     char_to_recv += num_done;
265   }
266 }
267
268
269
270
271 /* Read data from USB */
272 if(char_to_recv > 0 || Read_USB_Data)
273 {
274   Read_USB_Data = 0;
275   num_done = _io_cdc_serial_read(f_usb, usb2uart + usb2uart_num, (int_32)(sizeof(uart2usb) / sizeof(uart2usb[0]) - usb2uart_num));
276   if(num_done > 0 && usb2uart[usb2uart_num] != 0)
277   {
278     usb2uart_num += num_done;
279     if(char_to_recv >= num_done)
280     {
281       char_to_recv -= num_done;
282     }
283     else
284     {
285       char_to_recv = 0;
286     }
287   }
288   else
289   {
290     char_to_recv = 0;
291   }
292 }
293
294 /* write them to UART */
295 if (usb2uart_num > 0)
296 {
297   for (i = 0; i < usb2uart_num; i++)
298   {
299     MCU_MCF5113F32B
300     printf("received-> %c\n",usb2uart[i]);
301     sci2_PutChar(usb2uart[i]);
302   }
303   usb2uart_num = 0;
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

```

Write data to UART



Debug 2 – Workaround

–Can't transfer data to PC terminal

CDC_Task()

```

/* write them to UART */
if (usb2uart_num > 0)
{
    for (i = 0; i < usb2uart_num; i++)
    {
        //sci2_PutChar(usb2uart[i]);
        uart_putchar(usb2uart[i]);
        usb2uart[i] = 0; //clear buffer
    }
    usb2uart_num = 0;
}

```

```

216 *
217 *END*-----
218 void sci2_PutChar(char send)
219 {
220     [redacted]
221 }
222

```

Not thing done here

To call "uart_putchar()" instead

Sci.c

```

* Function Name      : TERMIO_PutChar
* Returned Value    :
* Comments          :
*
*                   This function sends a char via SCI.
*

```

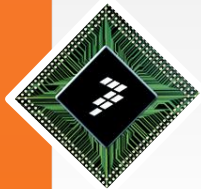
```

*END*-----
void uart_putchar (char ch)
{
    /* Wait until space is available in the FIFO */
    while(!(UART3_S1 & UART_S1_TDRE_MASK)){};

    /* Send the character */
    UART3_D = (uint_8)ch;
}

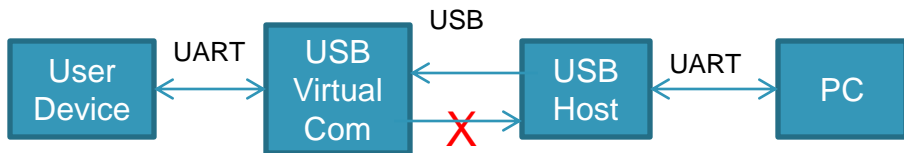
```





Debug 3– Can't receive data from CDC Device

- Issue :
 - After bug 1 fixed ,
 - When USB CDC Host transfer a data to CDC Device , we can receive one response from CDC Device
 - If CDC device transfer data to USB CDC Host **voluntarily** , there hasn't any response at Host side .
- Root cause :
 - Something wrong at "Read data from USB"
 - Read data from USB by "`_io_cdc_serial_read()`" , but it only called when "`char_to_recv`" asserted not periodicity .
 - And "`char_to_recv`" only asserted after USB CDC Host send data to USB CDC device . It will be cleared after two byte received from USB CDC device when one byte sent by host .
- Workaround :
 - To execute "`_io_cdc_serial_read()`" periodicity .
 - One more condition added to inter "read data from USB" .
 - Detail please refer to next page .



```

240
241 if (0 == check_open)
242 {
243     _io_cdc_serial_open(f_usb, device_name, (char *)usb_open_param);
244     check_open = 1;
245 }
246 }
247 else
248 {
249     DisableInterrupts;
250     uart2usb_num = buff_index;
251     for (i = 0; i < buff_index; i++)
252         uart2usb[i] = buff[i];
253     buff_index = 0;
254     EnableInterrupts;
255
256     /* write data to usb */
257     if (uart2usb_num)
258     {
259         num_done = _io_cdc_serial_write(f_usb, uart2usb, (int_32)(sizeof(uart2usb) * uart2usb_num));
260         if (num_done != 0)
261         {
262             for (i = (uint_32)num_done; i < uart2usb_num; i++) /* move buffer data, remove the written ones */
263                 uart2usb[i - num_done] = uart2usb[i];
264             uart2usb_num = num_done;
265             char_to_recv += num_done;
266         }
267     }
268
269     /* Read data from USB */
270     if (char_to_recv > 0 || Read_USB_Data)
271     {
272         Read_USB_Data = 0;
273         num_done = _io_cdc_serial_read(f_usb, usb2uart + usb2uart_num, (int_32)(sizeof(uart2usb) / sizeof(uart2usb[0]) - usb2uart_num));
274         if (num_done > 0 && usb2uart[usb2uart_num] != 0)
275         {
276             usb2uart_num += num_done;
277             if (char_to_recv == num_done)
278             {
279                 char_to_recv -= num_done;
280             }
281             else
282             {
283                 char_to_recv = 0;
284             }
285         }
286         else
287         {
288             char_to_recv = 0;
289         }
290     }
291
292     /* write them to UART */
293     if (usb2uart_num > 0)
294     {
295         for (i = 0; i < usb2uart_num; i++)
296             MCU_WFS315E0
297             printf(" --received-- %c\n",usb2uart[i]);
298         sct2_PutChar(usb2uart[i]);
299     }
300     usb2uart_num = 0;
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 }
  
```

Debug 3 – Workaround

– Can't receive data from CDC Device

One more condition "Read_USB_Data" added to inter "read data from USB"

CDC_Task()

```

291
292 /* Read data from USB */
293 if(char_to_recv > 0 || Read_USB_Data)
294 {
295     Read_USB_Data = 0 ;
296     num_done = _io_cdc_serial_read(&usb, usb2uart + usb2uart_num,
297 if(num_done > 0 && usb2uart[usb2uart_num] != 0 )
298     {
299         usb2uart_num += num_done;
300         if(char_to_recv >= num_done)
301         {
302             char_to_recv -= num_done;
303
304
305
306         char_to_recv = 0;
307     }
308 }
309 else
310 {
311     char_to_recv = 0;
312 }
313 }
314
  
```

To execute "_io_cdc_serial_read()" periodicity

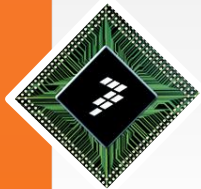
Main()

```

204 for(;;)
205 {
206     Poll();
207     CDC_Task();
208     if(char_to_recv == 0)
209     {
210         Read_USB_Data = 1 ;
211         time_delay(10);
212     }
213
214     __RESET_WATCHDOG(); /* feeds the dog */
215 } /* loop forever */
  
```

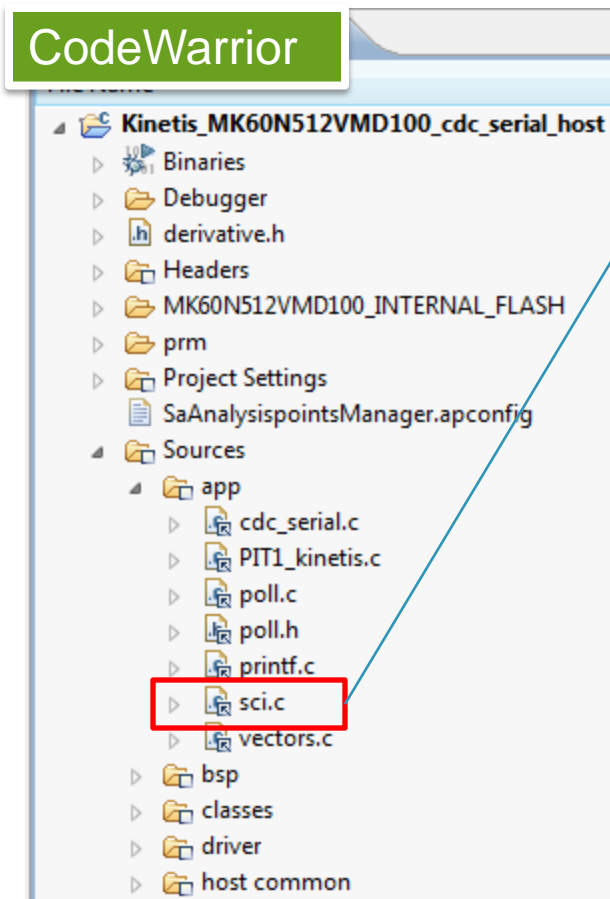
To set flag "Read_USB_Data" periodicity

- For this issue , please configure CDC device to send the data to USB Host voluntarily
 - May be press switch to send data to USB CDC host .



Backup

- There are a little different between IAR and CodeWarrior environment



SCI driver source code is different

Something we modified in "sci.c" please do it on "sci_kinetis.c"

