USB or not: CDC with Processor Expert

Posted on **March 10, 2012** by **Erich Styger**

I had a PREN student showing up into my office. He wanted to choose a microcontroller for that project. One requirement put on the table was *"it needs USB"*. Well, I asked **why** USB is required. I was not surprised by the answer: *"to use USB instead of RS-232"*. Wow. So what he really wanted was USB CDC (Communication Device Class). Yep. Most notebooks today have **no** serial COM port (see "Processor Expert Configurations"). But because "USB is serial" does not mean "USB CDC is simple". Nope. USB is **not simple**. But it can be with Processor Expert.

RS-232 **is** simple and not expensive. That's why I use it in so many designs. If I need a serial communication with my board, then my first choice is RS-232. It is straight forward and **just works**. Yes, RS-232 it is not plug&play: you have to configure baud and connection settings. But a typical RS-232/SCI/serial implementation in my embedded design takes typically one or two source files with a total of less than 20 KByte of sources code (counting the comments too), and impacting my application with less than 1 KByte of code.

USB on the other hand promises plug&play, plus it can power my typical board. USB means much more than two serial wires plus power: its needs a full-blown communication stack! As such, it is much more complex: the Freescale USB Stack stack for CDC uses more than 32 source files with about 400 KByte of source code. Adding CDC to my application means 10-20 KByte of code! Yikes. So a **lot** of complexity to send some strings over a serial wire. And if something does not work, I better buy one of these expensive USB protocol analyzer? Definitely not the easiest thing.

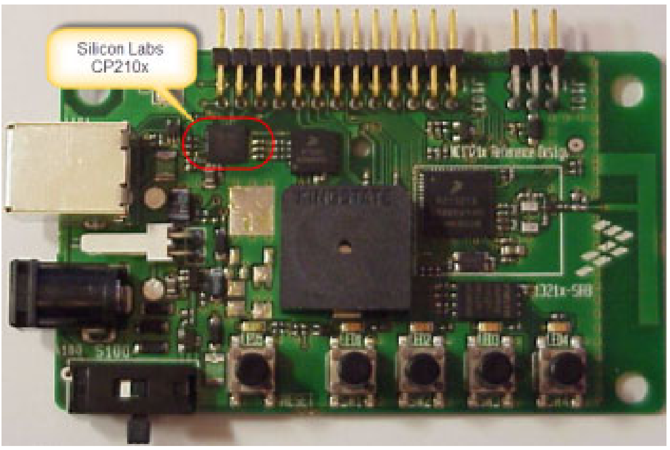
But: USB is reality and everywhere. Worse: new notebooks rarely have a physical COM port. I'm lucky that my old Dell still has one :-). Is there no demand any more for 'engineering notebooks' with a COM port? Sigh!

With no physical COM port, I end up use a USB-to-Serial converter:

USB-to-Serial Converter Cables on my bench

Yep, that works, but is a pain: multiple drivers. If I'm lucky, I get the 64bit drivers up and running. If I need one, for sure it is in the other lab. I need to make sure that the engineer in the field is equipped with the converters too. Imaging traveling to the customer, only to find out I left the adapter on the bench in the lab :-(.

To avoid that kind of problem, a solution is to add a Serial-To-USB chip on your board: What has been don on the Freescale MC13213 SRB board: it is using the Silicon Labs Dual CP210x UART to USB Bridge. Freescale itself is offering a similar solution presented here. Still, it typically does not solve your drivers problem. But you can use a normal microcontroller without USB capability: the USB interface is added with a separate IC.



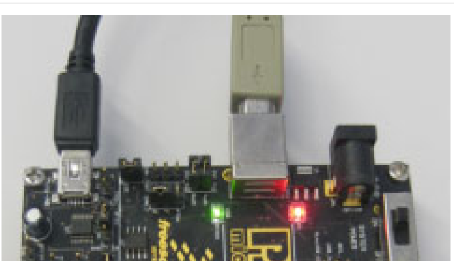
SRB Board with Silicon Labs UART to USB Chip

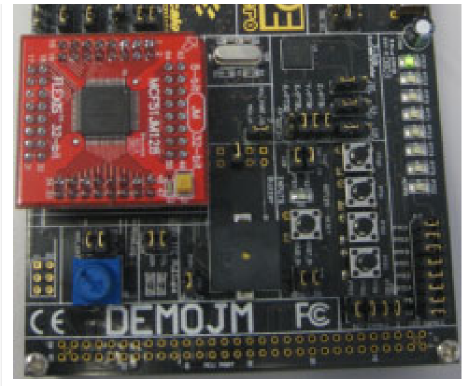
More and more silicon vendors (including Freescale) are offering microcontrollers with USB on the chip. So no need any more for that converter IC. They offer as well software and USB stacks. Freescale offers the USB stack with PHDC Support, or drivers come with MQX. It comes with documentation and examples and supports many processors and USB modes.

But if you want to integrate this USB stack into your own design, it gets really challenging. It is really hard to isolate the files and functions you need. There has to be a way to make things simpler for me! What I need is a Processor Expert component which I can easily integrate and use with my other Processor Expert components.

The good news is: I have finished a USB CDC stack integration as Processor Expert component. It is using the Freescale USB stack with PHDC Support. Up to switching back and forward between RS-232 and USB CDC as shown in the post on Processor Expert Configurations.

I'll show in the steps below how to use USB CDC with a DEMOJM board and the MCF51JM128: The application is a bare metal one which is using USB CDC for communication with the host.





— DEMOJM Board with MCF51JM128 running USB CDC

The USB CDC support is realized with two components:

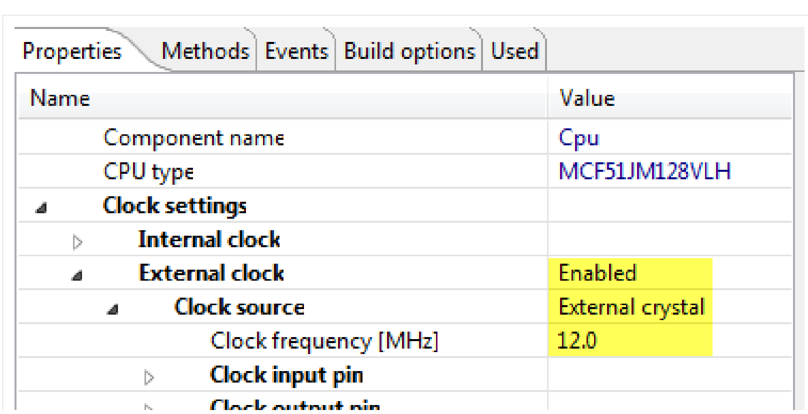
1. The **Init_USB_OTG** component: this one comes with CodeWarrior to initialize the USB peripheral.
2. The **FSL_USB_Stack** component: this one is a wrapper around the Freescale USB Stack. Additionally two circular buffer components implement buffering.

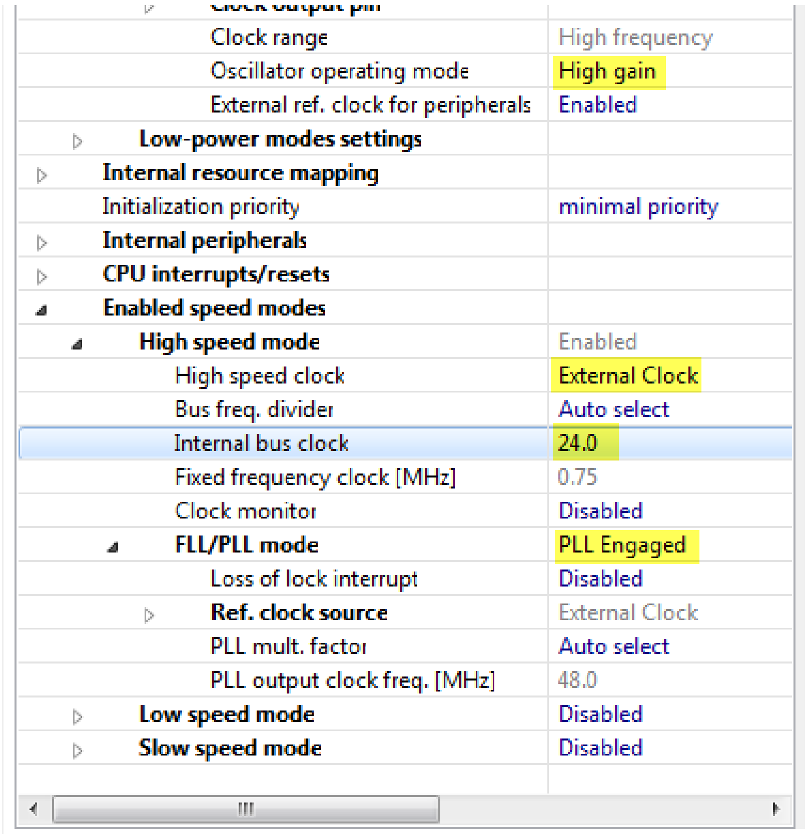
To start, I check the jumper settings on DEMOJM board:

- IRQ Fault: no jumper
- VHOST_EN: OFF position
- VBUS_SEL: VBUS position
- DN_DOWN/J13: on position
- DP_DOWN/J14: on position
- USB_ID: no jumper
- PULLUP: no jumper

Creating a project and adding USB CDC support is possible in 10 steps:

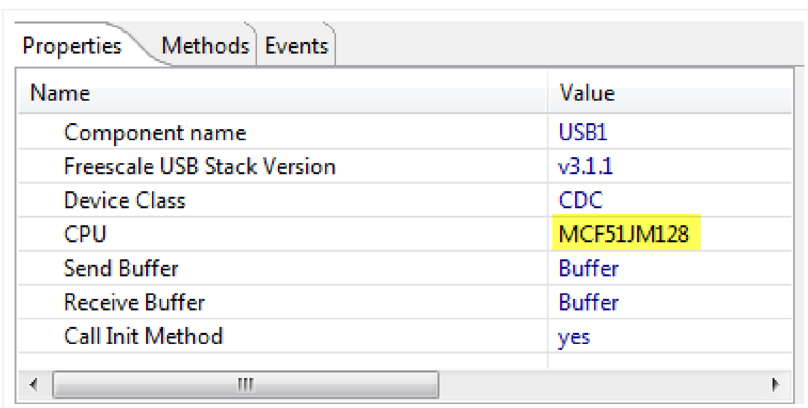
1. I Create a new project with the wizard with Processor Expert option enabled (see Quickstart for Processor Expert Project)
2. For USB I need to configure the clock to 24 MHz: Using the 12 MHz crystal on the board in high gain mode a PLL Engaged:





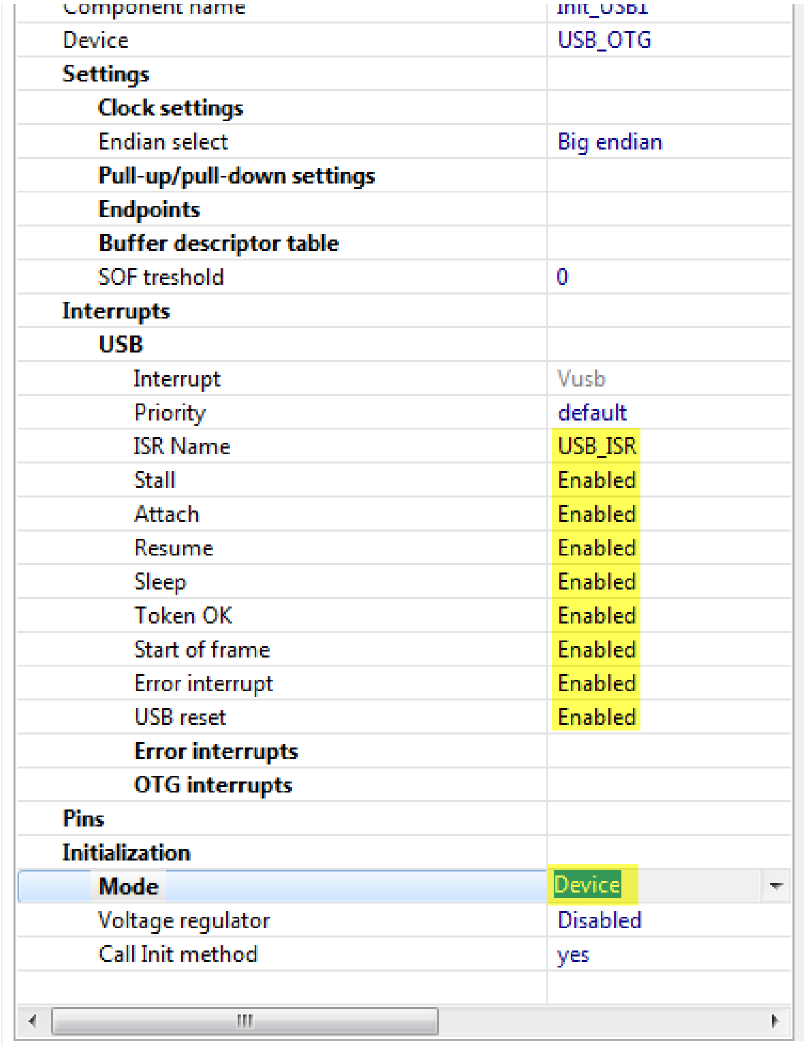
— Clock Settings for USB

3. I add the **FSL_USB_Stack** component to the project. I set the device as *MCF51JM128*:



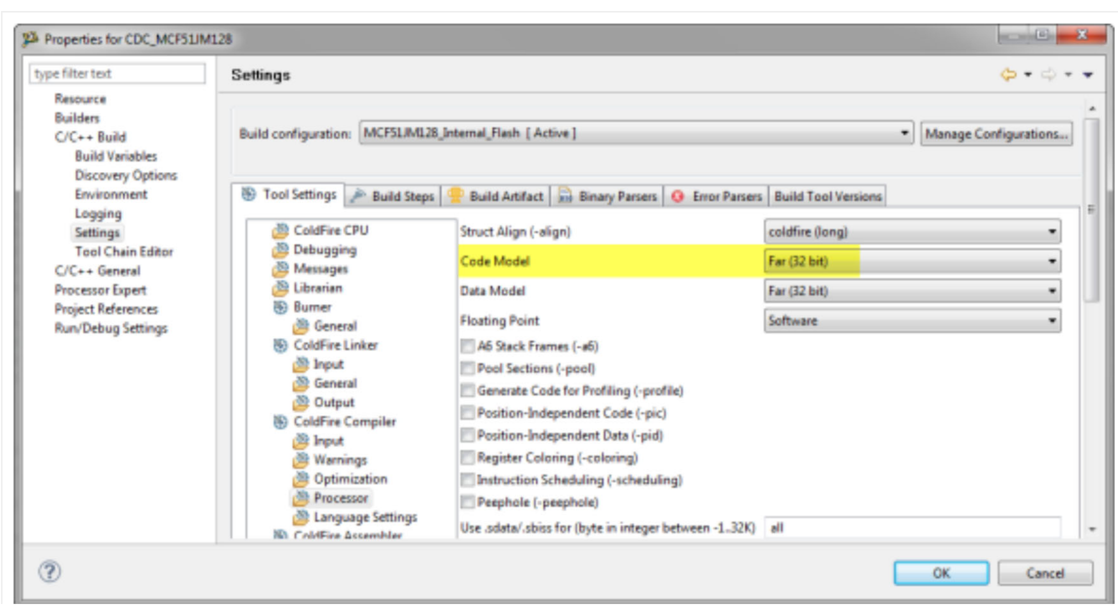
— FSL_USB_Stack Settings

4. The FSL_USB_Stack inherits from the **USB_OTG_Init** component, so it gets automatically added as 'sub-component'. All the settings get set up automatically, things like *USB_ISR* as interrupt name, *enabling*necessary interrupts and device settings. So nothing to do here, everything is automatically set up:



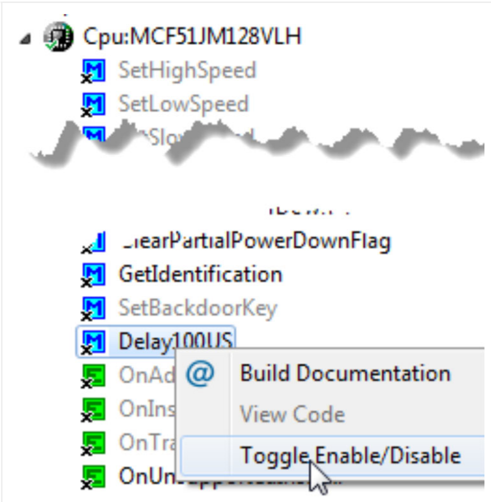
— USB_OTG_Init Settings

5. USB has a price: the small data model is not big enough. I need to change the build tools settings to the 32bit code model. I can do this in the project settings for the compiler:



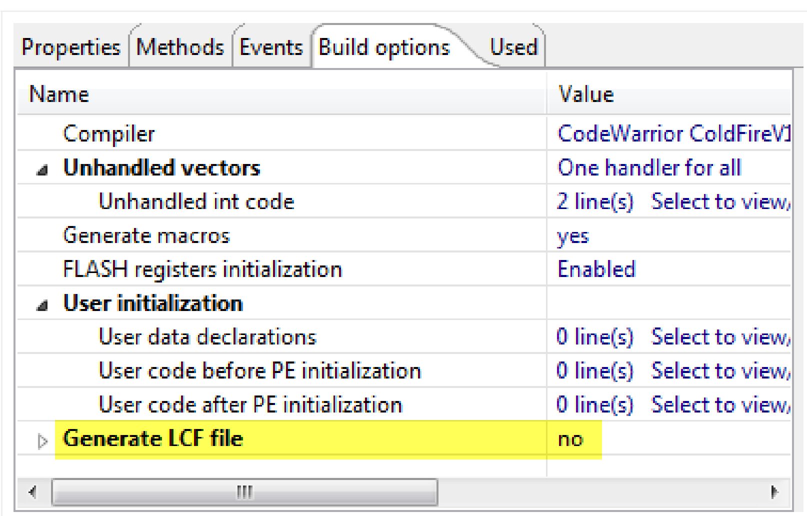
— 32bit Code Model for the ColdFire Compiler

6. I add the **Wait** component to the project and enable *Delay100us*in the CPU component:



— Enabling Delay100US in the CPU component

7. This is an optional step: A add two **LED** component to the project. The LED is configured with the *Anode on the port side*. I configure one LED for pin *PTE2*, the other for pin *PTE3*.
8. Now I *generate code* with Processor Expert. This generates the linker file which I need to change. To prevent further linker file modification, I disable the generation of the LCF file: For this I select the CPU component, use the Inspector View and change the setting in the *Build Options* tab:

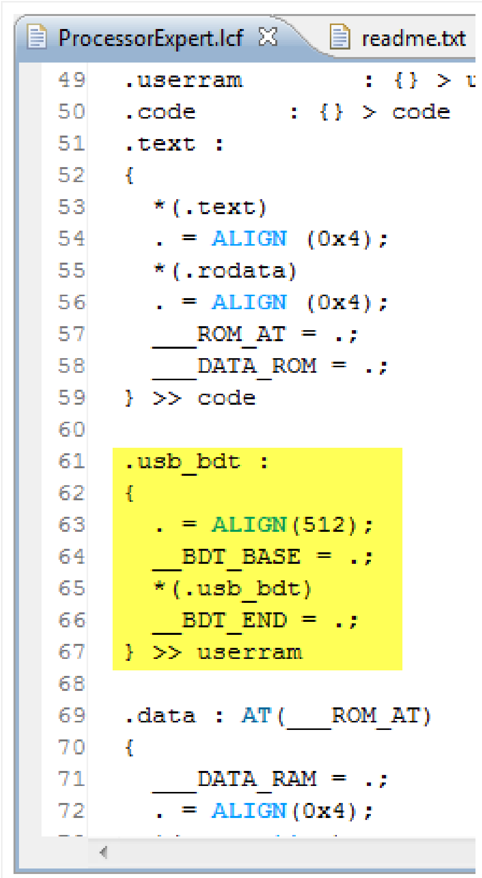


— Disabled Generation of Linker LCF File in CPU component

9. To change the linker file, I open the ProcessorExpert.lcf file. I need to allocate the USB Buffer Data block (BDT). I

add the following block between *.text* and *.data*:

```
.usb_bdt :
{
  . = ALIGN(512);
  __BDT_BASE = .;
  *(.usb_bdt)
  __BDT_END = .;
} >> userram
```



```
ProcessorExpert.lcf    readme.txt
  .userram        : {} > u
  .code       : {} > code
  .text :
  {
    *(.text)
    . = ALIGN (0x4);
    *(.rodata)
    . = ALIGN (0x4);
    ___ROM_AT = .;
    ___DATA_ROM = .;
  } >> code

  .usb_bdt :
  {
    . = ALIGN(512);
    __BDT_BASE = .;
    *(.usb_bdt)
    __BDT_END = .;
  } >> userram

  .data : AT(___ROM_AT)
  {
    ___DATA_RAM = .;
    . = ALIGN(0x4);
```

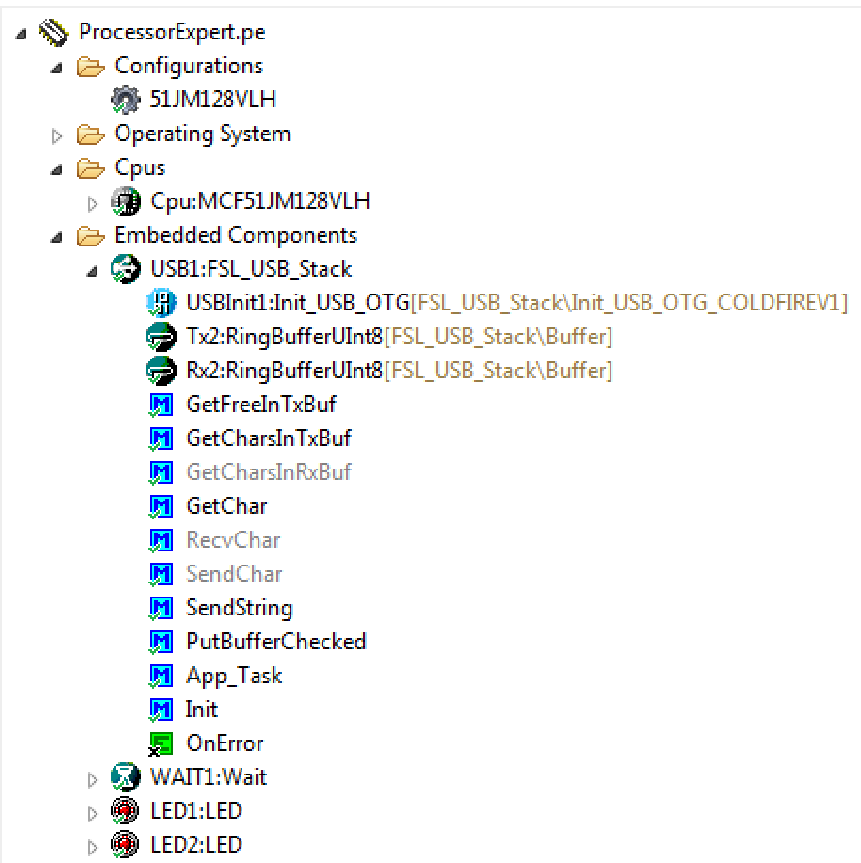
— USB BDT allocation in Linker File

10. I add the following code (e.g. in a separate file) and call *CDC_Run()* from *main()*.

```
#include "LED1.h"
#include "LED2.h"
#include "USB1.h"
#include "WAIT1.h"

static uint8_t cdc_buffer[USB1_DATA_BUFF_SIZE];
static uint8_t in_buffer[USB1_DATA_BUFF_SIZE];

void CDC_Run(void) {
    int i;

    for(;;) {
        while(USB1_App_Task(cdc_buffer, sizeof(cdc_buffer))==ERR_BUSOFF) {
        /* device not enumerated */
            LED1_Neg(); LED2_Off();
            WAIT1_Waitms(200);
        }
```

```
        LED1_Off(); LED2_Neg();
        if (USB1_GetCharsInRxBuf()!=0) {
            i = 0;
            while(   i<sizeof(in_buffer)-1
                && USB1_GetChar(&in_buffer[i])==ERR_OK
                )
            {
            i++;
            }
            in_buffer[i] = '\0';
            (void)USB1_SendString((unsigned char*)"echo: ");
            (void)USB1_SendString(in_buffer);
            (void)USB1_SendString((unsigned char*)"\r\n");
        } else {
            WAIT1_Waitms(50);
        }
    }
}
```

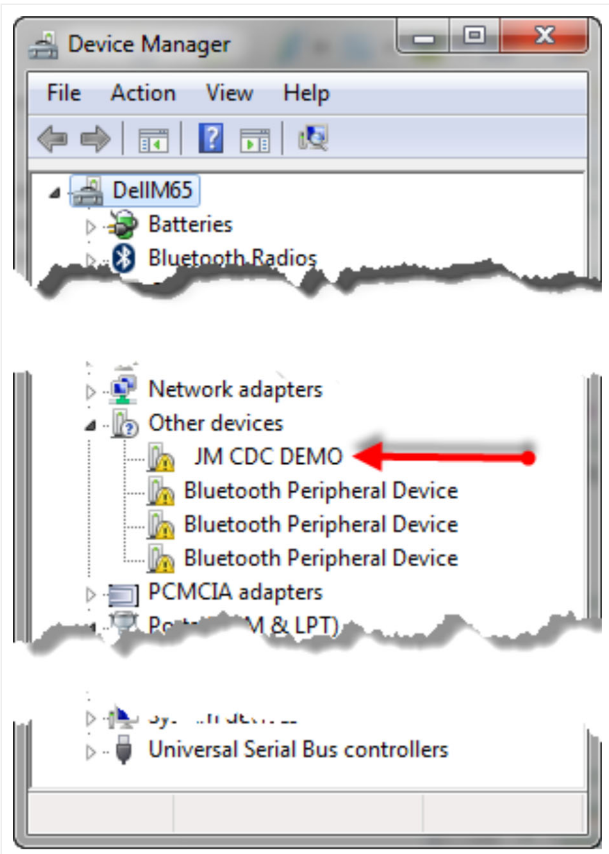
The code runs in an endless loop. It calls the USB application function **USB1_App_Task()** with a send buffer. **USB1_GetCharsInRxBuf()** is used to check if there are any incoming characters. **USB1_GetChar()** is used to get the characters from the input buffer, and **USB1_SendString()**is used to send strings. The component uses two circular ring buffers: one for the input stream and one for the output stream:



— Application Components

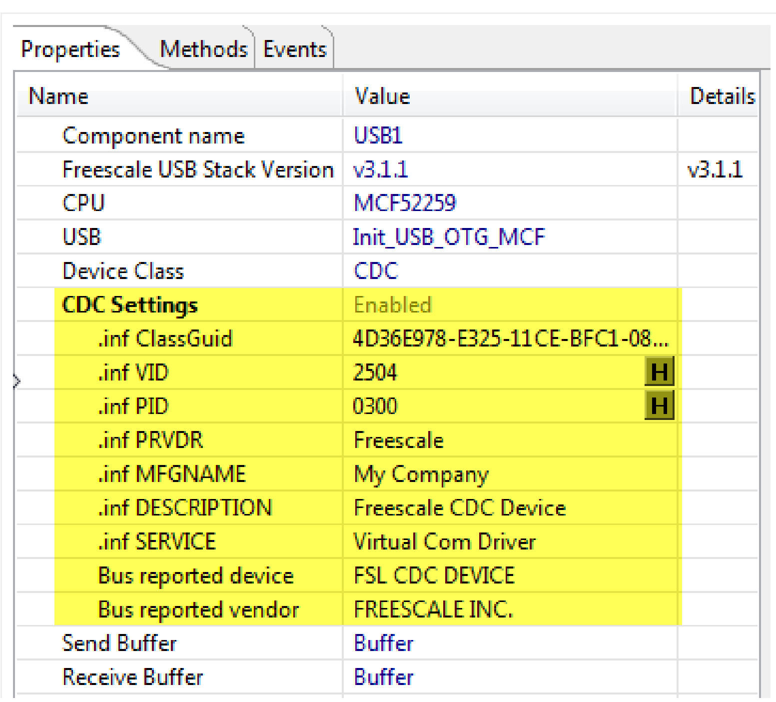
Time to build, download and run the application. For now I power the board, but do not connect the USB CDC connection to the host. I see the LED1 flashing slowly. Once I plug in the USB CDC cable, LED2 flashes fast to show that the device is enumerated on the USB bus. If not, then your host probably needs some drivers …..

The first time I attach the board on my host, it will show up as **JM CDC DEMO** in the Device Manager:



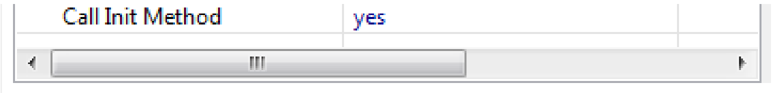
— JM CDC Demo Device

So I need a driver for it? Actually, a **.inf** file is enough. The FSL_USB_Stack has generated the .inf file based on my settings into the Documentation folder:

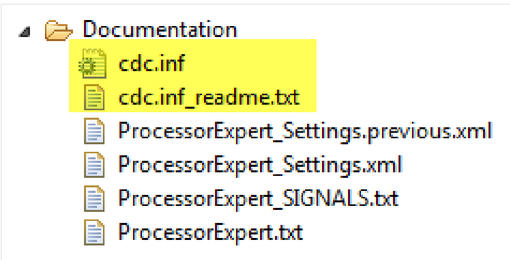


Properties | Methods | Events

Name	Value	Details
Component name	USB1	
Freescale USB Stack Version	v3.1.1	v3.1.1
CPU	MCF52259	
USB	Init_USB_OTG_MCF	
Device Class	CDC	
CDC Settings	Enabled	
.inf ClassGuid	4D36E978-E325-11CE-BFC1-08...	
.inf VID	2504 H	
.inf PID	0300 H	
.inf PRVDR	Freescale	
.inf MFGNAME	My Company	
.inf DESCRIPTION	Freescale CDC Device	
.inf SERVICE	Virtual Com Driver	
Bus reported device	FSL CDC DEVICE	
Bus reported vendor	FREESCALE INC.	
Send Buffer	Buffer	
Receive Buffer	Buffer	

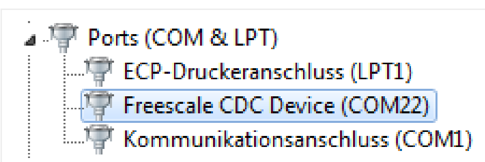


— USB driver and .inf settings



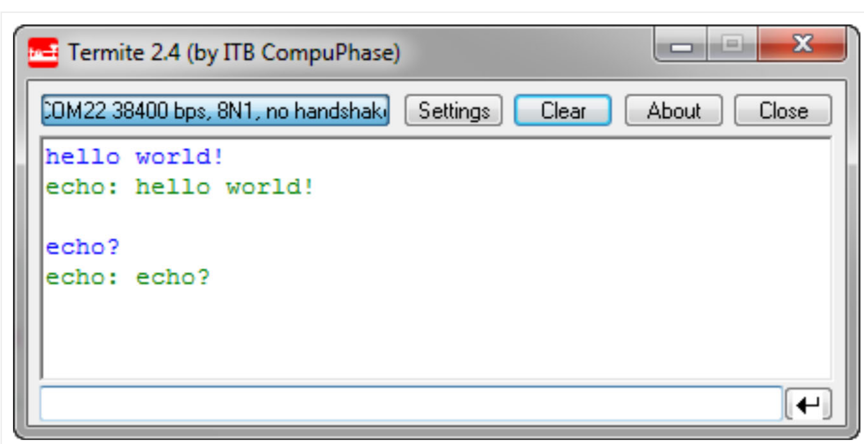
— cdc.inf fle created by the component

There is as well a *cdc.inf_readme.txt* file: follow the information to install the .inf file. With the driver installed it should show the device as specified in the .inf settings:



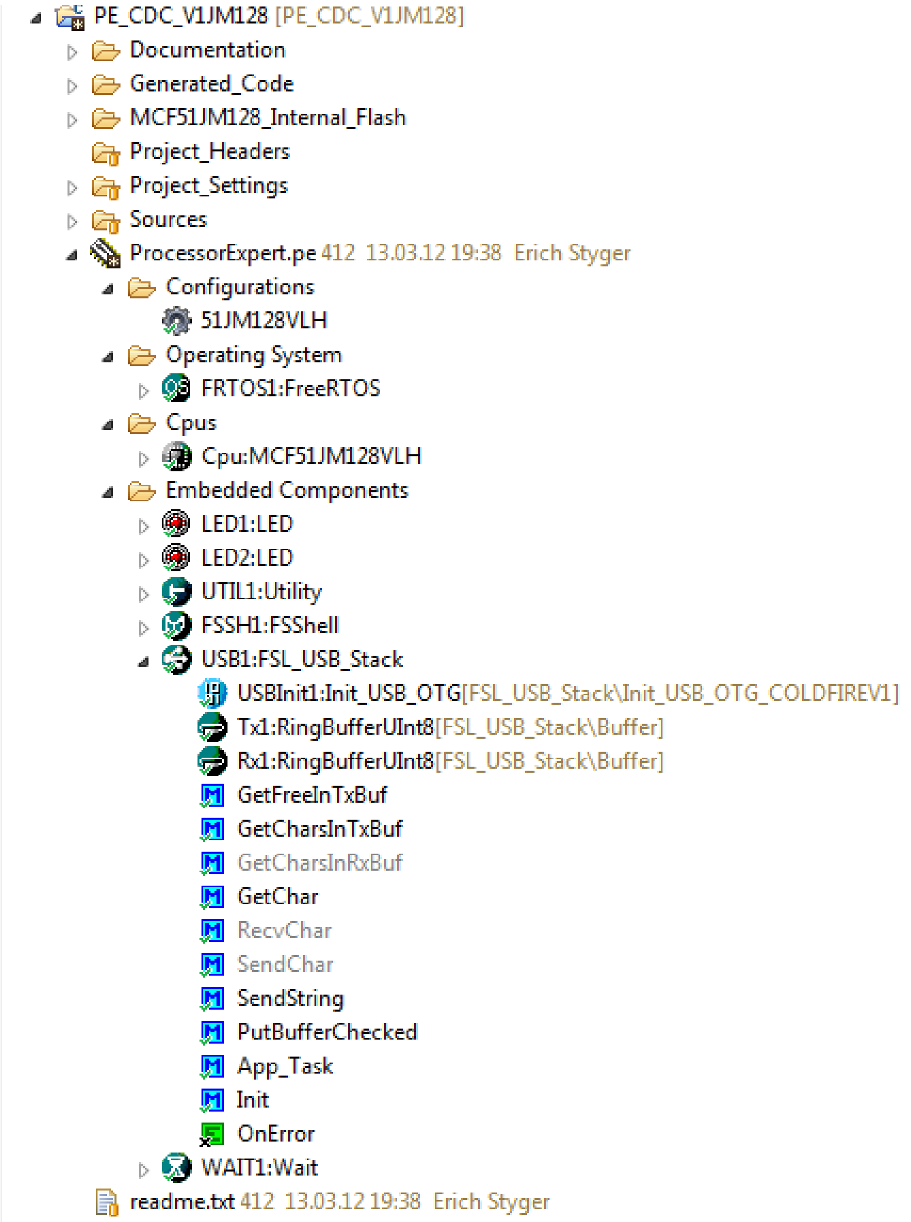
— Freescale CDC Device

Now I can connect to COM22 (or whatever port it is) with my terminal:



— USB CDC Communication

The above application is a very simple one, and does not use an RTOS. A more complex example using a Shell and FreeRTOS is shown below:



— USB CDC Example Project

The components and the example project can be downloaded from here.

Happy CDC USBeing 🙂