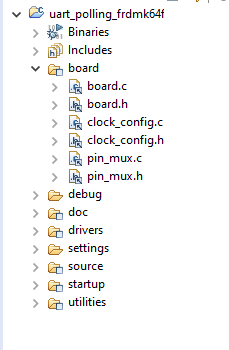
FRDM K64 - Uart Test

In this example we will send and receive structure from Windows C# application to K64 Uart interface

Sources:

Start from Kinetis design studio 3.2 , UART polling example from the FRDM k64 Sdk



The example use UART0

PORT\_SetPinMux(PORTB, PIN16\_IDX, *kPORT\_MuxAlt3*); /\* PORTB16 (pin E10) is configured as UART0\_RX \*/

PORT\_SetPinMux(PORTB, PIN17\_IDX, *kPORT\_MuxAlt3*); /\* PORTB17 (pin E9) is configured as UART0\_TX \*/

The problem is that PTB16 and PTB17 are not output in the FRDM k64

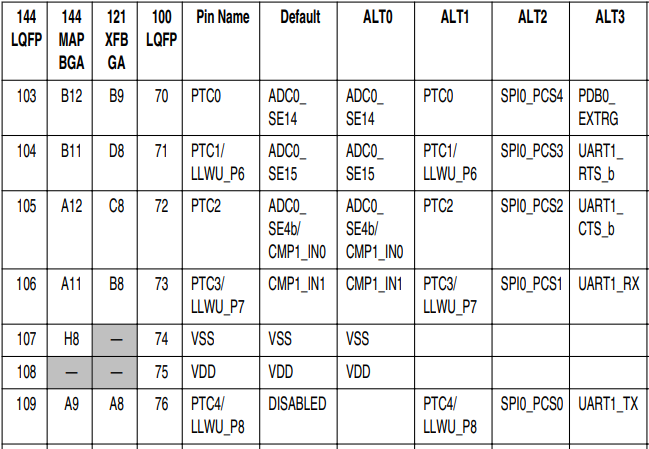
We will use UART1 TX And RX

K64 datasheet:

<https://www.nxp.com/docs/en/data-sheet/K64P144M120SF5.pdf>

k64 frdm schematic:

FRDM-K64F-SCH-E4.pdf



From the K64 datasheet we learn that

PTC3 is UART1\_RX at pin mux ALT3

PTC4 is UART1\_TX at pin mux ALT3

Let’s add in the pin\_mux.c the following:

**void** **BOARD\_InitPins**(**void**)

{

CLOCK\_EnableClock(*kCLOCK\_PortB*); /\* Port B Clock Gate

CLOCK\_EnableClock(*kCLOCK\_PortC*); Control: Clock enabled \*/

PORT\_SetPinMux(PORTB, PIN16\_IDX, *kPORT\_MuxAlt3*); /\* PORTB16 (pin E10) is configured as UART0\_RX \*/

PORT\_SetPinMux(PORTB, PIN17\_IDX, *kPORT\_MuxAlt3*); /\* PORTB17 (pin E9) is configured as UART0\_TX \*/

PORT\_SetPinMux(PORTC, PIN3\_IDX, *kPORT\_MuxAlt3*); /\*configured as UART1\_RX\*/

PORT\_SetPinMux(PORTC, PIN4\_IDX, *kPORT\_MuxAlt3*); /\* configured as UART1\_TX\*/

SIM->SOPT5 = ((SIM->SOPT5 &

(~(SIM\_SOPT5\_UART0TXSRC\_MASK))) /\* Mask bits to zero which are setting \*/

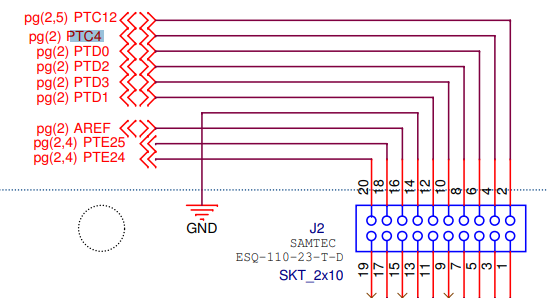
| SIM\_SOPT5\_UART0TXSRC(SOPT5\_UART0TXSRC\_UART\_TX) /\* UART 0 transmit data source select: UART0\_TX pin \*/

);

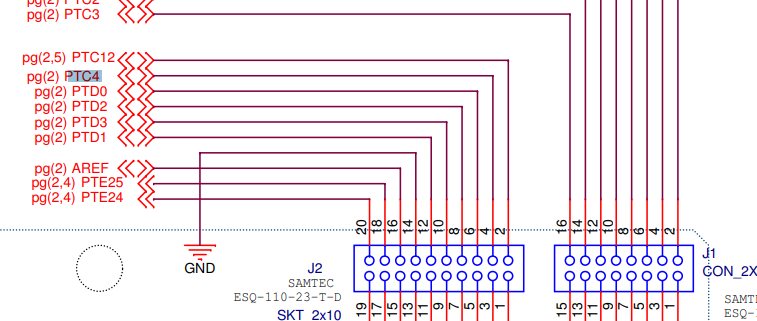
}

Connection:

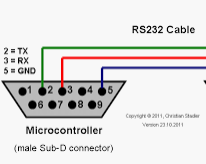
PTC4 located in J2 pin4



PTC3 located in J1 pin 16:



From the PC we are using COM Connector



Lets finally connect

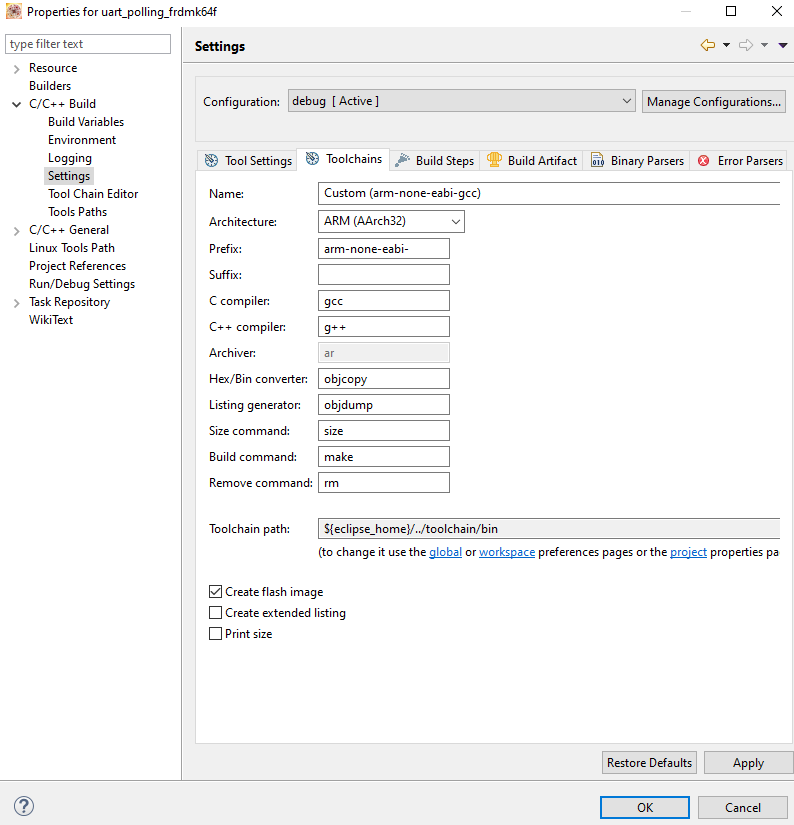
Pin 2 in rs232(tx) to j1 pin16(rx)

Pin3 in rs 232(rx) to j2 pin4(tx)

We cross the rx tx

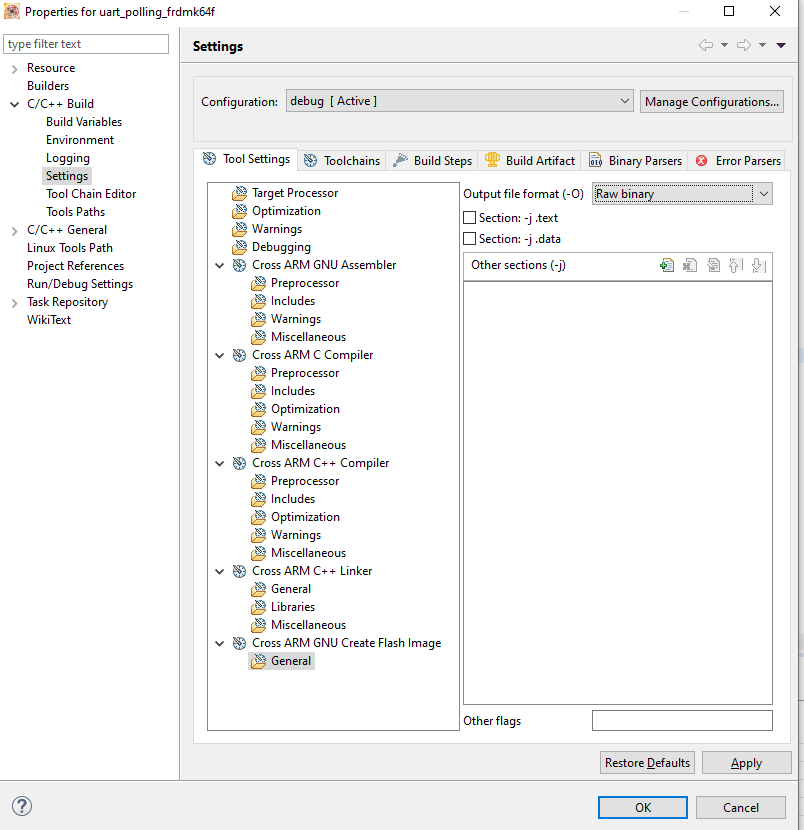
In Kinetis Design studio 3.2 we can make a bin file and use drag and drop , simple copy the bin into the Mbed disk

In settings, toolchanin:

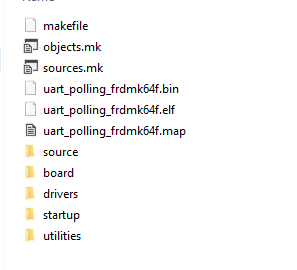


Select create flash image, press ok and open settings again:

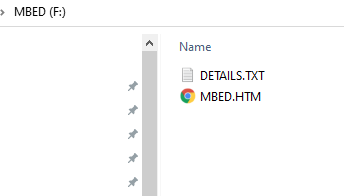
In project properties , select raw binary



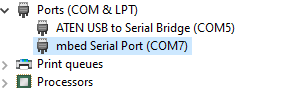
Compile your project , you should now have a bin file in the debug directory



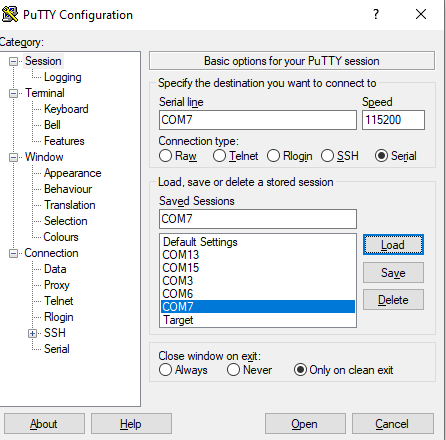
This file need to be copy into the MBED disk

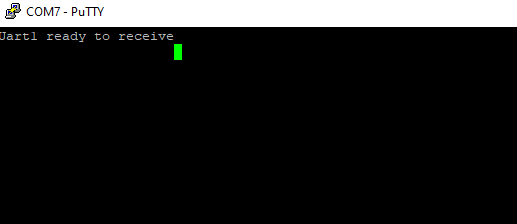


Go to device manager, ports , and find the mbed com port



Open terminal, like putty:

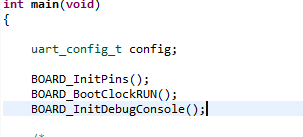




Hey , before we continue , there are some changes that need to be done in the code.

The original sample uses uart0 for communication, we change it to UART1,

We do want to see print in uart0 , so we just need to initialize the debug console:



Add BOARD\_InitDebugConsole and you will see PRINTF in your putty terminal.

Our code wait for specific structure:

**#pragma** pack(push, 1)

**typedef** **struct** Header

{

uint16\_t UartPrefix;

uint16\_t opcode;

uint8\_t msgSize;

} Header;

**typedef** **struct** MSG1

{

Header header;

uint32\_t data;

uint8\_t value;

uint16\_t value1;

uint16\_t crc;

} MSG1;

**typedef** **struct** MSG2

{

Header header;

uint16\_t data;

uint8\_t value;

uint32\_t value1;

uint16\_t crc;

} MSG2;

**#pragma** pack(pop)

It is important to make the structure alignment of both compiles , KDS and Visual studio c# on the same packing

We are using 1 byte alignment

We are Initializing UART1 and waiting for this messages from Windows c# application

UART\_GetDefaultConfig(&config);

config.baudRate\_Bps = BOARD\_DEBUG\_UART\_BAUDRATE;

config.enableTx = true;

config.enableRx = true;

UART\_Init(COMM\_UART, &config, CLOCK\_GetFreq(COMM\_UART\_CLKSRC));

PRINTF("Uart1 ready to receive\n");

//UART\_WriteBlocking(COMM\_UART, txbuff, sizeof(txbuff) - 1);

MSG1 msg;

**while** (1)

{

UART\_ReadBlocking(COMM\_UART, (uint8\_t \*)&msg, **sizeof**(msg));

PRINTF("UartPrefix 0x%x\n" , msg.header.UartPrefix);

PRINTF("msgSize 0x%x\n" , msg.header.msgSize);

PRINTF("opcode 0x%x\n" , msg.header.opcode);

PRINTF("data 0x%x\n" , msg.data);

PRINTF("value1 0x%x\n" , msg.value);

PRINTF("value1 0x%x\n" , msg.value1);

PRINTF("crc 0x%x\n" , msg.crc);

//UART\_WriteBlocking(DEMO\_UART, &ch, 1);

}

In Windows PC , we are using ATEN USB



It is COM5.

Now the intersting part of the story here:

The K64 by default send and receive inverted data to host and from host.

For example , 0x1 send is received 0x7f

The K64 have UART internal registers that can be found detaild in the K64 reference manual

By looking in the reference manual

We see two regisrers TXINV and RXINV

**typedef** **union** \_hw\_uart\_s2

{

uint8\_t U;

**struct** \_hw\_uart\_s2\_bitfields

{

uint8\_t RAF : 1; /\*!< [0] Receiver Active Flag \*/

uint8\_t LBKDE : 1; /\*!< [1] LIN Break Detection Enable \*/

uint8\_t BRK13 : 1; /\*!< [2] Break Transmit Character Length \*/

uint8\_t RWUID : 1; /\*!< [3] Receive Wakeup Idle Detect \*/

uint8\_t RXINV : 1; /\*!< [4] Receive Data Inversion \*/

uint8\_t MSBF : 1; /\*!< [5] Most Significant Bit First \*/

uint8\_t RXEDGIF : 1; /\*!< [6] RxD Pin Active Edge Interrupt Flag \*/

uint8\_t LBKDIF : 1; /\*!< [7] LIN Break Detect Interrupt Flag \*/

} B;

} hw\_uart\_s2\_t;

hw\_uart\_s2\_t S2;

S2.U = UART\_ReadS2(COMM\_UART);

PRINTF("S2: 0x%x\n", S2.U);

PRINTF("RXINV: %d\n", S2.B.RXINV);

PRINTF("MSBF: %d\n", S2.B.MSBF);

S2.B.RXINV = 1;

S2.B.MSBF = 0;

UART\_WriteS2(COMM\_UART,S2.U);

S2.U = UART\_ReadS2(COMM\_UART);

PRINTF("S2: 0x%x\n", S2.U);

PRINTF("RXINV: %d\n", S2.B.RXINV);

PRINTF("MSBF: %d\n", S2.B.MSBF);

uint8\_t C1 = UART\_ReadC1(COMM\_UART);

PRINTF("C1: 0x%x\n", C1);

uint8\_t C3 = UART\_ReadC3(COMM\_UART);

PRINTF("C3: 0x%x\n", C3);

C3 = C3 | 0x10;

UART\_WriteC3(COMM\_UART, C3);

The code here put 1 in both:

By doing so the data is now inverted right.

Another problem ( not yet solved ) is that the TX fifo on the startup , from k64 to host , send first zero before data.

Another concolusion is that the endianes is kept the same from k64 and Windows Host.

EDMA Example

Load the dma example , and lets do the same , send and receive data from out c# host application

Mofified quicly the same change to move to UART1

pin\_mux.c

**void** **BOARD\_InitPins**(**void**)

{

CLOCK\_EnableClock(*kCLOCK\_PortB*); /\* Port B Clock Gate Control: Clock enabled \*/

CLOCK\_EnableClock(*kCLOCK\_PortC*);

PORT\_SetPinMux(PORTB, PIN16\_IDX, *kPORT\_MuxAlt3*); /\* PORTB16 (pin E10) is configured as UART0\_RX \*/

PORT\_SetPinMux(PORTB, PIN17\_IDX, *kPORT\_MuxAlt3*); /\* PORTB17 (pin E9) is configured as UART0\_TX \*/

PORT\_SetPinMux(PORTC, PIN3\_IDX, *kPORT\_MuxAlt3*); /\* UART1\_RX \*/

PORT\_SetPinMux(PORTC, PIN4\_IDX, *kPORT\_MuxAlt3*); /\* UART1\_TX \*/

SIM->SOPT5 = ((SIM->SOPT5 &

(~(SIM\_SOPT5\_UART0TXSRC\_MASK))) /\* Mask bits to zero which are setting \*/

| SIM\_SOPT5\_UART0TXSRC(SOPT5\_UART0TXSRC\_UART\_TX) /\* UART 0 transmit data source select: UART0\_TX pin \*/

);

}

In Main add the debug console to see PRINTF

BOARD\_InitPins();

BOARD\_BootClockRUN();

BOARD\_InitDebugConsole();

The EDMA shows the same problem

1. Data is inverted
2. First TX is zero

One thing I can answer:

If the first command issue after reset is TX from micro controller to host , a extra zero will be send

If a command is first from Host to K64 and then we TX back , then we will not see the extra zero.