FORENEX TEL: 886-4-22969886 FAX : 886-4-22969887
Forenex Technology Co., Ltd 5F-2,No212, Sec.4,Wenxin Rd.,North Dist, Taichung City 404, Taiwan(R.O.C.)

Document Number: AN15001
Application Note

The Software Manual
of
FBM4_K60512 or FES_M4

Version: **V0.2**

Benson	Benson	Wayne Lai
Approved By	**Checked By**	**Prepared By**

Description

This document is referable for FES_M4 series, FBM4_K60512 series.

Software Manual

Forenex provide customers two ways for application development.
One is developing with Freescale MQX support, the other one is developing without Freescale MQX support.
Below will introduce how to develop customer's own application software and programming into the start address 0x0C800 where is FBM4-K60512 intend to reserve for customer's APP.

1. MQX application note

This section will introduce you how to construct the application with Freescale MQX. MQX is developed and maintained by Freescale. Customer can download the Freescale MQX packages from Freescale web site. The source code and documents are included in the downloaded packages.

For your convenience, Forenex provides you the pre-build library which initializes all the hardware settings, so customers can focus on the application programming.
For the library, you can download "FBM4_Demo.zip" from Forenex. Extract it, you can find the library in the directory "lib". There are also other directories named "src" and "uv4", "src" is used to save the source code of the application and "uv4" is used to save the Keil uVision4 project settings.
So, how to begin with the MQX application programming?
Below is two easy way to start with MQX application programming.

A. Forenex provides you the demo program, you can open the project in "uv4" directory and build the project to get the hex file to download to the demo board. After that, customer can modify the application based on the code that Forenex provided.

B. If you want to modify the demo program that came from Freescale, please download the

"MQX" package from Freescale and the install it. After that, you can add the Freescale demo program with Forenex's library and download the program to the demo board.

Step 1: Extract the "FBM4_Demo.zip" that came from Forenex and delete all the files in "src" directory..

Step 2: Find the source code of the example you want to test in Freescale MQX installation directory.

Step 3: Open the project in "uv4" that generated in Step1.

Step 4: Delete all the files in "Source"

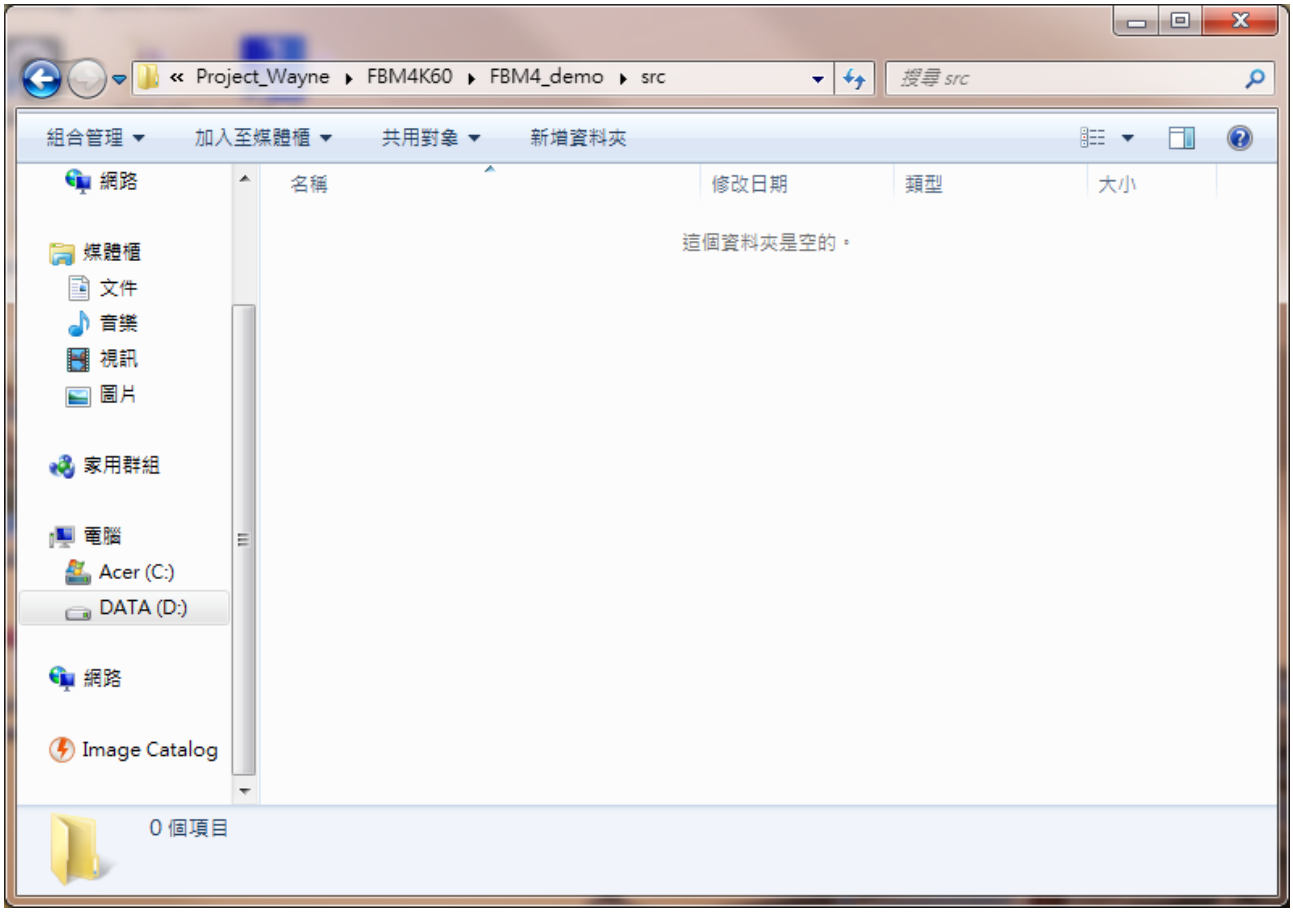
Step 5: Add the files to "Source" that generated in Step 2.

Step 6: Build the project and download the hex file to the demo board.

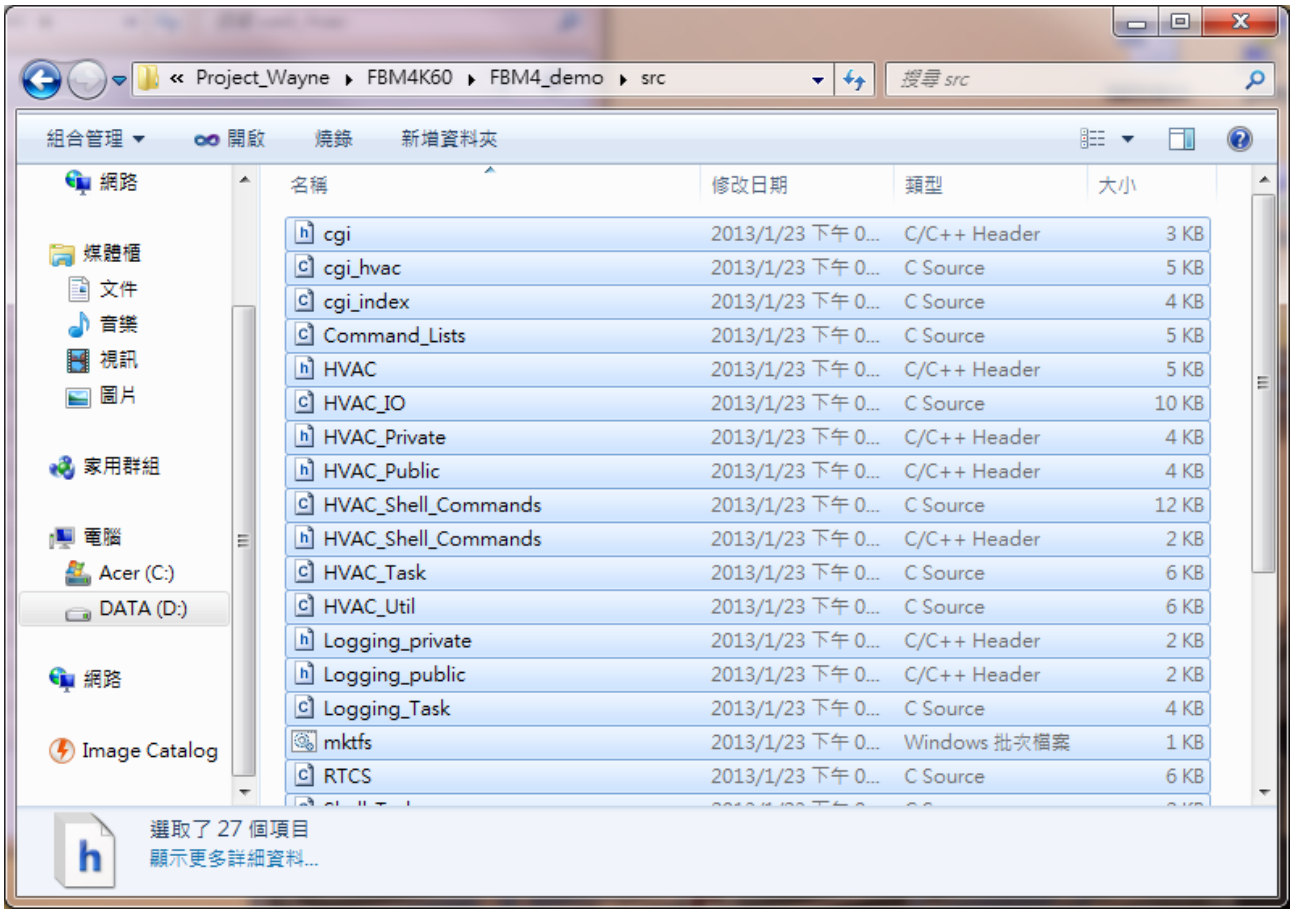
Step 7: Run and test the application.

Example :

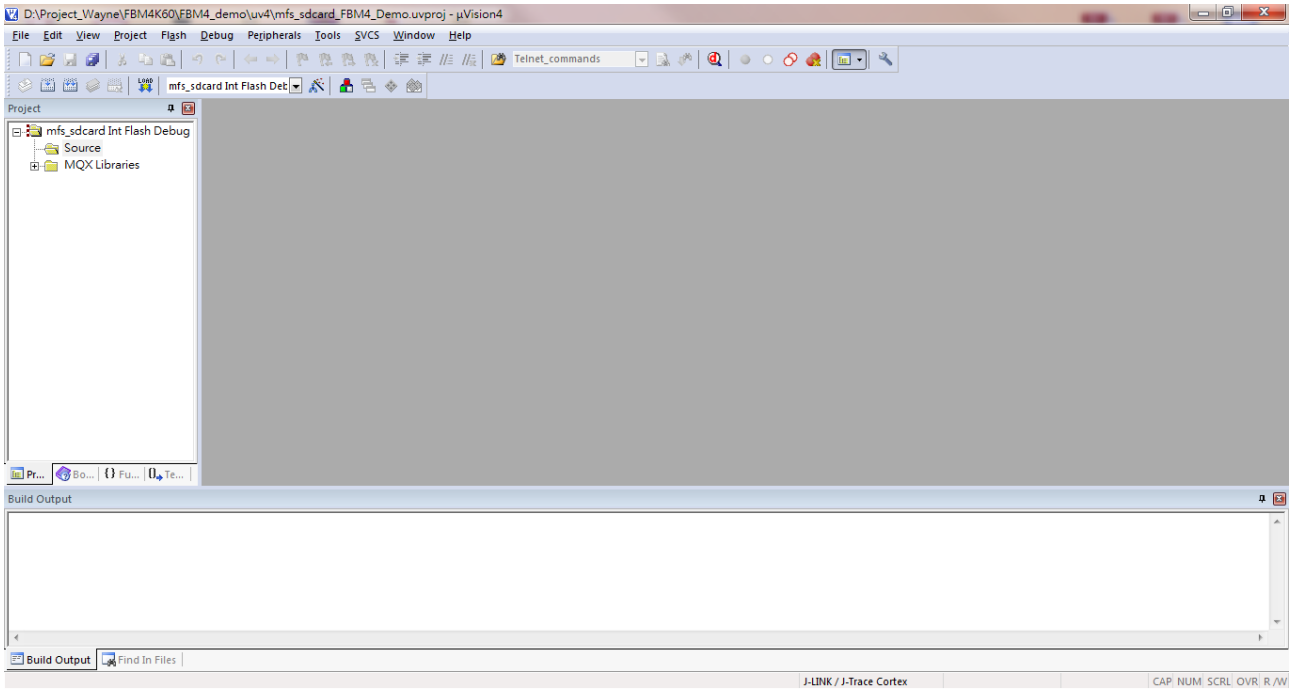
Step 1: Delete all the files in "src"



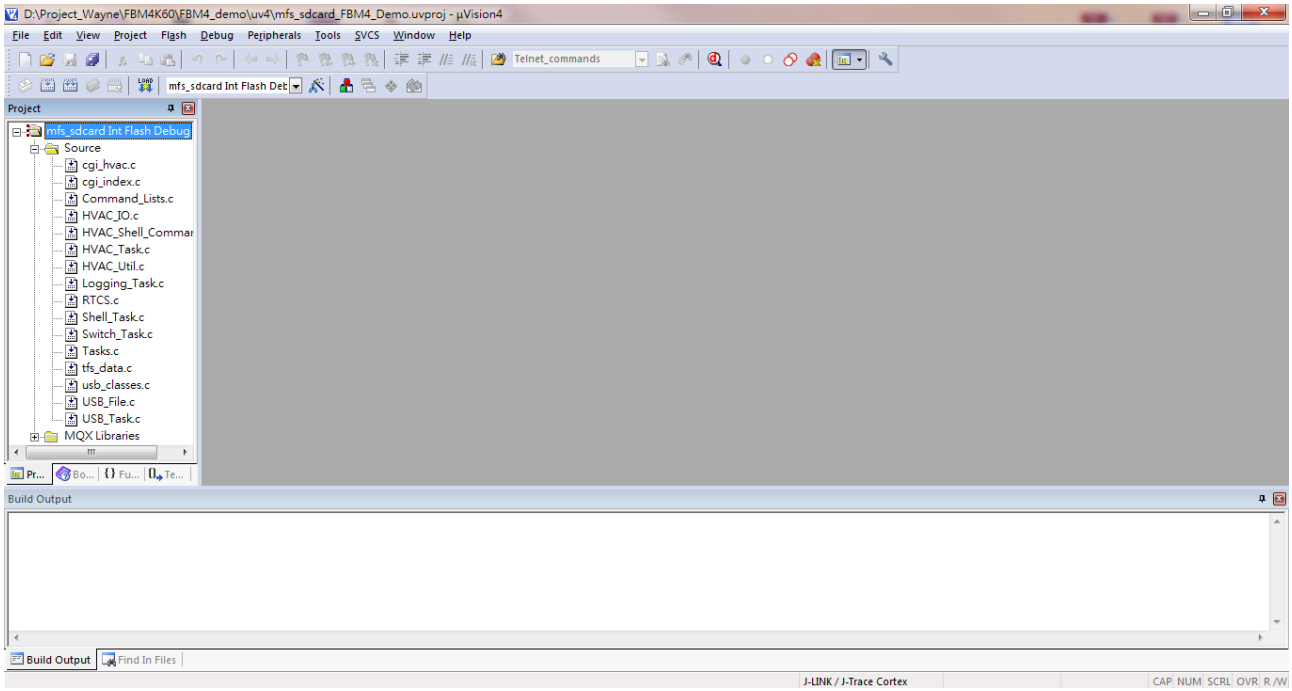
Step 2: Find desired MQX example, here we take “web_hvac”. You can find it in “C:\Freescale\Freescale_MQX_4_0\demo\web_hvac”, copy all the source code to “src” directory.



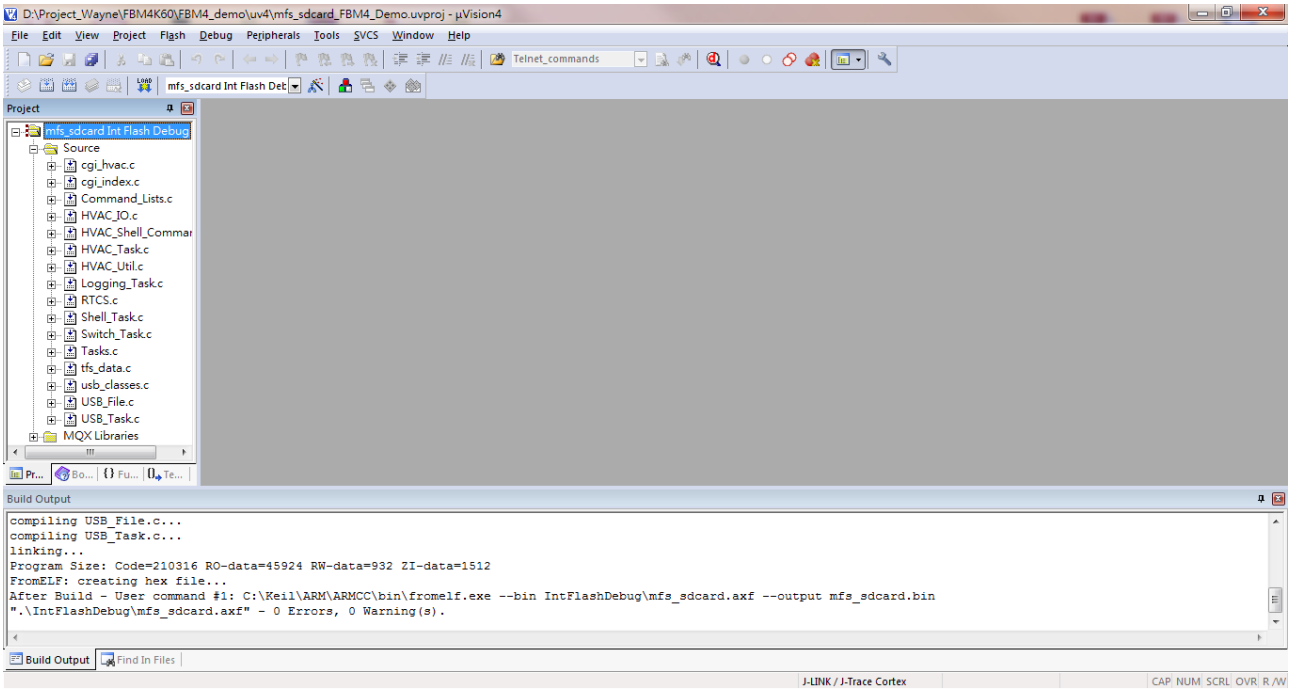
Step 3 &4: Open the project and delete all the files in “Source”



Step 5: Add the files to “Source” that generated in Step 2.



Step 6&7 : Build the project and download the hex file to the demo board. Then download to the demo board.



2. Stand-alone application note

All these function calls showed below were provided from Forenex Co., and developed under Forenex's Evaluation Board (FBM4_K60DN512). Below APIs are developed by Forenex and supposed to be used without MQX. You can get the source code "FBM4_K60512_TestAPP.rar" from Forenex.

ADC

K60 has two ADC modules : ADC0、ADC1. It uses linear successive approximation algorithm with up to 16-bit resolution. It has up to four pairs of differential and 24 single-ended external analog inputs. It also has programmable Gain Amplifier(PGA) with up to x64 gain.

ADC Function List

Function name	Description
ADC_Init	Initialize a ADC struct.
ADC_GerConversionValue	Read ADC conversion value.
ADC_ITConflg	Configure interrupt of ADC module.
ADC_GetITStatus	Get interrupt status of ADC module.
ADC_DMACmd	DMA command of ADC module.

ADC_Init

Function name	ADC_Init
Function description	Initialize ADC devices according to the parameters of ADC_InitTypeDef.
Input parameter	ADC_InitTypeDef struct
Output parameter	None
Return value	None
Requirement	None

ADCxMAP：Select pin and type, list below.

ADCxMAP	Description
ADC0_DP0_DM0	Differential mode，DP0、DM0 pin of ADC0 module
ADC0_SE0_DP0	Single-Ended mode，DP0 pin of ADC0 module, channel 0.
ADC0_SE1_DP1	Single-Ended mode，DP1 pin of ADC0 module, channel 1.
ADC1_SE0_DP0	Single-Ended mode，DP0 pin of ADC1 module, channel 0.

ADC_Precision：ADC module conversion precision, list below

ADC_Precision	Description
ADC_PRECISION_8BIT	Set ADC precision to 8Bit
ADC_PRECISION_10BIT	Set ADC precision to 10Bit
ADC_PRECISION_12BIT	Set ADC precision to 12Bit
ADC_PRECISION_16BIT	Set ADC precision to 16Bit

ADC_TriggerSelect：Select trigger condition, list below.

ADC_TriggerSelect	Description
ADC_TRIGGER_HW	Set as hardware trigger.
ADC_TRIGGER_SW	Set as software trigger.

Example;
Use module ADC0, single-ended mode for A/D conversion, precision is 16 bit, pin is DP0, using channel 0, software trigger. You can set below value to ADC_InitTypeDef struct :

```
ADC_InitTypeDefADC_InitStruct;
ADC_InitStruct.ADCxMPA          =   ADC0_SE0_DP0;
ADC_InitStruct.ADC_Precision    = ADC_PRECISION_16BIT;
ADC_InitStruct.ADC_TriggerSelect  = ADC_TRIGGER_SW;
ADC_Init(&ADC_InitStruct);
```

ADC_GetConversionValue

Function name	ADC_GetConversionValue
Function description	Read ADC conversion value
Input parameter	ADCxMAP：ADC module pin
Output parameter	None
Return value	A/D conversion result.
Requirement	Finish initializing ADC

Example, read the ADC conversion value and save to a variable “value”.

value = ADC_GetConversionValue;

ADC_ITConfig

Function name	ADC_ITConfig
Function description	Configure interrupt of ADC module.
Input parameter 1	ADCx: ADC0、ADC1
Input parameter 2	ADC_Mux: A/B channel select
Input parameter 3	ADC_IT: interrupt source select
Input parameter 4	NewState: ENABLE，DISABLE
Output parameter	None
Return value	None
Requirement	Finish initializing ADC

Example : Configure ADC0 module, channel A to generate interrupt when finishing A/D conversion.

ADC_ITConfig(ADC0,A,ADC_IT_AI,ENABLE);

ADC_GetITStatus

Function name	ADC_GetITStatus
Function description	Get interrupt status of ADC module.
Input parameter 1	ADCx: ADC0、ADC1
Input parameter 2	ADC_Mux: A/B channel select
Input parameter 3	ADC_IT: interrupt source select
Output parameter	None
Return value	Interrupt status flag
Requirement	Finish initializing ADC

Example: Get the interrupt status of ADC0 module, channel.

status = ADC_GetITStatus(ADC0,A,ADC_IT_AI);

ADC_DMACmd

Function name	ADC_DMACmd
Function description	DMA command of ADC module.
Input parameter 1	ADCx: ADC0、ADC1
Input parameter 2	ADC_DMAReq: Trigger DMA interrupt.
Input parameter 3	NewState: ENABLE，DISABLE
Output parameter	None
Return value	None
Requirement	Finish configuring DMA module

Example : Configure ADC0 module to trigger DMA function.

ADC_DMACmd(ADC0,ADC_DMAReq_COCO,ENABLE);

DAC

DAC is digital-to-analog converter. DAC of K60 has 16 word data buffer supported with configurable watermark and multiple operation modes. It can be setup as normal mode, swing mode and one-time scan mode.

DAC Function List

Function name	Description
DAC_Init	Initialize DAC module
DAC_StructInit	DAC module default settings
DAC_DMACmd	Set DMA function of DAC module
DAC_ITConfig	Configure interrupt of DAC module.
DAC_GetITStatus	Get interrupt status of DAC module.
DAC_SoftwareTrigger	Software trigger to DAC module
DAC_SetBuffer	Configure the data buffer
DAC_SetValue	Configure DAC module output voltage.

DAC_Init

Function name	DAC_Init
Function description	Initialize DAC module
Intput parameter	DAC_InitTypeDef
Output parameter	None
Return value	None
Requirement	None

DAC_TrigerMode：Select the trigger mode of DAC module.

DAC_TrigerMode	Description
DAC_TRIGER_MODE_NONE	No trigger
DAC_TRIGER_MODE_SOFTWARE	Software trigger
DAC_TRIGER_MODE_HARDWARE	Hardware trigger

DAC_BufferMode：Choose the buffer mode.

DAC_BufferMode	Description
BUFFER_MODE_DISABLE	Diable buffer mode
BUFFER_MODE_NORMAL	Normal mode
BUFFER_MODE_SWING	Swing mode
BUFFER_MODE_ONETIMESCAN	One-time scan mode

DAC_WaterMarkMode：Choose the watermark mode.

DAC_WaterMarkMode	Description
WATER_MODE_1WORD	Configure as 1 word
WATER_MODE_2WORD	Configure as 2 words
WATER_MODE_3WORD	Configure as 3 words
WATER_MODE_4WORD	Configure as 4 words

DAC_StructInit

Function name	DAC_StructInit
Function description	DAC module default settings.
Input parameter	DAC_InitStruct (DAC module settings)
Output parameter	DAC_InitStruct (DAC module default settings)
Return value	None
Requirement	None

Example : Use the default settings to initialize DAC module.

```
DAC_InitTypeDef   *DAC_InitStruct;
DAC_StructInit(DAC_InitStruct);
DAC_Init(DAC_InitStruct);
DAC_SetValue(DAC0,1000);
DAC_SoftwareTrigger(DAC0);
```

DAC_DMACmd

Function name	DAC_DMACmd
Function description	Set DMA function of DAC module
Input parameter1	DACx: DAC0
Input parameter2	DAC_DMAReq: DAC_DMAReq_DAC
Input parameter3	NewState: ENABLE，DISABLE
Output parameter	None
Return value	None
Requirement	Finish configuring DMA module

Example : Configure DAC0 module to trigger DMA function.DAC_DMACmd(DAC0,DAC_DMAReq_DAC,ENABLE);

DAC_ITConfig

Function name	DAC_ITConfig
Function description	Configure interrupt of DAC module.
Input parameter1	DACx: DAC0
Input parameter2	DAC_IT: DAC_IT_POINTER_BUTTOM DAC_IT_POINTER_TOP DAC_IT_WATER_MARK
Input parameter3	NewState: ENABLE，DISABLE
Output parameter	None
Return value	None
Requirement	Finish initializing DAC

Example : Configure the interrupt of DAC0 module.
DAC_ITConfig(DAC0,DAC_IT_POINTER_BUTTOM,ENABLE);

DAC_GetITStatus

Function name	DAC_GetITStatus
Function description	Get interrupt status of DAC module.
Input parameter1	DACx: ADC0
Input parameter2	DAC_IT: DAC_IT_POINTER_BUTTOM DAC_IT_POINTER_TOP DAC_IT_WATER_MARK
Output parameter	None
Return value	0: interrupt not occur 1: interrupt occur
Requirement	Finish initializing DAC

Example : Get the interrupt status of DAC0 module.

```
status = DAC_GetITStatus(DAC0,DAC_IT_POINTER_BUTTOM);
```

DAC_SoftwareTrigger

Function name	DAC_SoftwareTrigger
Function description	Software trigger to DAC module
Input parameter	DACx: ADC0
Output parameter	None
Return value	None
Requirement	Finish initializing DAC

Example : Trigger the DAC0 module for Digital to analog conversion.

```
DAC_SoftwareTrigger(DAC0);
```

DAC_SetBuffer

Function name	DAC_SetBuffer
Function description	Configure the data buffer
Input parameter1	DACx: ADC0
Input parameter2	DACBuffer
Input parameter3	NumberOfBuffer: index in the buffer(0~15)
Output parameter	None
Return value	None
Requirement	Finish initializing DAC

Example : Configure the buffer value is 100 and index is 10.
DAC_SetBuffer(DAC0,100,10);

DAC_SetValue

Function name	DAC_SetValue
Function description	Configure DAC module output voltage.
Input parameter1	DACx: ADC0
Input parameter2	DAC_Value: Output voltage (0~4095)
Output parameter	None
Return value	None
Requirement	Finish initializing DAC

Example : Configure the dac module output as 1000.
DAC_SetValue(DAC0,1000);

GPIO

The general-purpose input and output module communicates to the processor core via a zero wait state interface for maximum pin performance. The GPIO registers support 8-bit, 16-bit or 32-bit accesses.

GPIO Function List

Function name	Description
GPIO_Init	Initialize GPIO module
GPIO_WriteBit	Set GPIO bit output
GPIO_SetBits	Set GPIO bit output as high
GPIO_ResetBits	Set GPIO bit output as low
GPIO_Write	Set GPIO byte output
GPIO_ReadOutputDataBit	Read GPIO output bit status
GPIO_ReadOutputData	Read GPIO output byte status
GPIO_ReadInputDataBit	Read GPIO input bit status
GPIO_ReadInputData	Read GPIO input byte status
GPIO_GetITStates	Get interrupt status of GPIO module.
GPIO_ClearITPendingBit	Clear interrupt pending status

GPIO_Init

Function name	GPIO_Init
Function description	Initialize GPIO module
Input parameter	GPIO_InitTypeDef
Output parameter	None
Return value	None
Requirement	None

GPIO_Pin：Select GPIO pin

GPIO_Pin	Description
GPIO_Pin_0	Select pin 0
GPIO_Pin_1	Select pin 1
GPIO_Pin_n	Select pin n
GPIO_Pin_31	Select pin 31

GPIO_InitState：Select GPIO default value when set as output

GPIO_InitState	Description
Bit_SET	Output high
Bit_RESET	Output low

GPIO_Mode：Select GPIO mode

GPIO_Mode	Description
GPIO_Mode_IN_FLOATING	FLOATING input
GPIO_Mode_IPD	IPD input
GPIO_Mode_IPU	IPU input
GPIO_Mode_OOD	OOD output
GPIO_Mode_OPP	OPP output

GPIO_IRQMode：Select GPIO interrupt source

GPIO_IRQMode	Description
GPIO_IT_DISABLE	Disable interrupt
GPIO_IT_DMA_FALLING	DMA interrupt occurs when falling edge
GPIO_IT_DMA_RASING	DMA interrupt occurs when rising edge
GPIO_IT_DMA_RASING_FALLING	DMA interrupt occurs when rising or falling edge
GPIO_IT_LOW	Logic 0 Level Trigger
GPIO_IT_RISING	Rising edge trigger
GPIO_IT_FALLING	Falling edge trigger
GPIO_IT_RISING_FALLING	Falling or Rising edge trigger
GPIO_IT_HIGH	Logic 1 Level Trigger

GPIOx：Select GPIO

GPIOx	Description
PTA	Select GPIO，PORTA
PTB	Select GPIO，PORTB
PTC	Select GPIO，PORTC
PTD	Select GPIO，PORTD
PTE	Select GPIO，PORTE

Example : Configure PORTA, pin 31 as output with default output value to high. No interrupt function.

```
GPIO_InitTypeDef GPIO_Struct0;
GPIO_Struct0.GPIOx          =   PTA;
GPIO_Struct0.GPIO_InitState  = Bit_SET;
GPIO_Struct0.GPIO_Mode          = GPIO_Mode_OPP;
GPIO_Struct0.GPIO_Pin        = GPIO_Pin_31;
GPIO_Struct0.GPIO_IRQMode      = GPIO_IT_DISABLE;
GPIO_Init(&GPIO_Struct0);
```

GPIO_WriteBit

Function name	GPIO_WriteBit
Function description	Set GPIO bit output
Input parameter1	GPIOx：PTA、PTB、PTC、PTD、PTE
Input parameter2	GPIO_Pin：GPIO_Pin_0 ~ GPIO_Pin_31
Input parameter3	IO value：0 or 1
Output parameter	None
Return value	None
Requirement	GPIO must be configured as output mode.

Example : Set PORTA, pin 31, to output as low

```
GPIO_WriteBit(PTA, GPIO_Pin_31, Bit_SET);
```

GPIO_SetBit

Function name	GPIO_SetBit
Function description	Set GPIO bit output as high
Input parameter1	GPIOx：PTA、PTB、PTC、PTD、PTE
Input parameter2	GPIO_Pin：GPIO_Pin_0 ~ GPIO_Pin_31
Output parameter	None
Return value	None
Requirement	GPIO must be configured as output mode.

Example : Set PORTA, pin 31, to output as high

GPIO_SetBit(PTA, GPIO_Pin_31);

GPIO_ResetBit

Function name	GPIO_ResetBit
Function description	Set GPIO bit output as low
Input parameter1	GPIOx：PTA、PTB、PTC、PTD、PTE
Input parameter2	GPIO_Pin：GPIO_Pin_0 ~ GPIO_Pin_31
Output parameter	None
Return value	None
Requirement	GPIO must be configured as output mode.

Example : Set PORTA, pin 31, to output as low

GPIO_ResetBit (PTA, GPIO_Pin_31);

GPIO_Write

Function name	GPIO_Write
Function description	Set GPIO byte output
Input parameter1	GPIOx：PTA、PTB、PTC、PTD、PTE
Input parameter2	PortVal：32Bit data
Output parameter	None
Return value	None
Requirement	GPIO must be configured as output mode.

Example : Set PORTA to all output as low.

GPIO_Write (PTA, 0x00000000);

GPIO_ReadOutputDataBit

Function name	GPIO_ReadOutputDataBit
Function description	Read GPIO output bit status
Input parameter1	GPIOx：PTA、PTB、PTC、PTD、PTE
Input parameter2	GPIO_Pin：GPIO_Pin_0 ~ GPIO_Pin_31
Output parameter	None
Return value	Bit status of selected pin
Requirement	GPIO must be configured as output mode.

Example : Read PORTA, pin 31, status and save it to “data”
data= GPIO_ReadOutputDataBit(PTA, GPIO_Pin_31);

GPIO_ReadOutputData

Function name	GPIO_ReadOutputData
Function description	Read GPIO output byte status
Input parameter1	GPIOx：PTA、PTB、PTC、PTD、PTE
Output parameter	None
Return value	Byte status of selected GPIO
Requirement	GPIO must be configured as output mode.

Example : Read the status of all PORTA 32 bits and save to “data”
data= GPIO_ReadOutputDataBit(PTA);

GPIO_ReadInputDataBit

Function name	GPIO_ReadInputDataBit
Function description	Read GPIO input bit status
Input parameter1	GPIOx：PTA、PTB、PTC、PTD、PTE
Input parameter2	GPIO_Pin：GPIO_Pin_0 ~ GPIO_Pin_31
Output parameter	None
Return value	Bit status of selected pin
Requirement	GPIO must be configured as input mode.

Example : Read PORTA, pin 31, status and save it to “data”
data= GPIO_ReadInputDataBit (PTA, GPIO_Pin_31);

GPIO_ReadInputData

Function name	GPIO_ReadInputData
Function description	Read GPIO input byte status
Input parameter1	GPIOx：PTA、PTB、PTC、PTD、PTE
Output parameter	None
Return value	Byte status of selected GPIO
Requirement	GPIO must be configured as input mode.

Example : Read the status of all PORTA 32 bits and save to “data”
data= GPIO_ ReadInputData (PTA);

GPIO_GetITStates

Function name	GPIO_GetITStates
Function description	Get interrupt status of GPIO module.
Input parameter1	GPIOx：PTA、PTB、PTC、PTD、PTE
Input parameter2	GPIO_Pin：GPIO_Pin_0 ~ GPIO_Pin_31
Output parameter	None
Return value	Interrupt status
Requirement	Finish initializing GPIO

Example : Get PORTA, pin 31, interrupt status and save to “data”
data= GPIO_ GetITStates (PTA, GPIO_Pin_31);

GPIO_ClearITPendingBit

Function name	GPIO_ClearITPendingBit
Function description	Clear interrupt pending status
Input parameter1	GPIOx：PTA、PTB、PTC、PTD、PTE
Input parameter2	GPIO_Pin：GPIO_Pin_0 ~ GPIO_Pin_31
Output parameter	None
Return value	None

Requirement	Finish initializing GPIO

Example : Clear PORTA, pin 31, interrupt status.
GPIO_ ClearITPendingBit (PTA, GPIO_Pin_31);

UART

The UART allows asynchronous serial communication with peripheral devices and CPUs.

UART Function List

Function name	Description
UART_Init	Initialize UART module
UART_SendData	UART send 1Byte data
UART_ReceiveData	UART read 1Byte data
UART_SendDataInt	Use UART interrupt to send data
UART_SendDataIntProcess	Use UART interrupt to process function
UART_DebugPortInit	UART default debug port
UART_ITConfig	Configure interrupt of UART module.
UART_GetITStatus	Get interrupt status of UART module.

UART_Init

Function name	UART_Init
Function description	Initialize UART module
Input parameter	UART_InitTypeDef
Output parameter	None
Return value	None
Requirement	None

UARTxMAP：Select UART pin

UARTxMAP	Description
UART3_RX_PC16_TX_PC17	UART3 RX is PC16，TX is PC17
UART4_RX_PC14_TX_PC15	UART4 RX is PC14，TX is PC15
UART5_RX_PD8_TX_PD9	UART5 RX is PD8，TX is PD9

UART_BaudRate：Setup UART baudrate.

UART_BaudRate	Description
Decimal	Baudrate

Example : Configure UART3, PC16 as RX and PC17 as TX, baudrate is 115200.

```
UART_InitTypeDefUART_Struct0;
UART_Struct0.UARTxMAP          =  UART3_RX_PC16_TX_PC17;
UART_Struct0.UART_BaudRate     = 115200;
UART_Init(&UART_Struct0);
```

UART_SendData

Function name	UART_SendData
Function description	UART send 1Byte data
Input parameter1	UARTx：UART3、UART4、UART5
Input parameter2	Data：1Byte data
Output parameter	None
Return value	None
Requirement	Finish initializing UART

Example : Use UART3 to send Data.

```
UART_SendData(UART3,Data);
```

UART_ReceiveData

Function name	UART_ReceiveData
Function description	UART read 1Byte data
Input parameter1	UARTx：UART3、UART4、UART5
Input parameter2	Data：1Byte data
Output parameter	None
Return value	0: Fail 1: Success
Requirement	Finish initializing UART

Example : Use UART3 to receive data and save to variable “Data”.

```
UART_ReceiveData(UART3,&Data);
```

UART_SendDataInt

Function name	UART_SendDataInt
Function description	Use UART interrupt to send data
Input parameter1	UARTx：UART3、UART4、UART5
Input parameter2	DataBuf：Address of send data
	Len：data length
Output parameter	None
Return value	None
Requirement	Finish initializing UART

Example : Use UART3 to send data with length 10 bytes, and is stored in variable “Data[]”
UART_SendDataInt (UART3, Data,10);

UART_SendDataIntProcess

Function name	UART_SendDataIntProcess
Function description	Use this for your own UART interrupt process function.
Input parameter1	UARTx：UART3、UART4、UART5
Output parameter	None
Return value	None
Requirement	Finish initializing UART

Example : Put the function to the interrupt process function.

UART_DebugPortInit

Function name	UART_DebugPortInit
Function description	UART default settings
Input parameter1	UARTxMAP： pin settings
Input parameter2	UART_BaudRate：baudrate
Output parameter	None
Return value	None
Requirement	Finish initializing UART

Example : Please reference to “UART_Init”.

UART_ITConfig

Function name	UART_ITConfig
Function description	Configure interrupt of UART module.
Input parameter1	UARTx：UART3、UART4、UART5 UART_IT：interrupt type State：interrupt status(Enable、Disable)
Output parameter	None
Return value	None
Requirement	Finish initializing UART

Example : Use UART3 to receive interrupt.
UART_ITConfig(UART3,UART_IT_RDRF,ENABLE);

UART_GetITStatus

Function name	UART_GetITStatus
Function description	Get interrupt status of UART module.
Input parameter1 Input parameter2	UARTx：UART3、UART4、UART5 UART_IT：interrupt type
Output parameter	None
Return value	interrupt status
Requirement	Finish initializing UART

Example ; Get the interrupt status of UART3 and save to variable “state”
state= UART_GetITStatus (UART3, UART_IT_RDRF);

I2C

The inter-integrated circuit (I2C, I2C, or IIC) module provides a method of communication between a number of devices. The interface is designed to operate up to 100 kbit/s with maximum bus loading and timing. The I2C device is capable of operating at higher baud rates, up to a maximum of clock/20, with reduced bus loading. The maximum communication length and the number of devices that can be connected are limited by a maximum bus capacitance of 400 pF. The I2C module also complies with the *System Management Bus (SMBus) Specification, version 2*.

I2C Function List

Function name	Description
I2C_Init	Initialize I2C module
I2C_GenerateSTART	Use I2C module to generate “START”
I2C_GenerateRESTART	Use I2C module to generate “RESTART”
I2C_GenerateSTOP	Use I2C module to generate “STOP”
I2C_SendDate	Use I2C module to send 1Byte data
I2C_Send7bitAddress	Use I2C module to send 7bit slave address
I2C_WaitAck	Use I2C module to wait “ACK”
I2C_SetMasterMode	Configure I2C module as “Master” mode
I2C_GenerateAck	Use I2C module to generate “ACK”
I2C_EnableAck	Use I2C module to response, send “ACK” when read one byte.
I2C_ITConfig	Configure interrupt of I2C module.
I2C_GetITStatus	Get interrupt status of i2C module.
I2C_DMACmd	Enable DMA function of I2C module
I2C_ClearITPendBit	Clear I2C module interrupt pending status

I2C_Init

Function name	I2C_Init
Function description	Initialize I2C module
Input parameter	I2C_InitStruct:
Output parameter	None
Return value	None
Requirement	None

I2CxMAP：Configure I2C pins.

I2CxMAP	Description
I2C0_SCL_PB0_SDA_PB1	I2C0 module，SCL is PORT B 0； SDA is PORT B 1
I2C0_SCL_PB2_SDA_PB3	I2C0 module，SCL is PORT B 2； SDA is PORT B 3
I2C1_SCL_PE1_SDA_PE0	I2C0 module，SCL is PORT E 1； SDA is PORT E 0
I2C1_SCL_PC10_SDA_PC11	I2C0 module，SCL is PORT C 10； SDA is PORT C 11

I2C_ClockSpeed：This is use to configure the communication speed of I2C module

I2C_ClockSpeed	Description
I2C_CLOCK_SPEED_50KHZ	Configure communication speed as 50KHZ
I2C_CLOCK_SPEED_100KHZ	Configure communication speed as 100KHZ
I2C_CLOCK_SPEED_150KHZ	Configure communication speed as 150KHZ
I2C_CLOCK_SPEED_200KHZ	Configure communication speed as 200KHZ
I2C_CLOCK_SPEED_250KHZ	Configure communication speed as 250KHZ
I2C_CLOCK_SPEED_300KHZ	Configure communication speed as 300KHZ

Example: Configure I2C1 module as “Master” mode, communication speed is 200KHz, use PORTC10 and PORTC11 as SCL and SDA signal.

```
I2C_InitTypeDef   I2C_InitStruct1;
I2C_InitStruct1.I2CxMAP = I2C1_SCL_PC10_SDA_PC11;
I2C_InitStruct1.I2C_ClockSpeed = I2C_CLOCK_SPEED_200KHZ;
```

I2C_Init(&I2C_InitStruct1);

I2C_GenerateSTART

Function name	I2C_GenerateSTART
Function description	Use I2C module to generate “START”
Input parameter	I2Cx：I2C0、I2C1
Output parameter	None
Return value	None
Requirement	Finish initializing I2C

Example:
Generate START on I2C0.
I2C_GenerateSTART(I2C0);

I2C_GenerateRESTART

Function name	I2C_GenerateRESTART
Function description	Use I2C module to generate “RESTART”
Input parameter	I2Cx：I2C0、I2C1
Output parameter	None
Return value	None
Requirement	Finish initializing I2C

Example:
Generate RESTART on I2C0.
I2C_GenerateRESTART(I2C0);

I2C_GenerateSTOP

Function name	I2C_GenerateSTOP
Function description	Use I2C module to generate “STOP”
Input parameter	I2Cx：I2C0、I2C1
Output parameter	None
Return value	None
Requirement	Finish initializing I2C

Example:
Generate STOP on I2C0.

I2C_GenerateSTOP(I2C0);

I2C_SendData

Function name	I2C_SendData
Function description	Use I2C module to send 1Byte data
Input parameter1 **Input parameter2**	I2Cx：I2C0、I2C1 data8：1Byte data to be send
Output parameter	None
Return value	None
Requirement	Finish initializing I2C

Example:
Send 1Byte data on I2C0.

```
uint8 data = 0xaa;
I2C_SendData (I2C0, data);
```

I2C_Send7bitAddress

Function name	I2C_Send7bitAddress
Function description	Use I2C module to send 7bit slave address
Input parameter1 **Input parameter2** **Input parameter3**	I2Cx：I2C0、I2C1 Address：The slave address of the IC to communicate with I2C_Diretion：I2C_MASTER_WRITE、I2C_MASTER_READ
Output parameter	None
Return value	None
Requirement	Finish initializing I2C

Example:
Send 7bit slave address on I2C0 with address 0xA0 for write.

```
uint8 address = 0xA0;
I2C_Send7bitAddress(I2C0, address, I2C_MASTER_WRITE);
```

I2C_WaitAck

Function name	I2C_WaitAck
Function description	Use I2C module to wait “ACK”
Input parameter	I2Cx：I2C0、I2C1
Output parameter	None
Return value	TRUE(1)，Success to receive “ACK” FALSE(0)，Fail to receive “ACK”
Requirement	Finish initializing I2C

Example:
Wait ACK on I2C0.

```
I2C_WaitAck(I2C0);
```

I2C_ SetMasterMode

Function name	I2C_SetMasterMode
Function description	Configure I2C module as “Master” mode
Input parameter1	I2Cx：I2C0、I2C1
Input parameter2	I2C_Direction：I2C_MASTER_WRITE； I2C_MASTER_READ
Output parameter	None
Return value	None
Requirement	Finish initializing I2C

Example:
Configure I2C0 as master mode for reading.

```
I2C_SetMasterMode(I2C0, I2C_MASTER_READ);
```

I2C_ GenerateAck

Function name	I2C_GenerateAck
Function description	Use I2C module to generate “ACK”
Input parameter	I2Cx：I2C0、I2C1
Output parameter	None
Return value	None
Requirement	Finish initializing I2C

Example:
Generate ACK on I2C0.

```
I2C_GenerateAck(I2C0);
```

I2C_ EnableAck

Function name	I2C_EnableAck
Function description	Use I2C module to response, send “ACK” when read one byte.
Input parameter	I2Cx：I2C0、I2C1
Output parameter	None
Return value	None
Requirement	Finish initializing I2C

Example:
Enable ACK on I2C0.

```
I2C_EnableAck (I2C0);
```

I2C_ ITConfig

Function name	I2C_ITConfig
Function description	Configure interrupt of I2C module.
Input parameter1	I2Cx：I2C0、I2C1
Input parameter2	I2C_IT：I2C_IT_TCF
Input parameter3	NewState：ENABLE、DISABLE
Output parameter	None
Return value	None
Requirement	Finish initializing I2C

Example:

Enable I2C interrupt on I2C0.

```
I2C_ITConfig(I2C0, I2C_IT_TCF, ENABLE);
```

I2C_ GetITStatus

Function name	I2C_GetITStatus
Function description	Get interrupt status of I2C module.
Input parameter1	I2Cx：I2C0、I2C1
Input parameter2	I2C_IT：I2C_IT_TCF
Output parameter	None
Return value	0：Not occur；1：interrupt occurs
Requirement	Finish initializing I2C

Example:

Get interrupt status of I2C0.

```
status = I2C_GetITStatus(I2C0, I2C_IT_TCF);
```

I2C_ DMACmd

Function name	I2C_DMACmd
Function description	Enable DMA function of I2C module
Input parameter1	I2Cx：I2C0、I2C1
Input parameter2	I2C_DMAReq：I2C_DMAReq _TCF trigger DMA interrupt
Input parameter3	NewState：ENABLE、DISABLE
Output parameter	None
Return value	None
Requirement	Finish configuring DMA module

Example:
Enable DMA on I2C0

I2C_DMACmd(I2C0, I2C_DMAReq_TCF, ENABLE);

I2C_ ClearITPendingBit

Function name	I2C_ClearITPendingBit
Function description	Clear I2C module interrupt pending status
Input parameter1	I2Cx：I2C0、I2C1
Input parameter2	I2C_IT：clear selected interrupt status，I2C_IT_TCF
Output parameter	None
Return value	None
Requirement	Finish initializing I2C

Example:
Clear I2C0 pending interrupt status

I2C_ClearITPendingBit(I2C0, I2C_IT_TCF);

SPI

The serial peripheral interface (SPI) module provides a synchronous serial bus for communication between an MCU and an external peripheral device.

SPI Function List

Function name	Description
SPI_Init	Initialize SPI module
SPI_ReadWriteByte	Read/Write SPI module for one byte
SPI_ITConfig	Configure interrupt of SPI module.
SPI_GetITStatus	Get interrupt status of SPI module.
SPI_ClearITPendingBut	Clear interrupt status
SPI_DMACmd	Configure SPI in DMA mode

SPI_Init

Function name	I2C_Init
Function description	Initialize SPI module
Input parameter	SPI_InitStruct
Output parameter	None
Return value	None
Requirement	None

SPIxDataMap：Configure the SCK, MISo and MOSI pin

SPIxDataMap	Description
SPI0_SCK_PA15_ SOUT_PA16_SIN_PA17	SPI0 module，SCK signal is PORTA 15； SOUT signal is PORTA 16； SIN signal is PORTA 17
SPI0_SCK_PC5_ SOUT_PC6_SIN_PC7	SPI0 module，SCK signal is PORTC 5； SOUT signal is PORTC 6； SIN signal is PORTC 7
SPI0_SCK_PD1_ SOUT_PD2_SIN_PD3	SPI0 module，SCK signal is PORTD 1； SOUT signal is PORTD 2； SIN signal is PORTD 3

SPI1_SCK_PE2_ **SOUT_PE1_SIN_PE3**	SPI0 module，SCK signal is PORTE 2； SOUT signal is PORTE 1； SIN signal is PORTE 3
SPI1_SCK_PB11_ **SOUT_PB16_SIN_PB17**	SPI0 module，SCK signal is PORTB 11； SOUT signal is PORTB 16； SIN signal is PORTB 17
SPI2_SCK_PB21_ **SOUT_PB22_SIN_PB23**	SPI0 module，SCK signal is PORTB 21； SOUT signal is PORTB 22； SIN signal is PORTB 23

SPIxPCSMap：Configure the CS pin of SPI module

SPIxPCSMap	Description
SPI0_PCS0_PA14	SPI0 module CS channel 0，PORTA 14
SPI0_PCS1_PC3	SPI0 module CS channel 1，PORTC 3
SPI0_PCS2_PC2	SPI0 module CS channel 2，PORTC 2
SPI0_PCS3_PC1	SPI0 module CS channel 3，PORTC 1
SPI0_PCS4_PC0	SPI0 module CS channel 4，PORTC 0
SPI1_PCS0_PB10	SPI1 module CS channel 0，PORTB 10
SPI1_PCS1_PB9	SPI1 module CS channel 1，PORTB 9
SPI1_PCS2_PE5	SPI1 module CS channel 2，PORTE 5
SPI1_PCS3_PE6	SPI1 module CS channel 3，PORTE 6
SPI1_PCS4_PB20	SPI1 module CS channel 4，PORTB 20

SPI_DataSize：Configure the number of bits transferred per frame.

SPI_DataSize	Description
8	8 bit

SPI_CPOL：Configure the clock polarity

SPI_CPOL	Description
SPI_CPOL_Low	The inactive state value of SCK is low
SPI_CPOL_High	The inactive state value of SCK is high

SPI_Mode：Configure the Master/Slave mode

SPI_Mode	Description
SPI_Mode_Master	SPI module as Master mode
SPI_Mode_Slave	SPI module as Slave mode

SPI_CPHA：Configure the clock phase

SPI_CPHA	Description
SPI_CPHA_1Edge	Data is captured on the leading edge of SCK and changed on the following edge
SPI_CPHA_2Edge	Data is changed on the leading edge of SCK and captured on the following edge

SPI_BaudRatePrescaler：Configure the baud rate scaler

SPI_BaudRatePrescaler	Description
SPI_BaudRatePrescaler_2	Baud rate scaler value 2
SPI_BaudRatePrescaler_4	Baud rate scaler value 4
SPI_BaudRatePrescaler_6	Baud rate scaler value 6
SPI_BaudRatePrescaler_8	Baud rate scaler value 8
SPI_BaudRatePrescaler_16	Baud rate scaler value 16
SPI_BaudRatePrescaler_32	Baud rate scaler value 32
SPI_BaudRatePrescaler_64	Baud rate scaler value 64
SPI_BaudRatePrescaler_128	Baud rate scaler value 128
SPI_BaudRatePrescaler_256	Baud rate scaler value 256
SPI_BaudRatePrescaler_2048	Baud rate scaler value 2048

SPI_FirstBit：Configure MSB or LSB first

SPI_FirstBit	Description
SPI_FirstBit_MSB	Data is transferred MSB first
SPI_FirstBit_LSB	Data is transferred LSB first

Example : Use SPI0 module PORTA, pin 14,15,16,17, data size 8 bit, MSB first, SPI mode 0, SPI master,

```
SPT_InitTyteDef  SPI_InitStruct1;
SPI_InitStruct1.SPIxDataMap = SPI0_SCK_PA15_SOUT_PA16_SIN_PA17;
SPI_InitStruct1.SPIxPCSMap = SPI0_PCS0_PA14;
SPI_InitStruct1.SPI_DataSize = 8;
SPI_InitStruct1.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_2;
SPI_InitStruct1.SPI_Mode = SPI_Mode_Master;
SPI_InitStruct1.SPI_CPHA = SPI_CPHA_1Edge;
SPI_InitStruct1.CPOL = SPI_CPOL_Low;
SPI_InitStruct1.SPI_FirstBit_MSB;
```

SPI_ReadWriteByte

Function name	SPI_ReadWriteByte
Function description	Read/Write SPI module for one byte
Input parameter1 **Input parameter2** **Input parameter3**	SPICSMap： CS pin of SPI module Data：one byte data PCS_State ：CS level when finish
Output parameter	None
Return value	Read one byte data
Requirement	Finishing initializing SPI module

Example:

1. Write one byte data and set CS to low when finish

```
SPI_ReadWriteByte(SPI0_SCK_PA15_SOUT_PA16_SIN_PA17,Data,SPI_PCS_Inactive);
```

2.Read one byte data and set CS to low when finish.

```
Read = SPI_ReadWriteByte (SPI0_SCK_PA15_ SOUT_PA16_SIN_PA17 ,0
,SPI_PCS_Inactive);
```

SPI_ITConfig

Function name	SPI_ITConfig
Function description	Configure interrupt of SPI module.
Input parameter	SPIx：SPI0、SPI1、SPI2 SPI_IT：SPI module interrupt type NewState：Configure interrupt type， ENABLE、DIABLE
Output parameter	None
Return value	None
Requirement	Finish initializing SPI

Example : Configure SPI0 module to generate an interrupt when finishing sending data

```
SPI_ITConfig(SPI0,SPI_IT_TCF,ENABLE);
```

SPI_GetITStatuts

Function name	SPI_GetITStatus
Function description	Get interrupt status of SPI module.
Input parameter1	SPIx：SPI0、SPI1、SPI2
Input parameter2	SPI_IT：SPI module interrupt type
Output parameter	None
Return value	Interrupt status
Requirement	Finish initializing SPI

Example : Get the interrupt status of SPI module

```
Status = SPI_GetITStatus(SPI0,SPI_IT_TCF);
```

SPI_ClearITPendingBit

Function name	SPI_ClearITPendingBit
Function description	Clear interrupt status
Input parameter1	SPIx：SPI0、SPI1、SPI2
Input parameter2	SPI_IT：SPI module interrupt type
Output parameter	None
Return value	None
Requirement	Finish initializing SPI

Example : Clear the interrupt status of SPI module

```
SPI_ClearITPendingBit(SPI0,SPI_IT_TCF);
```

SPI_DMACmd

Function name	SPI_DMACmd
Function description	Configure SPI in DMA mode
Input parameter1	SPIx：SPI0、SPI1、SPI2
Input parameter2	SPI_DMAReq：SPI_DMAReq_TCF
Input parameter3	NewState：ENABLE、DISABLE
Output parameter	None
Return value	None
Requirement	Finish configuring DMA module

Example : Configure SPI0 module to use DMA function

```
SPI_DMACmd(SPI0, SPI_DMAReq_TCF,ENABLE);
```

ENET

The MAC-NET core, in conjunction with a 10/100 MAC, implements layer 3 network acceleration functions. These functions are designed to accelerate the processing of various common networking protocols, such as IP, TCP, UDP, and ICMP, providing wire speed services to client applications.

If customer want to use ENET with Ethernet protocol stack, please use Freescale MQX. Below APIs are used to access the ENET hardware registers, no protocol stack included.

ENET Function List

Function name	Description
ENET_Init	Initialize the ENET module
ENET_MacSendData	Ethernet MAC send data
ENET_MacRecData	Ethernet MAC receive data
ENET_MiiLinkState	Check if Ethernet is link ok

ENET_Init

Function name	ENET_Init
Function description	Initialize the ENET module
Input parameter	ENET_InitStrut
Output parameter	None
Return value	None
Requirement	None

Example :

```
const uint8_t LocalMacAddress[6] = {0x02,0x02,0x02,0x02,0x02,0x02};
ENET_InitTypeDef    ENET_InitStruct1;
ENET_InitStruct1.pMacAddress = (uint8_t*)LocalMacAddress;
ENET_Init(&ENET_InitStruct1)
```

ENET_MacSendData

Function name	ENET_MacSenData
Function description	Ethernet MAC send data
Input parameter1 Input parameter2 Input parameter3	ch : pointer to data len：data length None
Output parameter	None
Return value	None
Requirement	Finish initializing Ethernet

Example:

ENET_MacSendData(pData,100);

ENET_MacRecData

Function name	ENET_MacRecData
Function description	Ethernet MAC receive data
Input parameter1	ch : pointer of receive data
Output parameter	Received data length
Return value	None
Requirement	Finish initializing Ethernet

Example:

len = ENET_MacRecData(pData);

ENET_MiiLinkState

Function name	ENET_MiiLinkState
Function description	Check if Ethernet is link ok
Input parameter	None
Output parameter	None
Return value	TRUE Or FALSE
Requirement	Finish initializing Ethernet

Example:

```
LinkStates = ENET_MiiLinkState();
```

CAN

The CAN module is a communication controller implementing the CAN protocol according to the CAN 2.0B protocol specification. A general block diagram is shown in the following figure, which describes the main sub-blocks implemented in the CAN module, including one associated memory for storing Message Buffers, Rx Global Mask Registers, Rx Individual Mask Registers, Rx FIFO and Rx FIFO ID Filters. The functions of the sub-modules are described in subsequent sections.

CAN Function List

Function name	Description
CAN_Init	Initialize the CAN module
CAN_EnableReceiveMB	Enable the receive Message Buffer
CAN_Receive	CAN receive message
CAN_Transmit	CAN send message
CAN_ITConfig	Configure interrupt of CAN module.
CAN_GetITStatus	Get interrupt status of CAN module.
CAN_ClearITPendingBit	Clear interrupt bit of CAN
CAN_ClearAllITPendingBit	Clear all interrupt bit of CAN

CAN_Init

Function name	CAN_Init
Function description	Initialize the CAN module
Input parameter	CAN_InitTypeDef
Output parameter	None
Return value	None
Requirement	None

CANxMap：Select TX, RX pin

CANxMap	Description
CAN1_TX_PE24_RX_PE25	CAN1 Tx is PE24，Rx is PE25

CAN_BaudRateSelect：Select the baud rate

CAN_BaudRateSelect	Description
CAN_SPEED_33K	Baud rate 33K
CAN_SPEED_83K	Baud rate 83K
CAN_SPEED_50K	Baud rate 50K
CAN_SPEED_100K	Baud rate 100K
CAN_SPEED_125K	Baud rate 125K
CAN_SPEED_250K	Baud rate 250K
CAN_SPEED_500K	Baud rate 500K
CAN_SPEED_1000K	Baud rate 1000K

FilterEnable：Enable filter or not

FilterEnable	Description
ENABLE	Enable filter
DISABLE	Disable filter

Example : Configure CAN module with communication speed 500KHz, PE24 as TX and PE25 as RX, disable filter.

```
CAN_InitTypeDefCAN_Struct0;
CAN_Struct0.CAN_BaudRateSelect= CAN_SPEED_500K ;
CAN_Struct0.CANxMap           = CAN1_TX_PE24_RX_PE25;
CAN_Struct0.FilterEnable      = DISABLE;
CAN_Init(&CAN_Struct0);
```

CAN_EnableReceiveMB

Function name	CAN_EnableReceiveMB
Function description	Enable the receive Massage Buffer
Input parameter1	CANx：CAN1
Input parameter2	RxMessage：Rx Massage Buffer array
Output parameter	None
Return value	None
Requirement	Finish initializing CAN

Example：CAN_EnableReceiveMB(CAN1,CAN_RxMessage1);

CAN_Receive

Function name	CAN_Receive
Function description	CAN receive message
Input parameter1	CANx：CAN1
Input parameter2	RxMessage：Rx Massage Buffer array
Output parameter	None
Return value	None
Requirement	Finish initializing CAN

Example：CAN_ Receive (CAN1,CAN_RxMessage1);

CAN_Transmit

Function name	CAN_Transmit
Function description	CAN send message
Input parameter1	CANx：CAN1
Input parameter2	TxMessage：Tx Massage Buffer array
Output parameter	None
Return value	None
Requirement	Finish initializing CAN

Example：CAN_Transmit(CAN1,CAN_TxMessage1);

CAN_ITConfig

Function name	CAN_ITConfig
Function description	Configure interrupt of CAN module.
Input parameter1	CANx：CAN1
Input parameter2	CAN_IT： CAN_IT_MB0 ~ CAN_IT_MB15：MBn receive interrupt State：ENABLE、DISABLE
Output parameter	None
Return value	None
Requirement	Finish initializing CAN

Example：Enable CAN1 interrupt.

CAN_ITConfig(CAN1,CAN_IT_MB0, ENABLE);

CAN_GetITStatus

Function name	CAN_GetITStatus
Function description	Get interrupt status of CAN module.
Input parameter1	CANx：CAN1
Input parameter2	CAN_IT： CAN_IT_MB0 ~ CAN_IT_MB15：MBn receive interrupt
Output parameter	None
Return value	SET、RESET
Requirement	Finish initializing CAN

Example：Get interrupt status

CAN_GetITStatus(CAN1, CAN_IT_MB0)

CAN_ClearITPendingBit

Function name	CAN_ClearITPendingBit
Function description	Clear interrupt bit of CAN
Input parameter1	CANx：CAN1
Input parameter2	CAN_IT： CAN_IT_MB0 ~ CAN_IT_MB15：MBn receive interrupt
Output parameter	None
Return value	None
Requirement	Finish initializing CAN

Example：Clear pending bit.

CAN_ClearITPendingBit(CAN1, CAN_IT_MB0);

CAN_ClearAllITPendingBit

Function name	CAN_ClearAllITPendingBit
Function description	Clear all interrupt bit of CAN
Input parameter1	CANx：CAN1
Output parameter	None
Return value	None
Requirement	Finish initializing CAN

Example：Clear all pending bits.

CAN_ClearAllITPendingBit(CAN1);

FTM

The FlexTimer module (FTM) is a two-to-eight channel timer that supports input capture, output compare, and the generation of PWM signals to control electric motor and power management applications. The FTM time reference is a 16-bit counter that can be used as an unsigned or signed counter.

FTM Function List

Function name	Description
FTM_Init	Initialize the FTM module
FTM_PWM_ChangeDuty	Change PWM duty

FTM_Init

Function name	FTM_Init
Function description	Initialize the FTM module
Input parameter	FTM_InitTypeDef
Output parameter	None
Return value	None
Requirement	None

FTMxMAP：Select the pin and type

FTMxMAP	Description
FTM0_CH0_PC1	FTM0 module , channel 0，PortC1
FTM0_CH5_PD5	FTM0 module , channel 5，PortD5
FTM0_CH0_PA12	FTM1 module , channel 0，PortA12
FTM0_CH1_PB19	FTM2 module , channel 1，PortB19

FTM_Mode_TypeDef：Select the FTM mode

FTM_Mode_TypeDef	Description
FTM_Mode_EdgeAligned	Select edge aligned mode
FTM_Mode_CenterAligned	Select center aligned mode
FTM_Mode_Combine	Select combine mode
FTM_Mode_Complementary	Select complementary mode

Example:
Configure PORTC1 as PWM output, frequency 1KHz, edge aligned mode, duty 40%

```
FTM_InitTypeDef FTM_InitStruct1;
FTM_InitStruct1.Frequency     =    1000;
FTM_InitStruct1.FTMxMAP    =    FTM0_CH0_PC1;
FTM_InitStruct1.FTM_Mode    =FTM_Mode_EdgeAligned;
FTM_InitStruct1.InitalDuty     =    4000;
FTM_Init(&FTM_InitStruct1);
```

FTM_PWM_ChangeDuty

Function name	FTM_PWM_ChangeDuty
Function description	Change PWM duty
Input parameter1 **Input parameter2**	FTMxMAP : FTM module pin PWMDuty : FTM duty，precision is 0.1
Output parameter	None
Return value	None
Requirement	Finish initializing FTM

Example : Change duty to 50%.

```
FTM_PWM_ChangDuty(FTM0_CH0_PC1,5000);
```

RTC

The RTC module includes independent power supply, POR and 32kHz crystal oscillator. 32-bit seconds counter with roll-over protection and 32-bit alarm.

RTC Function List

Function name	Description
RTC_Init	Initialize the RTC module
RTC_SetData	Set the RTC data
RTC_GetData	Get the RTC data

RTC_init

Function name	RTC_Init
Function description	Initialize the RTC module
Input parameter	None
Output parameter	None
Return value	None
Requirement	None

Example:
Init the RTC module

```
RTC_init();
```

RTC_SetData

Function name	RTC_SetData
Function description	Set the RTC data
Input parameter	RTC_CalanderTypeDef
Output parameter	None
Return value	None
Requirement	Finish initializing RTC

Example:
Set the RTC time

```
RTC_CalanderTypeDef myRTC;
    myRTC.Year      = 2014;
    myRTC.Month     = 8;
    myRTC.Date      = 25;
    myRTC.Hour      =10;
    myRTC.Minute    = 30;
    myRTC.Second    = 30;
    RTC_SetData(&myRTC);
```

RTC_GetData

Function name	RTC_GetData
Function description	Get the RTC data
Input parameter	RTC_CalanderTypeDef
Output parameter	None
Return value	RTC_CalanderTypeDef
Requirement	Finish initializing RTC

Example:
Get the RTC time

```
RTC_CalanderTypeDef myRTC
RTC_ReadData(&myRTC);
UART_printf("Year:%d  Month:%d  Date:%d  %d:%d:%d
\r\n",myRTC.Year,myRTC.Month,myRTC.Date,myRTC.Hour,myRTC.Minute,myRTC.Second);
```

USB

Currently, Forenex does not write our own code for USB. Customer can use Freescale MQX's code to access USB.

SDHC

Currently, Forenex does not write our own code for SDHC. Customer can use Freescale MQX's code to access SDHC.

FlexBus

For the FlexBus hardware settings, we configure it as "8080 like" interface in the library. So, customers can access the flexbus with "flexbus_base_address", which is 0x60000000 to read/write data with flexbus. If the devices connected to flexbus need to send commands/index, the "flexbus_address_mask" is 0x00200000. So, to send/receive command/index with flexbus, you can use (flexbus_base_address | flexbus_address_mask), which is 0x60200000.

For example, if customer connect Forenex's product FG875D to the flexbus. We have below definition:

```
#define FLEX_ADDRESS_MASK 0x00200000
#define FLEX_BASE_ADDRESS 0x60000000
#define FLEX_DC_ADDRESS (FLEX_ADDRESS_MASK | FLEX_BASE_ADDRESS)
```

Below is the function to write commands to FG875D.

```
void FG875D_CmdWrite(u16_t Cmd)
{
    *((u16_t *)FLEX_DC_ADDRESS) = Cmd;
}
```

Below is the function to write data to FG875D.

```
void FG875D _DataWrite(u16_t Data)
{
    *((u16_t *)FLEX_BASE_ADDRESS) = Data;
}
```

Below is the function to read status from FG875D.

```
u16_t FG875D _StatusRead(void)
{
    u16_t u16Data;
    u16Data = *((u16_t *)FLEX_DC_ADDRESS);
    return u16Data;
}
```

Below is the function to read data from FG875D.

```
u16_t FG875D _DataRead(void)
{
    u16_t u16Data;
    u16Data = *((u16_t *)FLEX_BASE_ADDRESS);
    return u16Data;
}
```

Another example is that if user connect memory module to flexbus.
We have below definition:

```
#define FLEX_BASE_ADDRESS 0x60000000
```

Below is the function to write data to memory module

```
void memory_Write(u32_t offset, u16_t Data)
{
    *((u16_t *)(FLEX_BASE_ADDRESS+ offset)) = Data;
}
```

Below is the function to read data from memory module.

```
u16_t memory_Read(u32_t offset)
{
    u16_t u16Data;
    u16Data = *((u16_t *)(FLEX_BASE_ADDRESS+ offset));
    return u16Data;
}
```

3. Downloading APP into FBM4-K60512

● When the user's application software has completed and compiled over MQX or Keil_C. to generate a filename "template.hex" in sub-directory of "IntFlashDebug".
The file "template.hex" must be changed into " template.K60". And then, save it into SD card and insert card into socket (J2) for down-load automatically later.
Pushing and hold the key (re-load) then toggled Reset key until status LED is turned to red light. During the period of red light, the FBM4-K60512's boot-loader will looks for one specific file with the attribute of /.K60/ in SD card. And then, Update the file (xxxx.K60) into the ROM of BM4-K60512 automatically until the red light turned back to green light. (Notice to make sure only one of .k60 file is stored in SD card).
Pushing the reset key again to implement the user's application software.

● Refer to the Reference Manual of FBM4-K60512 ("FBM4-K60512_RM_Vxx") for more detail description about location of push-bottom and SD socket.