Application Note

BRT_AN_062

Porting BRT_AN_025 code to an NXP K64 MCU

Version 1.0

Issue Date: 22-04-2024

This application note shows how to port the EVE code examples given in BRT_AN_025 to an NXP Kinetis K64 MCU on a FRDM K64 evaluation board.

Application Note BRT_AN_025 contains example code projects for a range of MCUs, allowing you to interface them to an EVE powered display with touch screen attached. This code was designed to be easily ported to other MCU types too. This document uses the NXP device through the Kinetis Design Studio toolchain as an example, but the same technique can be used for porting the code to other MCUs and toolchains.

Bridgetek Pte Ltd (BRTChip)
1 Tai Seng Avenue, Tower A, #03-05, Singapore 536464
Tel: +65 6547 4827
Web Site: http://www.brtchip.com

Table of Contents

1 Introduction

1.1 Overview

This application note shows how to port the EVE code examples given in BRT_AN_025 to an NXP Kinetis K64 MCU on a FRDM K64 evaluation board.

BRT_AN_025 contains example code projects for a range of MCUs, allowing you to interface them to an EVE touchscreen display. This code was designed to be easily ported to other MCU types too. Most of the code and header files are generic and the code changes required to support other MCUs are primarily within the MCU-specific C file as well as in some cases the main.c file.

This document uses the NXP device through the Kinetis Design Studio toolchain as an example, but the same technique can be used for porting the code to other MCUs and toolchains. It can be used with almost any MCU but please check that the MCU SPI Master can support the modes detailed in section 1.4**.**

1.2 Scope

This application note focuses on the process of porting the code, and particularly the edits required to the MCU layer of the code. It does not cover the overall BRT_AN_025 framework and demo application or the low-level SPI protocol used by EVE. For these topics, the following documents can be referenced. They are available from our application notes page at the links below.

- BRT_AN_025 EVE Portable MCU Library
  EVE framework and example which is portable across MCU types. The code from BRT_AN_025 is used as the basis for this porting example.

- BRT_AN_008 FT81x Creating a Simple Library For PIC MCU
  Background information only - Shows how to create a library framework for an MCU using principles discussed in BRT_AN_006 FT81x Simple PIC Example Introduction. This application note takes the basic SPI transfers detailed in BRT_AN_006 and adds an additional layer on top to allow the main application to use more user-friendly commands similar to the EVE Programmers guide.

- BRT_AN_006 FT81x Simple PIC Example Introduction
  Background information only - Explains the low-level SPI transfers used and the different interfaces to transfer data to/from EVE (including RAM_DL, RAM_CMD, RAM_G, Registers). This includes the data formatting used in the SPI transfers for addressing and data, and also the operation of the co-processor FIFO and display list and how these lead to the final screen content.

1.3 Compatibility

The application note uses the code from BRT_AN_025 (EVE Portable MCU Library) and is targeted to the NXP K64 MCU from the Kinetis family. The FRDM K64 demo board was used along with an EVE module (such as the ME813A-WH50C, VM816C50A-D or ME817EV with display).

The compiler used is the Kinetis Design Studio 3 which is available from the NXP website.

1.4 Prerequisites

This code can be used on a wide range of MCUs. Some minimum requirements for the MCU are listed below:

- The MCU requires a Single or Quad Master SPI module with SPI Mode 0 capability.
- SPI signals used are SCK and MOSI/MISO (or four bi-directional data lines for Quad SPI)
- GPIO line for Chip Select signal to EVE
- GPIO line for Power Down signal to EVE

The BRT_AN_025 code is designed for SPI master routines which can read/write a single byte at a time and have manual control over chip select. It is therefore required to have access to the SPI peripheral and the line used for chip select rather than using higher level SPI APIs which some MCU toolchains use. It would require significant modification if the MCU uses an SPI API library which sends a buffer of bytes with automatic chip select control and this is out with the scope of this document. Most MCUs can however be programmed at a level which interacts directly with the SPI hardware registers and can use a GPIO for chip select.

Note: This code is intended to act as a starting point for customers to create their own application rather than being a complete library package. It is necessary that developers of the final application incorporating this library review all layers of the code as part of their product validation. By using any part of this code, the customer agrees to accept full responsibility for ensuring that their final product operates correctly and complies with any operational and safety requirements and accepts full responsibility for any consequences resulting from its use.

Note: The library functions are intended to perform a basic set-up of the MCU so that the EVE functionality can be demonstrated. The reader must consult the product documentation provided by the manufacturer of their selected MCU, and the Bridgetek documentation for their EVE device, to confirm that their final code and hardware complies with all recommendations, best practices, and specifications, so that reliable operation of the final product can be assured. The information provided in this document and code is not intended to override any information or specifications in the product datasheets.

2 Getting Started

2.1 Install the IDE and Development Board

It is recommended to install the IDE for the chosen MCU and connect the MCU development board to the computer and to build and flash one of the basic samples (for example many boards are provided with a sample code which flashes an LED on the board etc.) provided by the manufacturer. This is a good way to ensure that the IDE, board, and debugger are all working well.

2.2 Familiarisation with BRT_AN_025

Next, it is recommended to download the BRT_AN_025 code and familiarise yourself with the code structure. The code can be downloaded from:

https://github.com/Bridgetek/EVE-MCU-BRT_AN_025 (external link)

The accompanying application note BRT_AN_025 has an explanation of the folder structure and the files within. A brief description is given below but please refer to BRT_AN_025 chapter 2.3 onwards for details of the folder structure.

BRT_AN_025 has four main code folders:

- Examples
    - Project files for each MCU IDE (MCU-specific, folder for each MCU type)
    - Source and header files for the application itself, a simple digital counter example (common to all MCUs)
- Include
    - Header files used by BRT_AN_025 (common to all MCUs)
- Source
-
    - Source code files for the middle layers of the BRT_AN_025 library (common to all MCUs)
- Ports
    - Contains the low-level SPI and GPIO code for each MCU. (MCU-specific, folder for each MCU type)

The reader is encouraged to review the files for one of the existing platforms (for example the FT900 or PIC) at this stage to get an idea of its functionality.

```
EVE-MCU-BRT_AN_025
↳  examples
    ↳  simple
        ↳  common
            ↳  eve_calibrate.c
            ↳  eve_example.c
            ↳  eve_example.h
            ↳  eve_fonts.c
            ↳  eve_helper.c
            ↳  eve_images.c
        ↳  ESP32
            ↳  [toolchain project files for this platform]
        ↳  ft900
            ↳  [toolchain project files for this platform]
        ↳  Libmpsse
            ↳  [toolchain project files for this platform]
        ↳  MSP430
            ↳  [toolchain project files for this platform]
```

```
        ↳ PIC18F
            ↳ [toolchain project files for this platform]
        ↳ pico
            ↳ [toolchain project files for this platform]
        ↳ raspberry_pi
            ↳ [toolchain project files for this platform]
        ↳ STM32
            ↳ [toolchain project files for this platform]
        ↳ BeagleBone
            ↳ [toolchain project files for this platform]
↳ include
    ↳ EVE.h
    ↳ EVE_config.h
    ↳ FT8xx.h
    ↳ HAL.h
    ↳ MCU.h
    ↳ Platform.h
↳ ports
    ↳ eve_arch_ft9xx
        ↳ EVE_MCU_FT9XX.c
    ↳ eve_arch_pic
        ↳ EVE_MCU_PIC.c
    ↳ eve_arch_stm32
        ↳ EVE_MCU_STM32.c
    ↳ eve_arch_esp32
        ↳ EVE_MCU_ESP32.c
    ↳ eve_arch_msp430
        ↳ EVE_MCU_MSP430.c
    ↳ eve_arch_beaglebone
        ↳ EVE_Linux_BBB.c
    ↳ eve_arch_rpi
        ↳ EVE_Linux_RPi.c
        ↳ EVE_MCU_RP2040.c
    ↳ eve_libmpsse
        ↳ EVE_libmpsse.c
↳ source
    ↳ EVE_API.c
    ↳ EVE_HAL.c
    ↳ EVE_HAL_Linux.c
```

2.3 The code layers of BRT_AN_025

The BRT_AN_025 library is created using several layers of code to make it more easily ported and edited.

The main application is where the user will add the lists of commands used to build the desired screen content, in the syntax of the EVE Programmers Guide.

The API layer is responsible for taking these commands called by the main application and translating them into a series of 32-bit and 8-bit values which encode the command and parameters.

The HAL layer then translates these calls into a series of SPI transfers (controlling chip select and data) to match the API used by EVE over SPI. This layer then calls the functions in the MCU layer below to send and receive the data.

The MCU layer contains the MCU-specific parts of the code. The HAL layer and API layer above simply request a value to be sent/received or a chip select line to be set or cleared using a pre-defined set of function calls. The MCU layer has the code added to these functions to handle the low-level register accesses needed to do this on the specific MCU.

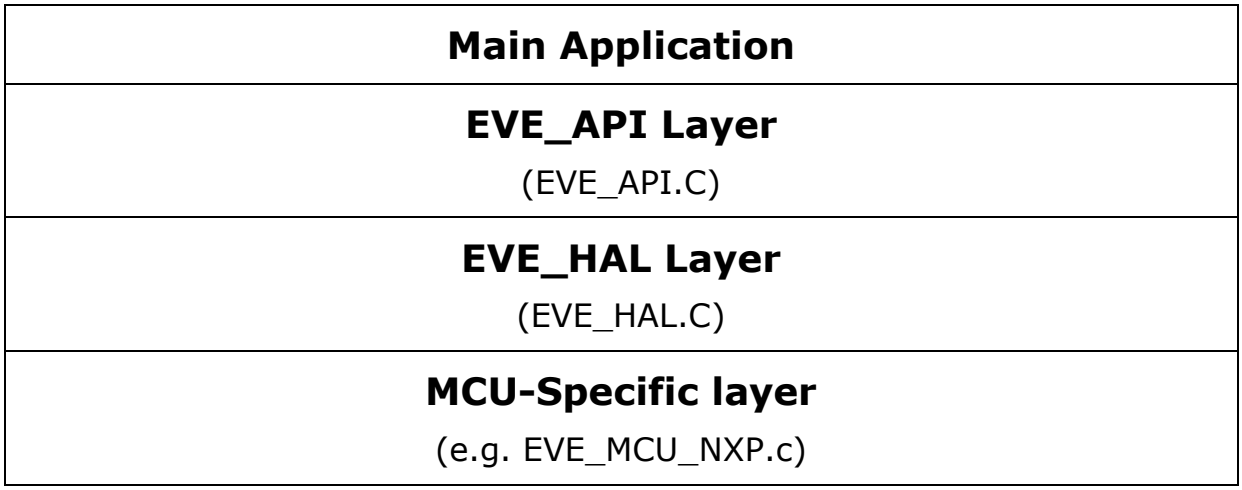


Figure 1 - Layers of the software example

2.4 The MCU Layer

The following functions will be dependent on the MCU:

`void MCU_Init(void)`

This function is called after starting up and so should contain the following-

- Any general code needed to initialise the MCU such as setting the watchdog, oscillator, MCU memory, UART, or any other configuration tasks needed by the application.

- Configure two GPIO outputs for the EVE CS and PD lines. Check the MCU documentation and check the schematic of the MCU board to ensure these lines are able to be set as general purpose outputs (e.g. some analog lines may be input only) and that the lines are not connected to another device on the MCU demo board which may cause contention (e.g. if the board has a switch or output from a sensor which is trying to drive this line as an input).

- Configure the SPI Master of the MCU for SPI mode 0 and any pin mixing settings needed to allow it to access the pins of the MCU. Check that this will not conflict with any other peripherals on the board (such as an SPI flash or other peripheral). It may be possible to use the SPI Master even if it is linked to other peripherals on the board so long as the chip selects are de-asserted for the other peripherals. However, check the product documentation to confirm that de-asserting the CS will be sufficient to avoid them having any effect on the lines which may affect the communication with EVE.

- The SPI clock rate must be below 11MHz initially but can then be increased. Consult the documentation for the version of EVE which you will use but in general the SPI rate used must always be less than half of the clock speed (e.g. if running at 60MHz on FT81x, the SPI must be below 30MHz). However, many applications can use much lower SPI clock rates and still give more than required performance, whilst being less sensitive to the wiring used for the SPI and susceptibility/emission of noise etc.

- It is recommended that any wires/connections between EVE and the MCU for the SPI signals are as short as possible. Using lower SPI data rates and setting the slew rate control on the pins to give stronger drive may help reduce data errors.

`void MCU_Setup(void)`

This function is called from EVE_HAL.c and is used to enable the quad-SPI mode on an FT900 MCU. In most platforms (particularly those which do not support Quad SPI) it can be left blank and will simply return when called.

void MCU_CSlow(void)

This function should have a GPIO pin write to put the GPIO pin assigned as chip select to a low state. If the MCU or the hardware conditions require a small delay after enabling CS, then some MCU NOP commands can be added after putting the pin low.

void MCU_CShigh(void)

This function should have a GPIO pin write to put the GPIO pin assigned as chip select to a high state.

void MCU_PDlow(void)

This function should have a GPIO pin write to put the GPIO pin assigned as power down to a low state.

void MCU_PDhigh(void)

This function should have a GPIO pin write to put the GPIO pin assigned as power down to a high state.

uint8_t MCU_SPIReadWrite8(uint8_t DataToWrite)

This function should include all the register writes/reads needed to clock one byte (DataToWrite) out of the MOSI pin and at the same time clock in one byte on MISO. The byte should be returned from the function call. Consult the MCU documentation and examples for the recommended sequence of register writes and reads to ensure all SPI module flags are checked and cleared correctly as this can vary a lot between MCU types.

void MCU_Delay_20ms(void)

This function should provide a delay for a minimum of 20ms before returning. For example, it may have a loop using a series of NOP instructions for the MCU or the MCU may have other recommended ways of creating a delay such as using timers.

void MCU_Delay_500ms(void)

This function should provide a delay for a minimum of 500ms before returning. In many cases calling the 20ms delay function 25 times will give the desired result (e.g. if the 20ms delay is created by a loop using NOPs). Note that this function was originally used for the 500ms start-up delay but newer versions of the BRT_AN_025 code have this commented out and use the check of REG_CPURESET == 0 instead.

Note: The structure of the BRT_AN_025 code can be changed to suit your main application, but it is suggested to port the original code in as shown in this application note and check that the basic demo works before doing so.

3 Preparing the SPI Master and GPIO

It is useful to check at this stage that all the required functions mentioned in the previous section are working as expected.

To do this, a small project was created which uses all of these. This was compiled, and loaded into the MCU, and a USB-based logic analyser was used to confirm that all the function calls had the desired effect.

The project was created using the new project wizard in the Kinetis Design Studio IDE. Please consult the documentation for the MCU and IDE to get the recommended procedure for creating a new project for the chosen MCU and development board. Many boards are provided with example projects which are pre-configured to work with the development board. It is also necessary to check the manufacturers documentation for any required set-up of the flash programming and debugging interface.

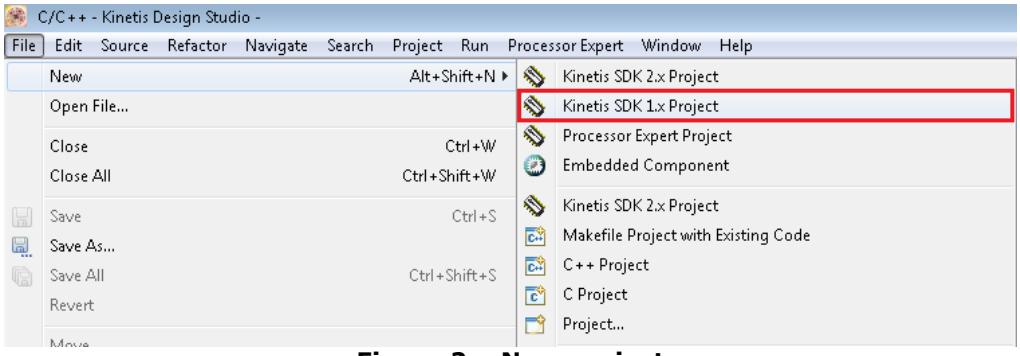


Figure 2 – New project

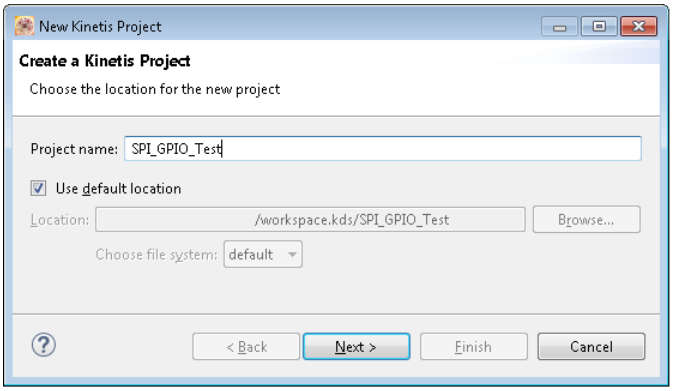


Figure 3 – Setting up new project name and location

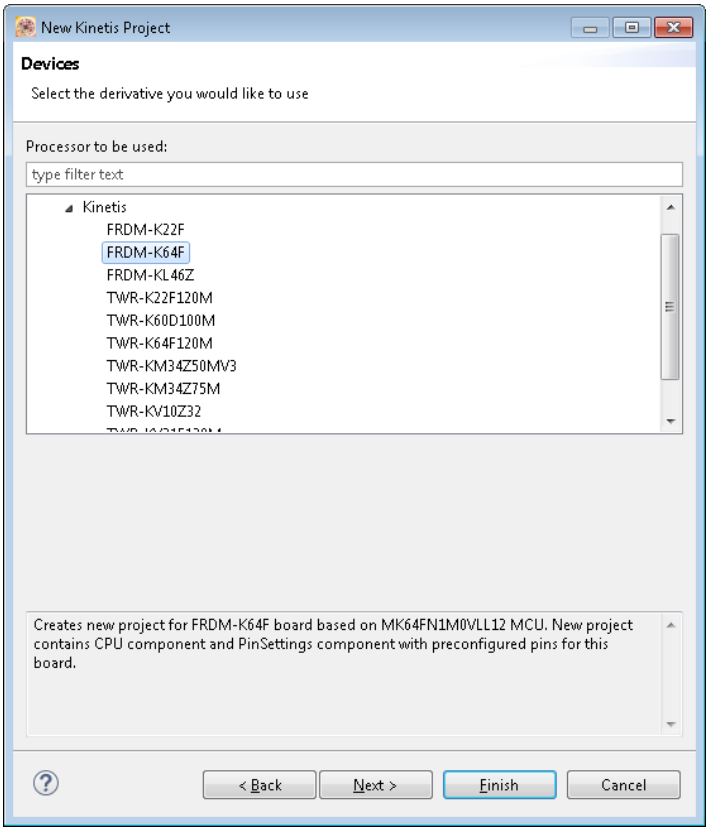


Figure 4 - Layers of the software example

A project is created by the wizard with the standard code inside as shown below-

```
#include "MK64F12.h"

static int i = 0;

int main(void)
{

    /* Write your code here */

    /* This for loop should be replaced. By default, this loop allows a single stepping.
*/
    for (;;) {
        i++;
    }
    /* Never leave main */
    return 0;
}
////////////////////////////////////////////////////////////////////////////////
// EOF
////////////////////////////////////////////////////////////////////////////////
```

This code was edited as explained in section 2.4 and the resulting code is shown below. Note that this code is for illustration purposes and so has a basic minimal set-up. Please ensure to follow the recommended settings when configuring your MCU.

```
#include "MK64F12.h"

void MCU_Init();
void MCU_CSlow();
void MCU_CShigh();
void MCU_PDlow();
void MCU_PDhigh();
uint8_t MCU_SPIReadWrite8(uint8_t DataToWrite);
void MCU_Delay_20ms();
```

```
void MCU_Delay_500ms();


int main(void)
{
	uint8_t SPI_rxd = 0;

	MCU_Init();

	MCU_CShigh();


	MCU_PDlow();
	MCU_Delay_20ms();
	MCU_PDhigh();

	MCU_Delay_20ms();

	MCU_PDlow();
	MCU_Delay_500ms();
	MCU_PDhigh();

	while(1)
	{
		MCU_CSlow();

		SPI_rxd = MCU_SPIReadWrite8(0x55);

		MCU_CShigh();
	}

	return 0;
}

// -------------------- MCU specific initialisation  ----------------------------
void MCU_Init(void)
{
	// Write 0xC520 to the unlock register
	WDOG_UNLOCK = 0xC520;

	// Followed by 0xD928 to complete the unlock
	WDOG_UNLOCK = 0xD928;

	// Clear the WDOGEN bit to disable the watchdog
	WDOG_STCTRLH &= ~WDOG_STCTRLH_WDOGEN_MASK;

	//configure ports
	// Set port D for SPI Master
	SIM_SCGC5 |= SIM_SCGC5_PORTD_MASK;	// Enable PORT clock gating ctrl
	SIM_SCGC6 |= SIM_SCGC6_SPI1_MASK;	// Enable SPI1

	GPIOD_PSOR = 0x00000010;		// Set bit 4 of port D

	PORTD_PCR(4) = PORT_PCR_MUX(1);		// PCS0
	PORTD_PCR(5) = PORT_PCR_MUX(7);		// SCK
	PORTD_PCR(6) = PORT_PCR_MUX(7);		// SOUT
	PORTD_PCR(7) = PORT_PCR_MUX(7);		// SIN
	GPIOD_PDDR |= 0x00000010;		// Port D direction register

	// set port B20 as PD line
	SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK;	// Enable PORT clock gating ctrl

	GPIOD_PSOR = 0x00000010;		// Set bit 4 of port D

	PORTB_PCR(20) = PORT_PCR_MUX(1);	// SIN
	GPIOB_PDDR |= 0x00100000;		// PD pin

	// Shut down the SPI Master to initialise it
```

```
SPI1_MCR |= SPI_MCR_HALT_MASK;

// Set up the SPI1 CTAR0
SPI1_CTAR0 = 0;
SPI1_CTAR0 |= 0x38000001;  // 0011 1000 0000 0000 0000 0000 0000 0000
// 31         DBR           = 0    Double Baud rate off
// 30:27      FMSZ 3:0      = 0111 Frame Size 0111 means frame is 8 bits
// 26         CPOL          = 0    Set Mode 0 SPI, Clock idle low
// 25         CPHA          = 0    Set Mode 0 SPI,
// 24         LSBFE         = 0    LSB First bit, not set
// 23:22      PCSSCK        = 00   PCS to SCK divider (00 means 1)
// 21:20      PASC          = 00   PCS after SCK divider (00 means 1)
// 19:18      PDT           = 00   Delay after Transfer Prescaler (00 means 1)
// 17:16      PBR           = 00   Baud rate Prescaler (00 = 2)
// 15:12      CSSCK         = 0000 PCS to SCK Delay Scaler (0000 = 2)
// 11:08      ASC           = 0000 After SCK Delay Scaler
// 07:04      DT            = 0000 Delay after Transfer Scaler
// 03:00      BR            = 0001 Baud rate scaler (set to divide by 4)

SPI1_RSER = 0;
SPI1_TCR = 0;

// Set up the SPI1 SR
SPI1_SR = 0;
SPI1_SR |= 0xda0a0000;// 1 1 0 1 1 0 1 0 0 0 0 0 1 0 1 0 0000 0000 0000 0000
// 31         TCF           = 1    Transfer Complete flag
// 30         TXRXS         = 1    Tx and Rx Status
// 29         RESERVED      = 0
// 28         EOQF          = 1    End of Queue Flag
// 27         TFUF          = 1    Tx FIFO Underflow flag
// 26         RESERVED      = 0
// 25         TFFF          = 1    Tx FIFO Full flag
// 24         RESERVED      = 0
// 23         RESERVED      = 0
// 22         RESERVED      = 0
// 21         RESERVED      = 0
// 20         RESERVED      = 0
// 19         RFOF          = 1    Rx FIFO Overflow flag
// 18         RESERVED      = 0
// 17         RFDF          = 1    Rx FIFO Drain flag
// 16         RESERVED      = 0
// 15:12      TXCTR         = 0000 Tx FIFO Counter
// 11:8       TXNXTPTR      = 0000 Tx Next Pointer
// 7:4        RXCTR         = 0000 Rx FIFO Counter
// 3:0        POPNXTPTR     = 0000 Pop Next Pointer

// Set up the SPI1 MCR
SPI1_MCR =0;
SPI1_MCR |= 0x80013c01;//1 0 00 0 0 0 0 00 000001 0 0 1 1 1 1 00 00000 0 0 1
// 31         MSTR          = 1    Master/Slave mode select, set Master
// 30         CONT_SCKE     = 0    Continuous SCK enable/disable
// 29:28      DCONF         = 00   SPI Configuration
// 27         FRZ           = 0    Freeze
// 26         MTFE          = 0    Modified Timing Format
// 25         PCSSE         = 0    Peripheral Chip Select Strobe Enable
// 24         ROOE          = 0    Rx FIFO Overflow Overwrite enable
// 23:22      RESERVED      = 00   Reserved
// 21:16      PCSIS         = 000001 Peripheral Chip Select x Inactive State
// 15         DOZE          = 0    Doze power saving enable/disable
// 14         MDIS          = 0    Module Disable
// 13         DIS_TXF       = 1    Disable Tx FIFO
// 12         DIS_RXF       = 1    Disable Rx FIFO
// 11         CLR_TXF       = 1    Clear Tx FIFO
// 10         CLR_RXF       = 1    Clear Rx FIFO
// 9:8        SMPL_PT       = 00   Sample Point
// 7:3        RESERVED      = 00000             Reserved
// 2          RESERVED      = 0    Reserved
// 1          RESERVED      = 0    Reserved
```

```
        // 0          HALT          = 1    Halt
}

// ---------------------- Chip Select line low ---------------------------------
void MCU_CSlow(void)
{
        uint8_t dly = 0;

        for(dly =0; dly < 5; dly++)
    {
        __asm__ __volatile__ ("nop");
    }

        GPIOD_PCOR = 0x00000010;   // clear bit 4 of port D

        for(dly =0; dly < 5; dly++)
    {
        __asm__ __volatile__ ("nop");
    }
}


// ---------------------- Chip Select line high --------------------------------
void MCU_CShigh(void)
{
        uint8_t dly = 0;

        for(dly =0; dly < 5; dly++)
    {
        __asm__ __volatile__ ("nop");
    }

        GPIOD_PSOR = 0x00000010;   // Set bit 4 of port D

        for(dly =0; dly < 5; dly++)
    {
        __asm__ __volatile__ ("nop");
    }
}


// -------------------------- PD line low -------------------------------------
 void MCU_PDlow(void)
{
    GPIOB_PCOR = 0x00100000;      // Clear bit 20 of port B
}

// -------------------------- PD line high ------------------------------------
 void MCU_PDhigh(void)
{
    GPIOB_PSOR = 0x00100000;      // Set bit 20 of port B
}




// ---------------------- SPI Send and Receive ---------------------------------
 uint8_t MCU_SPIReadWrite8(uint8_t DataToWrite)
 {
        uint8_t DataRead[4];

        // ------- Clear flags and configure --------
        SPI1_MCR &= ~(SPI_MCR_HALT_MASK | SPI_MCR_FRZ_MASK);
        // Un-Halt and Un-Freeze the SPI Master
        SPI1_MCR |= SPI_MCR_CLR_RXF_MASK;
                                                                    // Clear
the RxF flag
        SPI1_SR = (SPI_SR_EOQF_MASK | SPI_SR_TFUF_MASK | SPI_SR_TFFF_MASK |
```

```
                SPI_SR_RFOF_MASK | SPI_SR_RFDF_MASK);    // Clear flags

        // -------- Clock out one byte and await for one byte to come back --------
        SPI1_PUSHR = (SPI_PUSHR_EOQ_MASK | DataToWrite | SPI_PUSHR_CTAS(0) |
                SPI_PUSHR_PCS(1));  // Write the data byte to the PUSHR register

        // Wait until the byte is received back
        while ((SPI1_SR & SPI_SR_RFDF_MASK) == 0)
        {
        }
        DataRead[0] = SPI1_POPR;   // Read the received byte

        // Clear flags
        SPI1_SR = (SPI_SR_EOQF_MASK | SPI_SR_TFUF_MASK | SPI_SR_TFFF_MASK |
                SPI_SR_RFOF_MASK | SPI_SR_RFDF_MASK);
        // Halt and Freeze the SPI Master
        SPI1_MCR |= (SPI_MCR_HALT_MASK | SPI_MCR_FRZ_MASK);

        return DataRead[0];
 }

void MCU_Delay_20ms(void)
{
        uint32_t dly = 0;

        for(dly =0; dly < 40000; dly++)
        {
                __asm__ __volatile__ ("nop");
        }
}

 void MCU_Delay_500ms(void)
 {
        uint8_t dly = 0;

        for(dly =0; dly < 25; dly++)
        {
                MCU_Delay_20ms();
        }
 }

////////////////////////////////////////////////////////////////////////////////
// EOF
////////////////////////////////////////////////////////////////////////////////
```

The full project is also available at https://github.com/Bridgetek/EVE-MCU-BRT_AN_025/tree/main/examples/BRT_AN_062_Source/NXP_K64 (external link)

As shown in the main() function above, the code will initialise the MCU and then perform the following actions:

- Put the PD line low, wait 20ms, and then put the PD line high again
- After 20ms, put the PD line low again, wait 500ms, and then put the PD line high again
- Loop continuously, carrying out a single byte read/write on SPI
    - CS line low
    - Send 0x55 over SPI and read the MISO line (which will be high)
    - CS line high

After programming this code to the board and running it, the following waveforms were observed. These were used to confirm the following requirements were met.

- Delay 20ms is correct Figure 6
- Delay 500ms is also correct Figure 7

- PD operating as expected in the code Figure 7
- SPI Mode 0 Figure 8
- CS operating as expected in the code Figure 8
- SPI data on MOSI is 0x55 Figure 9
- SPI data on MISO is 0xFF * Figure 9
- SPI rate below 11MHz for start-up Figure 9

Note *: Value is 0xFF as no external device is driving the line. Put a breakpoint and confirm that the value read by the function is 0xFF and you can also pull MISO low and confirm that 0x00 is received or loop MOSI to MISO)

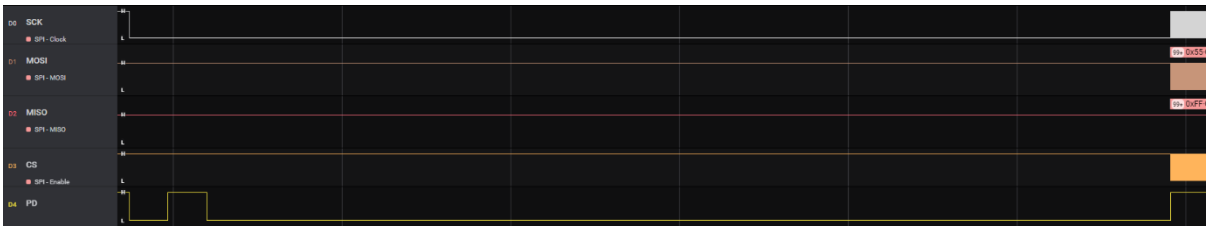


Figure 5 - Overall waveform

Figure 6 - Delay of at least 20ms

Figure 7 - Delay of at least 500ms

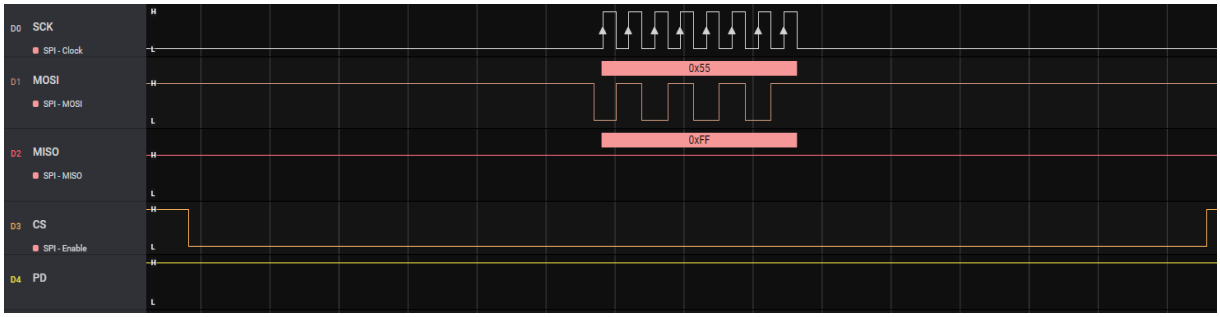


Figure 8 - Chip Select operation and confirm SPI Mode 0

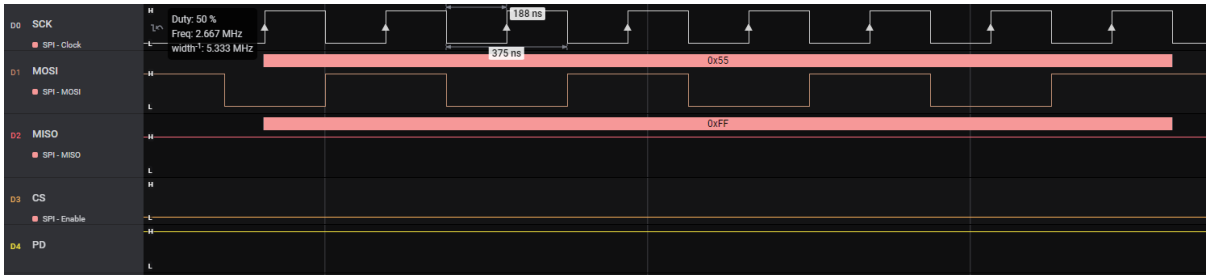


Figure 9 - 0x55 on MOSI and 0xFF on MISO, SPI Rate 2.667MHz

4 Creating the full code project

Download the BRT_AN_025 code package from the GitHub page below. For example, you can download it as a zip file by selecting the green Code button and choosing Download ZIP.
https://github.com/Bridgetek/EVE-MCU-BRT_AN_025/

Note that this package includes the full NXP K64 version of the project which is being created here and can be used as a reference whilst reading this document.

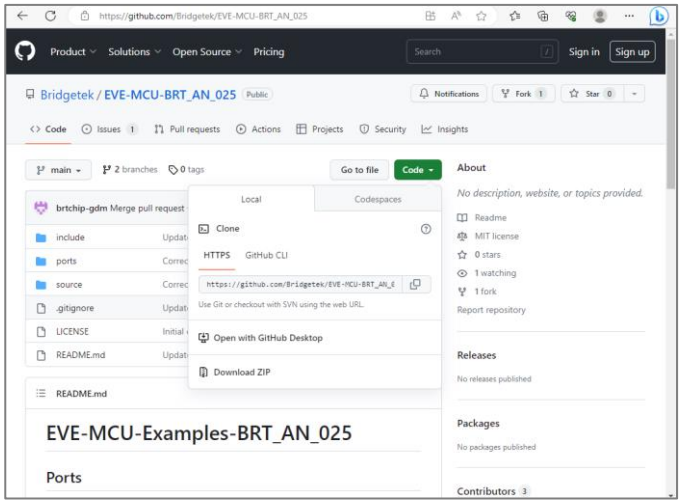


Figure 10 - Downloading the code from GitHub

In this case we downloaded and unzipped it to the workspace for the NXP K64 Kinetis IDE.

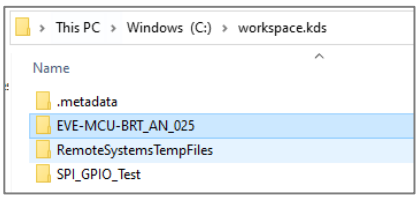


Figure 11 - Saving code to workspace

Create a folder in the examples -> Simple folder for your new MCU. In this case we created folder NXP_K64. This is where the MCU project will go for this platform.

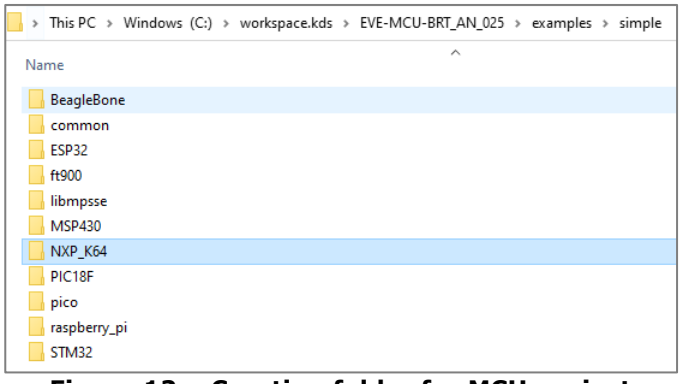


Figure 12 – Creating folder for MCU project

Also, create a folder within ports, which is where the MCU-specific file based on the SPI and GPIO test carried out in chapter 3 will go.

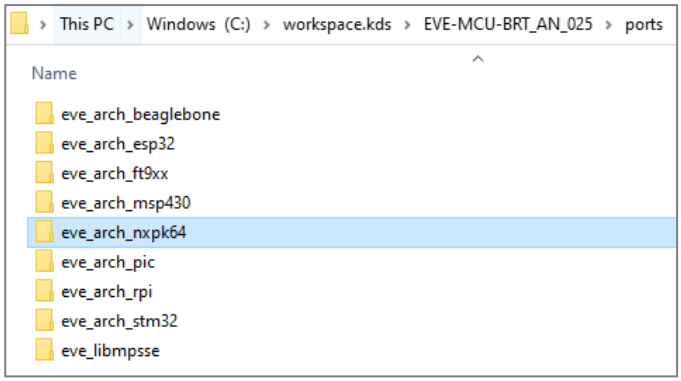


Figure 13 - Creating folder for the platform files

Repeat the steps from chapter 3 to create a new project named suitably for your platform (in this case we used "NXP_K64").

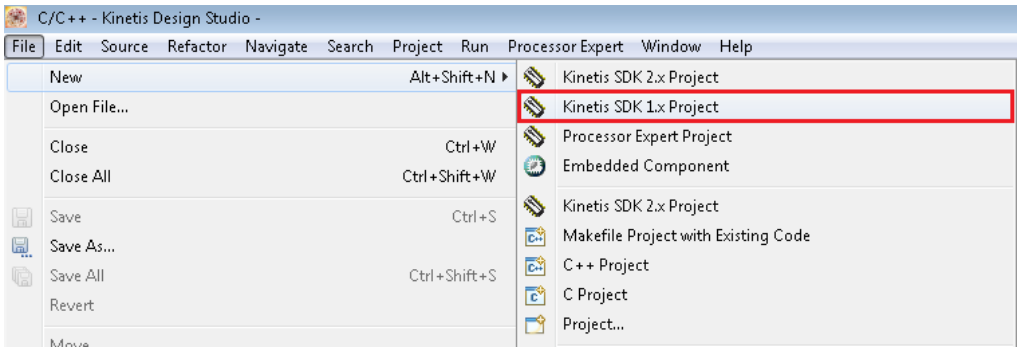


Figure 14 – New project

Note: The library layers were the latest ones at the time of writing but the BRT_AN_025 code may be updated after this time. Therefore, it is recommended to check the BRT_AN_025 Github repository and repeat the porting process using the latest files, to ensure your code has all the enhancements of the latest version.

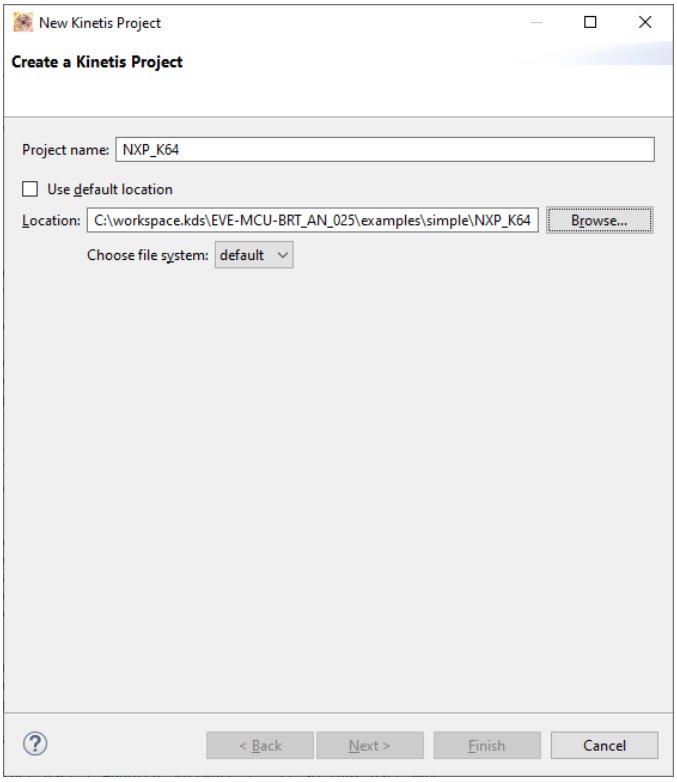


Figure 15 - New project for this porting example

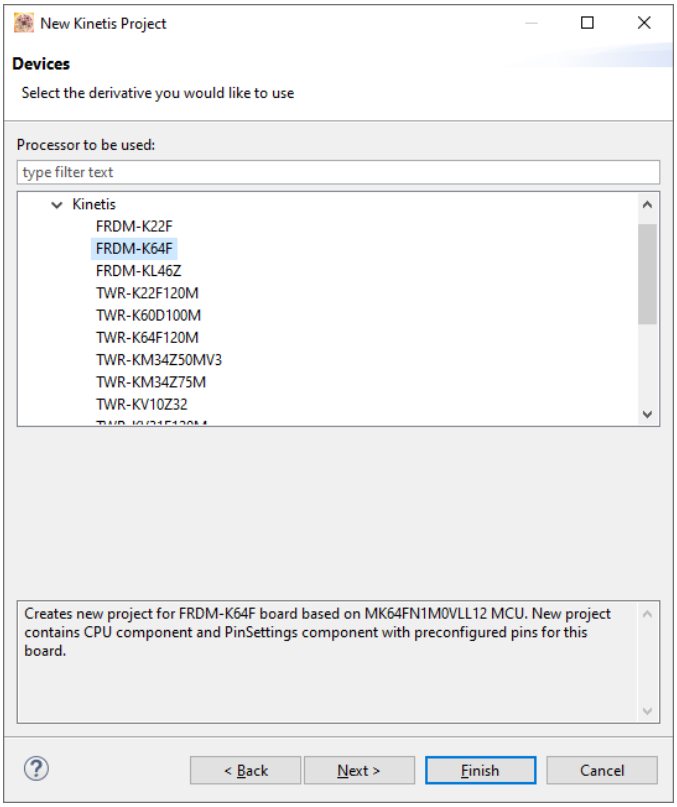


Figure 16 - Selecting the FRDM-K64F board

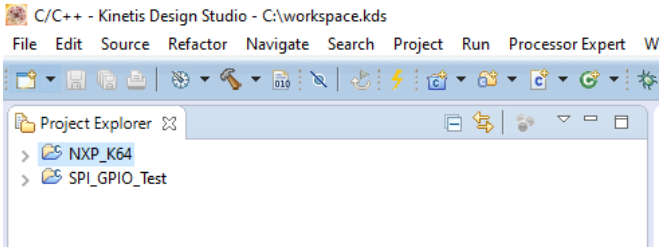


Figure 17 - New project created

4.1 Adding the BRT_AN_025 files to the Project

The next step is to add the common BRT_AN_025 EVE files into the project. These are already located in the folder structure (see chapter 2.2).
In many IDEs, particularly Eclipse based ones, the folders can be added via the Paths and Symbols menu.

In the Kinetis Design Studio, right-click the project in the left-hand tree view and choose Properties > C/C++ General > Paths and Symbols > GNU C

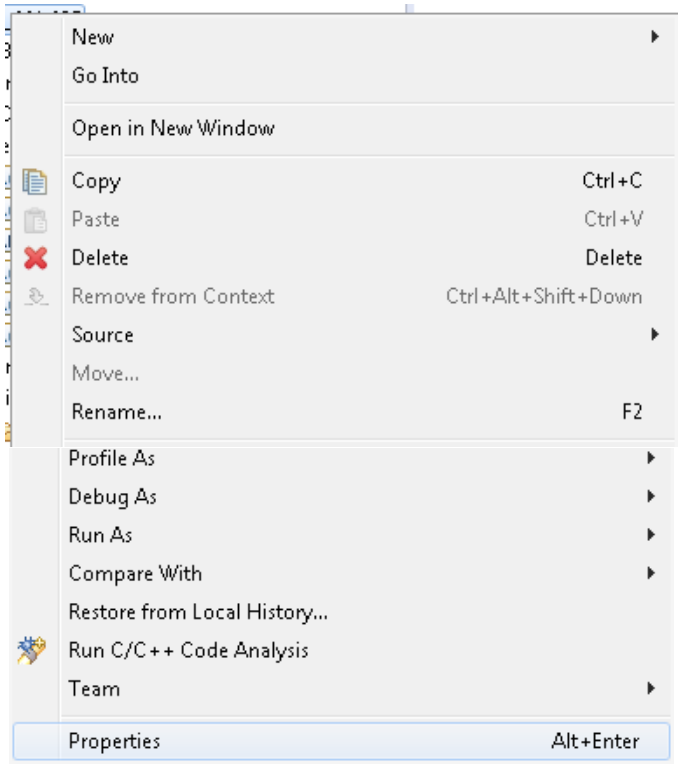


Figure 18 – Project properties

The following folders both have header files and so are added under Includes (see Figure 19).

- \EVE-MCU-BRT_AN_025\include
- \EVE-MCU-BRT_AN_025\simple\common

The include folders can be added as absolute paths or as relative paths to the project itself (as shown here)

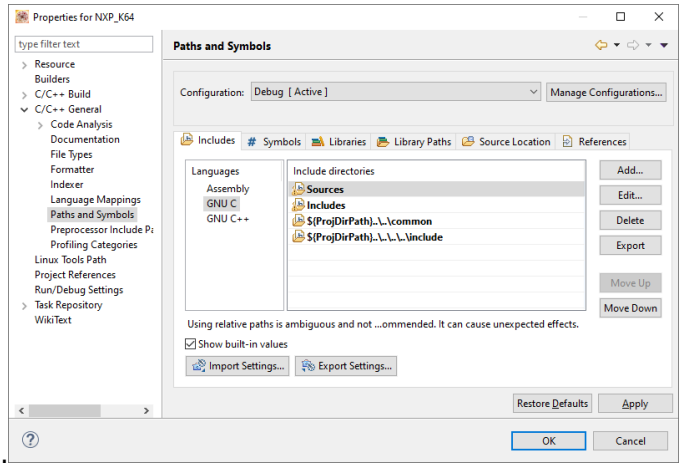


Figure 19 - Setting the Includes

Some IDEs allow you to specify the folders containing the source files within the Resource -> Linked Resources menu. This is shown in Figure 20. The source files are in:

- \EVE-MCU-BRT_AN_025\source
- \EVE-MCU-BRT_AN_025\simple\common
- \EVE-MCU-BRT_AN_025\ports\[folder for your specific MCU type]
-

You can either add the folders containing the source files in the dialog shown in Figure 20, or you can drag the folders into the IDE device tree instead and they will be added (see Figure 21). Some IDEs may also have other options or ways to add the files.

As with the include files, source files can be added as absolute or relative links.

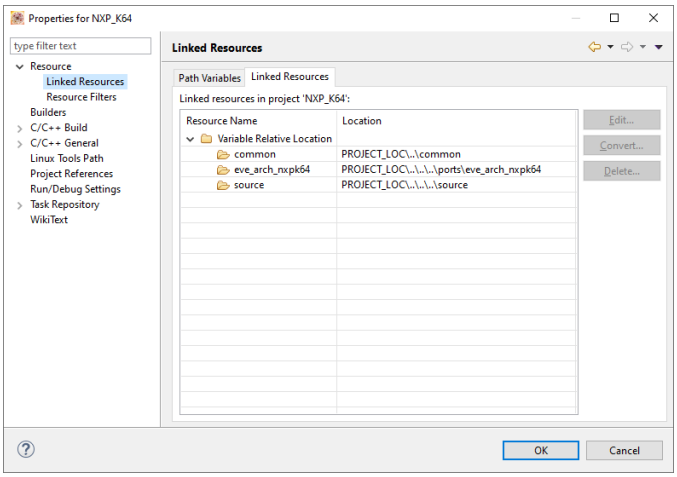


Figure 20 - Setting the Source folders

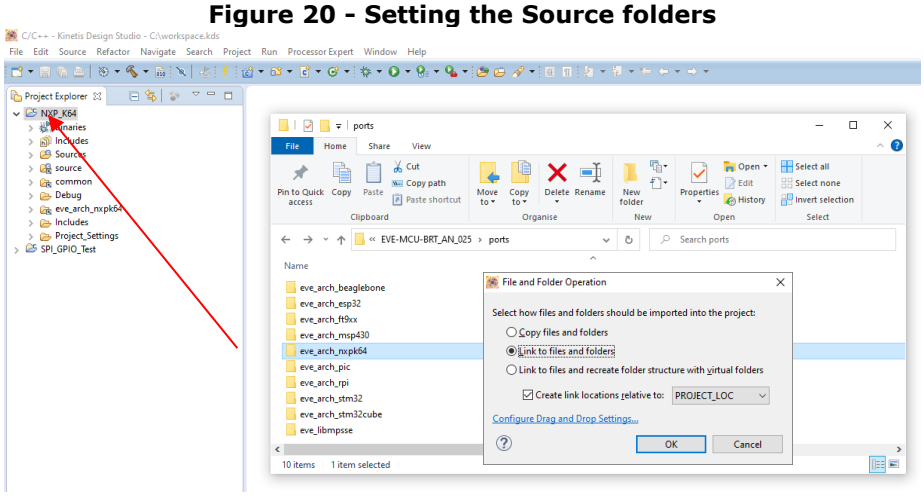


Figure 21 - Dragging the folders into the IDE

Whilst in the Properties menu, it is also necessary to set a preprocessor symbol to gate the MCU-specific code. This code will be shown later in this guide. Here, PLATFORM_NXPK64 is added as a symbol. Click OK and go back to the main window. Ensure all files are saved.

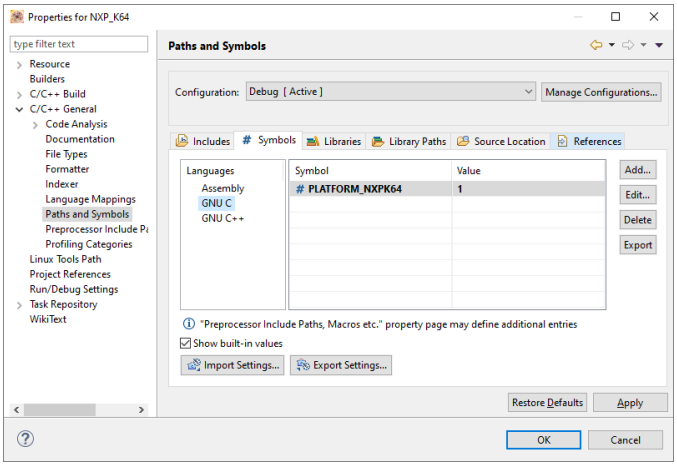


Figure 22 - Setting the platform symbol

Some IDEs may have different ways of adding references to the files. If the IDE does not use a similar method to that shown above, please check the documentation for the IDE to see the recommended way of adding files to the project.

Since this platform is not a Linux platform, you may also wish to exclude the EVE_HAL_Linux.c file so that it is not included in the build. Note that the file is gated by the following '#if defined' and so should normally not affect the build in any case.

#if defined(USE_LINUX_SPI_DEV) && !defined(USE_MPSSE)

Likewise, if this was a Linux platform, you could include the EVE_HAL_Linux.c and exclude the EVE_HAL.c instead.

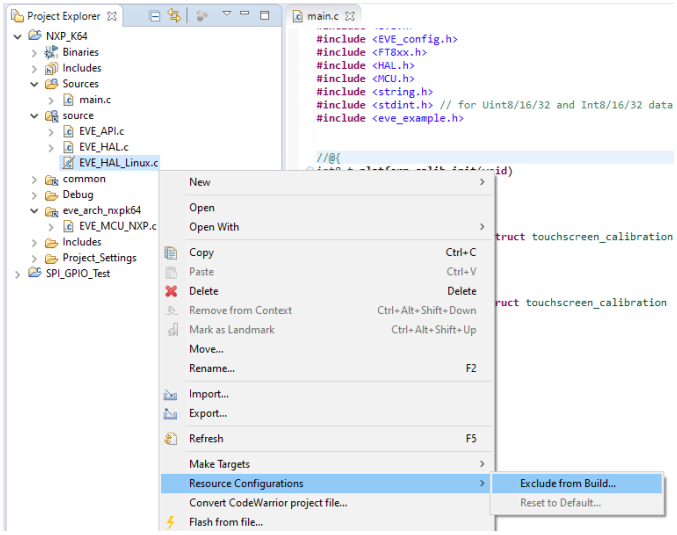


Figure 23 - Excluding the EVE_HAL_Linux.c file

4.2 Folder Structure

The diagram below shows the header files (green outline) and source files (orange outline) which would be added. The main.c and EVE_MCU_NXP.c are covered in the following sections of this document.

Some MCUs may use a makefile instead and so in this case please refer to the ESP32, RP2040 Pico, Raspberry Pi, and BeagleBone Black examples in BRT_AN_025 for examples of how the project and makefile are arranged.

It is possible to use the existing BRT_AN_025 folder structure or to add the files to your own MCU's folder structure. One advantage of using the BRT_AN_025 folder structure is that the same application can be built on each supported platform using the shared main application files such as eve_example.c. It may be required however to add the files to an existing MCU project instead to allow the existing parts of the application (such as the other duties performed by the MCU in the application) to continue to work.

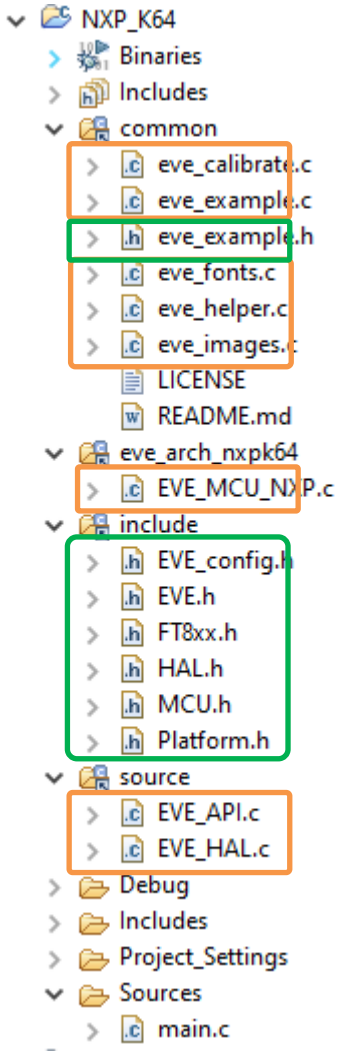


Figure 24 - Source and include files

4.3 Replacing Main.c with the BRT_AN_025 main file

The NXP K64 project creation in the IDE will have created a file main.c (in this case it is in \EVE-MCU-BRT_AN_025\examples\simple\NXP_K64\Sources)

The main.c file can be replaced with the BRT_AN_025 one. One way is to copy from one of the platforms in BRT_AN_025 which have no MCU-specific content. For example, the PIC main.c file at the path below contains only the BRT_AN_025 general content and so could be copied into the location below to replace the existing main.c.

\EVE-MCU-BRT_AN_025\examples\simple\PIC18F\main

Note that it is also possible to merge the BRT_AN_025 content from the other platform (such as the PIC) instead of doing a direct replace, if you need to retain any content from the original main.c file.

```
#define bswap16(x) (((x) >> 8) | ((x) << 8))
#define bswap32(x) (((x) >> 24) | (((x) & 0x00FF0000) >> 8) \
                  | (((x) & 0x0000FF00) << 8) | ((x) << 24))

#include <EVE.h>
#include <EVE_config.h>
#include <FT8xx.h>
#include <HAL.h>
#include <MCU.h>
```

```
#include <string.h>
#include <stdint.h> // for Uint8/16/32 and Int8/16/32 data types
#include <eve_example.h>


//@{
int8_t platform_calib_init(void)
{
    return 1;
}

int8_t platform_calib_write(struct touchscreen_calibration *calib)
{
    return 0;
}

int8_t platform_calib_read(struct touchscreen_calibration *calib)
{
    return -1;
}
//@}

/**
  * @brief  Main program.
  * @param  None
  * @retval None
  */
int main(void)
{

    eve_example();

    /* Infinite loop */
    while (1)
    {
    };

    return 0;
}
```

4.4 Create the Platform file

The platform-specific file EVE_MCU_NXP.c also needs to be created in \EVE-MCU-BRT_AN_025\ports\eve_arch_nxpk64.

In a similar way to the main.c file in the previous section, one way is to copy a file from another platform such as from the PIC folder (\EVE-MCU-BRT_AN_025\ports\eve_arch_pic).

Go to \EVE-MCU-BRT_AN_025\ports\eve_arch_pic and copy the file EVE_MCU_PIC.c. Paste it into the folder BRT_AN_025\ports\eve_arch_nxpk64\
Re-name the file to EVE_MCU_NXP.c

The next step is to edit the EVE_MCU_NXP.c file to set up the NXP device and to interface with its SPI and GPIO module registers.

First, remove the #include <xc.h> line as this was a PIC header file.

The platform define and the associated message can also be edited as shown below.

The include for xc.h can also be replaced with the NXP file "MK64F12.h" which was present in the earlier GPIO test section (See section 3 Preparing the SPI Master and GPIO).

```
// Guard against being used for incorrect CPU type.
#if defined(PLATFORM_PIC)
```

```
#if defined(PLATFORM_NXPK64)

#pragma message "Compiling " __FILE__ " for Microchip PIC"
#pragma message "Compiling " __FILE__ " for NXP K64"

#define bswap16(x) (((x) >> 8) | ((x) << 8))
#define bswap32(x) (((x) >> 24) | (((x) & 0x00FF0000) >> 8) \
                  | (((x) & 0x0000FF00) << 8) | ((x) << 24))

#include <xc.h>
#include "MK64F12.h"
```

After these edits, the next step is to replace the contents of the following functions with the same code developed for the NXP K64 in the SPI_GPIO_Test program (see section 3).

Replace the contents of each of the following functions and leave the other functions unchanged. An example is also provided on the following page.

- void MCU_Init(void)
- void MCU_Setup(void) (see additional note below)
- void MCU_CSlow(void)
- void MCU_Cshigh(void)
- void MCU_Pdlow(void)
- void MCU_Pdhigh(void)
- uint8_t MCU_SPIReadWrite8(uint8_t DataToWrite)
- void MCU_Delay_20ms(void)
- void MCU_Delay_500ms(void)

Note that MCU_Setup is used on some platforms to set QSPI and is not used here. Therefore, keep MCU_Setup as a blank function for this NXP K64 example.

The other functions within the file which are not listed above do not contain any MCU-specific code (for example MCU_SPIReadWrite16 calls only MCU_SPIReadWrite8 and MCU_be16toh which are within the EVE_MCU_NXP.c file).

For each function which is to be edited, keep the function call itself and replace the code inside.
For example, in this case below, the original PIC code for the MCU_SPIReadWrite shown in ~~red~~ colour is deleted and is replaced with the code for the NXP K64 device.

```
// --------------------- SPI Send and Receive ---------------------------------

uint8_t MCU_SPIReadWrite8(uint8_t DataToWrite)
{

        uint8_t DataRead = 0;

        DataRead = SSP1BUF;         // Dummy read, ensure BF clearedDataRead = SSP1BUF;
        SSP1BUF = (DataToWrite);  // Write data to SPI data register
        while(!SSP1STATbits.BF);  // Wait for completion of the SPI transfer
        DataRead = SSP1BUF;         // Get the value clocked in from the FT8xx

        return DataRead;

        uint8_t DataRead[4];

        // ------- Clear flags and configure --------
        SPI1_MCR &= ~(SPI_MCR_HALT_MASK | SPI_MCR_FRZ_MASK);                    //
Un-Halt and Un-Freeze the SPI Master
        SPI1_MCR |= SPI_MCR_CLR_RXF_MASK;                                       //
Clear the RxF flag
        SPI1_SR = (SPI_SR_EOQF_MASK | SPI_SR_TFUF_MASK | SPI_SR_TFFF_MASK |
                SPI_SR_RFOF_MASK | SPI_SR_RFDF_MASK);    // Clear flags

        // -------- Clock out one byte and wait for one byte to come back --------
```

```
        SPI1_PUSHR = (SPI_PUSHR_EOQ_MASK | DataToWrite | SPI_PUSHR_CTAS(0) |
               SPI_PUSHR_PCS(1));  // Write the data byte to the PUSHR register

        // Wait until the byte is received back
        while ((SPI1_SR & SPI_SR_RFDF_MASK) == 0)
        {
        }
        DataRead[0] = SPI1_POPR;          // Read the received byte

        // Clear flags
        SPI1_SR = (SPI_SR_EOQF_MASK | SPI_SR_TFUF_MASK | SPI_SR_TFFF_MASK |
               SPI_SR_RFOF_MASK | SPI_SR_RFDF_MASK);
        // Halt and Freeze the SPI Master
        SPI1_MCR |= (SPI_MCR_HALT_MASK | SPI_MCR_FRZ_MASK);
        return DataRead[0];
}
```

As noted on the previous page, MCU_Setup is blank and will just return immediately on this MCU platform.

```
Void MCU_Setup(void)
{

}
```

Right-click on the project and choose Build

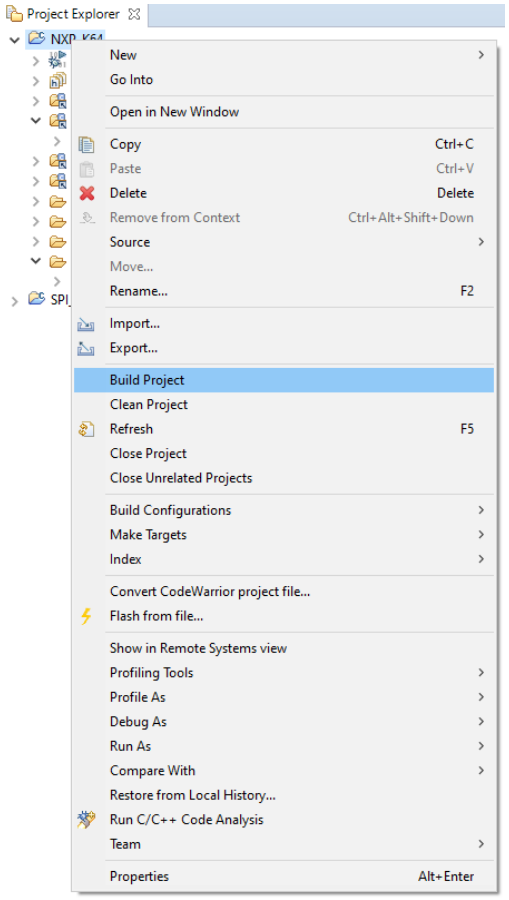


Figure 25 – Building the project

The code can then be programmed to the MCU, or a debug session can be started to program and run the code.

4.5 Additional Steps

Note: You can optionally delete the EVE_HAL_Linux.c file in the \EVE-MCU-BRT_AN_025\source directory as this is not used on MCU hosts. If using a Linux-based host this should be retained and EVE_HAL.c can be removed instead. Normally the #if defined condition should already ensure that the build does not include both but excluding from the project or removing will help in the unlikely case that there are any conflicts.

You can also remove other unused platforms to reduce the file size by deleting folders inside the \EVE-MCU-BRT_AN_025\ports and the \EVE-MCU-BRT_AN_025\examples\simple folders. Note that the common folder should be kept within the \EVE-MCU-BRT_AN_025\examples\simple folder, along with the folder for anu MCU which you want to be able to build for.

5 Hardware

The FRDM K64F board was used in this case.

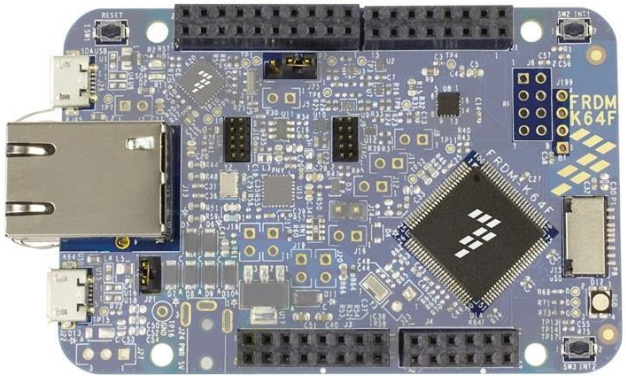


Figure 26 – FRDM K64F

The connections to the EVE module are shown in Figure 27. Many of the EVE modules from Bridgetek use the 16-way (8x2) dual row connector or the 10-way (10x1) single row connector. Examples include the following modules (correct at the time of writing but please check the Bridgetek website www.brtchip.com for the latest information on the modules)

8x2 Connector: ME817EV, ME813-WH50C, ME812A-WH50R

10x1 Connector: VM816C50A-D, VM810C50A-D, VM880C, VM801B, VM800B

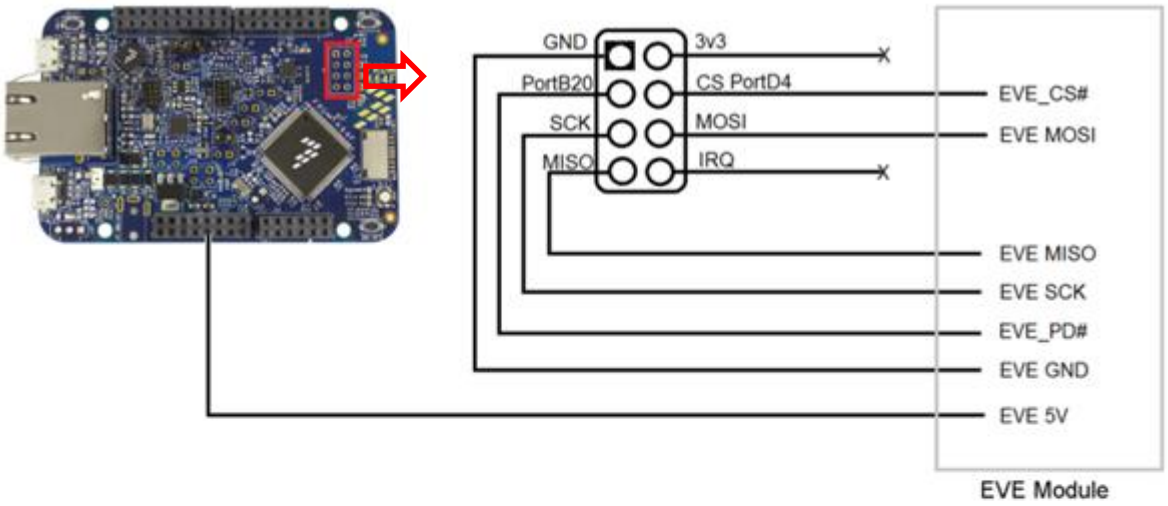
EVE modules from other manufacturers can also be used by connecting the SCK, MOSI, MISO, CS and PD signals. Refer to the module datasheet for details of the pinout, signal voltage levels, and power supply requirements.

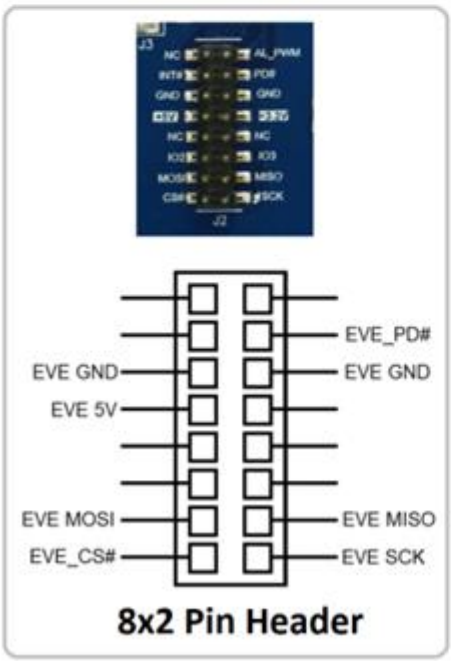
The FRDM K64F uses 3.3V I/O levels and so can be connected directly to EVE. If the MCU uses I/O voltages which do not match the EVE spec, they may require level translating buffers between the MCU and the EVE module.

Important Notes:

- The 5V supply from the FRDM K64F (as shown in the figure above) may be able to be used to power some EVE modules for testing. However, it is important to check the current capability spec of your MCU board to make sure that it can provide sufficient current. Exceeding the rated current may damage the MCU board and cause the voltage to drop leading to unreliable operation of the EVE module. Many EVE modules require more current than is available from development boards especially those with larger screens, brighter backlights, and audio amplifiers. Use an external power supply if necessary and connect the grounds of the EVE module, the MCU module, and the external PSU together.

- Check the schematic for the version of the NXP K64F board which will be used as some versions of this board may not have Port B20 on the pin shown in the diagram above. It may be replaced by another signal.





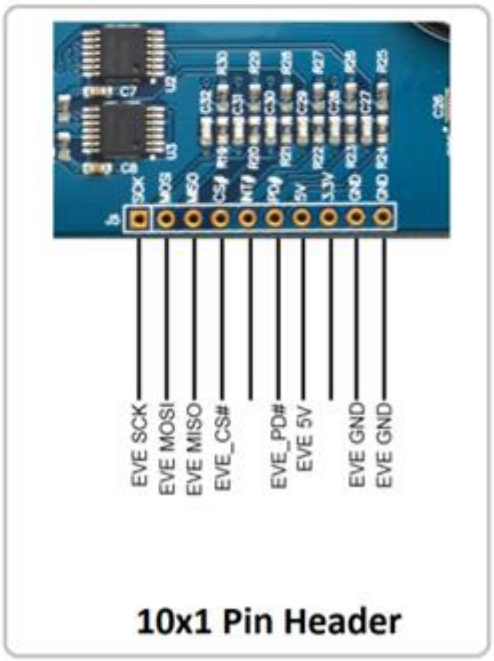


Figure 27 – Connections to EVE module

6 Testing the application

After building and programming the code, your display should follow the same operation shown in BRT_AN_025. On power-up, the screen will ask the user to tap the calibration dots and will then proceed to the screen below where you can touch over the counter
and it will increment.

Figure 28 – Main application screen

7 Conclusion

This application note is intended to help developers port the BRT_AN_025 code to their selected MCU and IDE.

Whilst BRT_AN_025 supports a range of MCUs already, many customers will have their own preferred MCU platform or have an existing product with an existing MCU in place where they would like to add support for the EVE family of devices.

This document uses the NXP K64F board, but a similar procedure may be used for other MCUs.
Because of the arrangement of the BRT_AN_025 code, adapting it for various MCU types is easily achievable by modifying the MCU-specific file (located in the EVE-MCU-BRT_AN_025/ports folder) to align with their MCU's registers and the suggested MCU/SPI/GPIO configuration"

After porting this BRT_AN_025 framework, the product developer can focus on the application layer where they can create content on the LCD screen using commands in the user-friendly syntax of the EVE Programmers Guide.

Note that the full source code project which is created during this application note is available as one of the platforms supported in the BRT_AN_025 code at the Bridgetek Github page.

8 Contact Information

Refer to https://brtchip.com/contact-us/ for contact information

Distributor and Sales Representatives

Please visit the Sales Network page of the Bridgetek Web site for the contact details of our distributor(s) and sales representative(s) in your country.

Appendix A– References

Document References

BRT_AN_025_source	Source code for BRT_AN_025 which includes the NXP_K64

Note that the code package above now includes the NXP_K64 platform code. This was ported using the steps in this application note BRT_AN_062. Refer to the following folders in this repository:

- EVE-MCU-BRT_AN_025/ports/eve_arch_nxpk64
- EVE-MCU-BRT_AN_025/examples/simple/NXP_K64

BRT_AN_062_Source	Source code for the SPI/GPIO test code in chapter 3
BRT_AN_025	EVE Portable MCU Example
NXP FRDM K64F	Development board and IDE used in this example
BRT_AN_006	Low level details of EVE SPI transfers
BRT_AN_008	Usage of the MCU library
BRT_AN_014	Examples using the MCU library from BRT_AN_008
EVE Programming Guides	Programming guides for the different EVE families
Bridgetek Website	See our website for details of the EVE devices and modules

Acronyms and Abbreviations

Terms	Description
EVE	Embedded Video Engine
MCU	Microcontroller
FT80x	First generation of EVE (FT800 and FT801)
FT81x	Second generation of EVE (FT810, FT811, FT812, FT813)
BT88x	New second-generation EVE devices (BT880, BT881, BT882, BT883)
BT81x	Third generation (BT815/6) and fourth generation of EVE (BT817/8)
LCD	Liquid Crystal Display
SPI	Serial Peripheral Interface
QSPI	Quad SPI
CS#	SPI Chip Select signal (Active low)
SCK	SPI Clock
MOSI	SPI Master Out Slave In data line carries data from MCU to EVE
MISO	SPI Master In Slave Out data line carries data from EVE to MCU
UART	Universal Asynchronous Receiver Transmitter for serial data transfer

Appendix B – List of Tables & Figures

List of Figures

Appendix C– Revision History

Document Title: BRT_AN_062 Porting BRT_AN_025 code to an NXP K64 MCU
Document Reference No.: BRT_000281
Clearance No.: BRT#213
Product Page: http://brtchip.com/i-ft8/
Document Feedback: Send Feedback

Revision	Changes	Date
1.0	Initial release	22-04-2024