UM0233
User manual

STR91xFA software library

Introduction

About this manual

This document is the STR91xFA software library user manual. It describes the STR91xFA peripheral software library: a collection of routines, data structures and macros that cover the features of each peripheral.

This manual is structured as follows: some definitions, document conventions and software library rules are provided in *Section 1*. *Section 2* provides a detailed description of the software library: The package content, the installation steps, the library structure and an example on how to use the library. Finally, Sections *3* to *20* describe the software library, peripheral configuration structure and functions description for each peripheral in detail.

About STR91xFA library

The STR91xFA software library is a software package consisting of device drivers for all standard STR91xFA peripherals. You can use any STR91xFA device in applications without in-depth study of each peripheral specification. As a result, using this library can save you a lot of the time that you would otherwise spend in coding and also the cost of developing and integrating your application.

Each device driver consists of a set of functions covering the functionality of the peripheral. Since all the STR91xFA peripherals and their corresponding registers are memory-mapped, a peripheral can be easily controlled using ‘C’ code. The source code, developed in ‘C’, is fully documented. A basic knowledge of ‘C’ programming is required.

The library contains a complete software in ‘C’ that can be easily ported to any ARM compatible ‘C’ compiler.

Contents

1 Document and library rules

The user manual document and the software library use the conventions described in the sections below.

1.1 Abbreviations

The table below describes the different abbreviations used in this document

Acronym	Peripheral / Unit
ADC	Analog-to-Digital Converter
CAN	Controller Area Network
SCU	System Control Unit
DMA	DMA Controller
VIC	Vectored Interrupt Controller
GPIO	General Purpose I/O Ports
I2C	I^2C Interface module
RTC	Real Time Clock
WIU	Wake-Up Interrupt Unit
AHBAPB	AHB/APB Bridges
MC	3-phase induction Motor Controller (MC)
FMI	Flash Memory Interface
EMI	External Memory Interface
SSP	Synchronous Serial Peripheral
TIM	Standard Timer
UART	Universal Asynchronous Receiver Transmitter
WDG	Watchdog Timer

The Software library uses the following naming conventions:

- ***PPP*** is used as reference to any peripheral acronym, e.g. ***ADC***. See the section above for more information on peripheral acronyms.
- System and source/header file names are preceded by '91x_', e.g. ***91x_conf.h***.
- Constants used in one file are defined within this file. A constant used in more than one file is defined in a header file.
- Registers are considered as constants. Their names are in upper case letters and have in most cases the same acronyms as in the STR91xFA reference manual document.
- Peripheral function names are preceded with the corresponding peripheral acronym in upper case followed by an underscore. The first letter in each word is upper case, e.g. ***SSP_SendData***. Only one underscore is allowed in a function name to separate the peripheral acronym from the rest of the function name.
- Functions for initializing PPP peripheral according to the specified parameters in the ***PPP_InitTypeDef*** are named ***PPP_Init***, e.g. ***TIM_Init***.
- Functions for deinitializing PPP peripheral registers to their default reset values are named ***PPP_DeInit***, e.g. ***TIM_DeInit***.
- Functions for filling the ***PPP_InitTypeDef*** structure with the reset value of each member are named ***PPP_StructInit***, e.g. ***UART_StructInit***.
- Functions for enabling or disabling the specified PPP peripheral are named ***PPP_Cmd***, e.g. ***SSP_Cmd***.
- Functions for enabling or disabling an interrupt source of the specified PPP peripheral are named ***PPP_ITConfig***, e.g. ***DMA_ITConfig***.
- Functions for enabling or disabling the DMA interface of the specified PPP peripheral are named ***PPP_DMACmd***, e.g. ***SSP_DMACmd***.
- Functions used to configure a peripheral function end with Config, e.g. ***UART_DTRConfig***.
- Functions for checking whether the specified PPP flag is set or not are named ***PPP_GetFlagStatus***, e.g. ***I2C_GetFlagStatus***.
- Functions for clearing a PPP flag are named ***PPP_ClearFlag***, e.g. ***I2C_ClearFlag***.
- Functions for checking whether the specified PPP interrupt has occurred or not are named ***PPP_GetITStatus***, e.g. ***DMA_GetITStatus***.
- Functions for clearing a PPP interrupt pending bit are named ***PPP_ClearITPendingBit***, e.g. ***WDG_ClearITPendingBit***.

1.2 Coding Rules

The following rules are used in the Software Library.

- Specific types are defined for variables whose type and size are fixed. These types are defined in the file ***91x_type.h***:

```
typedef unsigned long            u32;
typedef unsigned short           u16;
typedef unsigned char            u8;

typedef signed long              s32;
typedef signed short             s16;
typedef signed char              s8;

typedef volatile unsigned long   vu32;
typedef volatile unsigned short  vu16;
typedef volatile unsigned char   vu8;

typedef volatile signed long     vs32;
typedef volatile signed short    vs16;
typedef volatile signed char     vs8;
```

- ***bool*** type is defined in the file ***91x_type.h*** as:

```
typedef enum
{
  FALSE = 0,
  TRUE  = !FALSE
} bool;
```

- ***FlagStatus*** & ***ITStatus*** types are defined in the file ***91x_type.h***. Two values can be assigned to this variable: ***SET*** or ***RESET***.

```
typedef enum
{
  RESET = 0,
  SET   = !RESET
} FlagStatus, ITStatus;
```

- ***FunctionalState*** type is defined in the file ***91x_type.h***. Two values can be assigned to this variable: ***ENABLE*** or ***DISABLE***.

```
typedef enum
{
  DISABLE = 0,
  ENABLE  = !DISABLE
} FunctionalState;
```

- ***ErrorStatus*** type is defined in the file ***91x_type.h***. Two values can be assigned to this variable: ***SUCCESS*** or ***ERROR***.

```
typedef enum
{
  ERROR   = 0,
  SUCCESS = !ERROR
} ErrorStatus;
```

- Pointers to peripherals are used to access the peripheral control registers. Peripheral pointers point to data structures that represent the mapping of the peripheral control registers. A structure is defined for each peripheral in the file ***91x_map.h***. The example below illustrates the ***SSP*** register structure declaration:

```
/*----------------------- Synchronous Serial Peripheral ----------------------*/
typedef struct
{
  vu16 CR0;         /* Control Register 1                   */
  vu16 EMPTY1;
  vu16 CR1;         /* Control Register 2                   */
  vu16 EMPTY2;
  vu16 DR;          /* Data Register                        */
  vu16 EMPTY3;
  vu16 SR;          /* Status Register                      */
  vu16 EMPTY4;
  vu16 PR;          /* Clock Prescaler Register              */
  vu16 EMPTY5;
  vu16 IMSCR;       /* Interrupt Mask Set or Clear Register */
  vu16 EMPTY6;
  vu16 RISR;        /* Raw Interrupt Status Register        */
  vu16 EMPTY7;
  vu16 MISR;        /* Masked Interrupt Status Register     */
  vu16 EMPTY8;
  vu16 ICR;         /* Interrupt Clear Register             */
  vu16 EMPTY9;
  vu16 DMACR;       /* DMA Control Register                 */
  vu16 EMPTY10;
}SSP_TypeDef;
```

Register names are the register acronyms written in upper case for each peripheral. EMPTYi (i is an integer that indexes the reserved field) replaces a reserved field.

Peripherals are declared in ***91x_map.h*** file. The following example shows the declaration of the *SSP* peripheral:

```
#ifndef EXT
 #Define EXT extern
#endif
...
#define AHB_APB_BRDG1_U    (0x5C000000) /* AHB/APB Bridge 1 UnBuffered Space */
#define AHB_APB_BRDG1_B    (0x4C000000) /* AHB/APB Bridge 1 Buffered Space   */
...
#define APB_SSP0_OFST      (0x00007000) /* Offset of SSP0 */
...
#ifndef Buffered
#define AHBAPB1_BASE           (AHB_APB_BRDG1_U)
...
#else /* Buffered */
...
#define AHBAPB1_BASE           (AHB_APB_BRDG1_B)

/* SSP0 Base Address definition*/
#define SSP0_BASE          (AHBAPB1_BASE + APB_SSP0_OFST)

...
/* SSP0 peripheral declaration*/
#ifndef DEBUG
...
#define SSP0         ((SSP_TypeDef *) SSP0_BASE)
```

```
...
#else
...
#ifdef _SSP0
EXT SSP_TypeDef           *SSP0;
#endif /*_SSP0 */
...
#endif
```

To enter debug mode you have to define the label *DEBUG* in the file ***91x_conf.h***. Debug mode allows you to see the contents of peripheral registers but it uses more memory space. In both cases *SSP0* is a pointer to the first address of *SSP0* peripheral.

The *DEBUG* variable is defined in the file ***91x_conf.h*** as follows:

```
#define DEBUG
```

DEBUG mode is initialized as follows in the ***91x_lib.c*** file:

```
#ifdef DEBUG
void debug(void)
{
  ...
  #ifdef _SSP0
  SSP0 = (SSP_TypeDef *)SSP0_BASE;
  #endif /*_SSP0 */
  ...
}
#endif /* DEBUG*/
```

To include the *SSP* peripheral library in your application, define the label *_SSP* and to access the *SSPn* peripheral registers, define the label *_SSPn*, i.e to access the registers of *SSP1* peripheral, *_SSP1* label must be defined in ***91x_conf.h*** file. *_SSP* and *_SSPn* labels are defined in the file ***91x_conf.h*** as follows:

```
#define _SSP
#define _SSPn
```

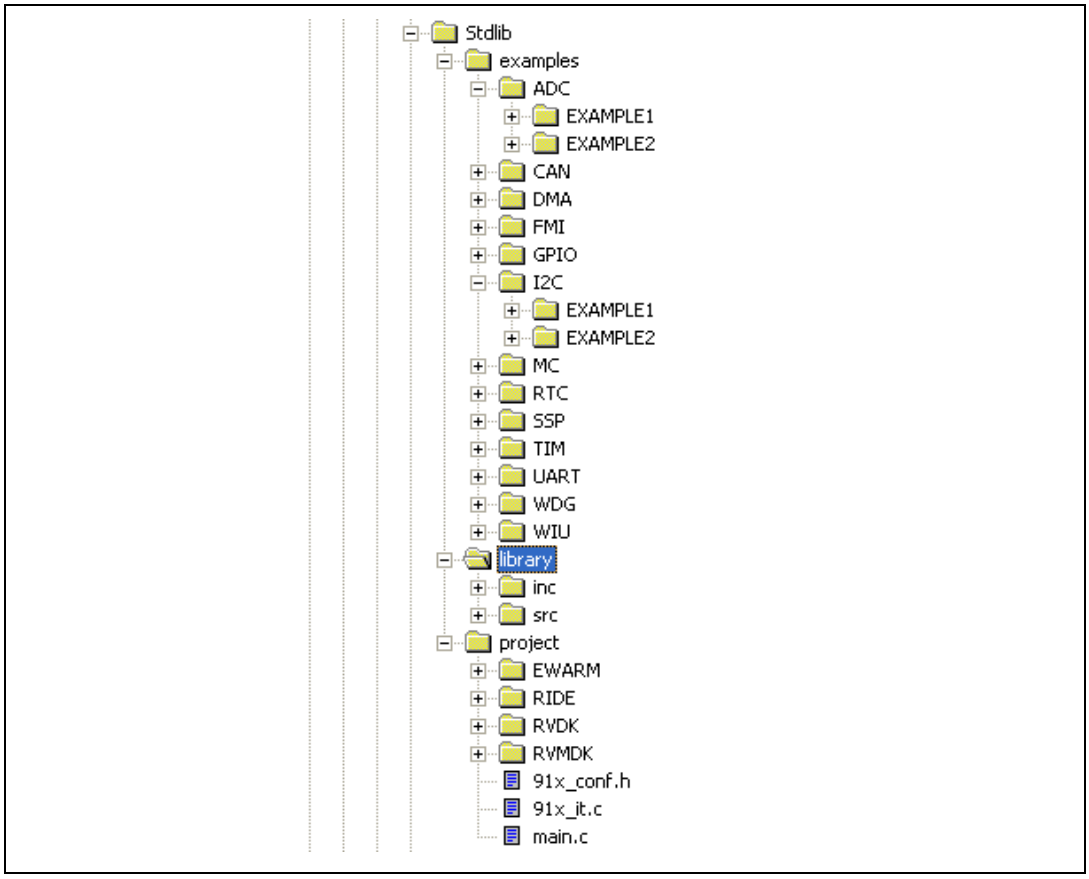
Each peripheral has several dedicated registers which contain different flags. Registers are defined within a dedicated structure for each peripheral. Flag definition is adapted to each peripheral case (In almost cases defined in ***91x_ppp.h*** file). Flags are defined as acronyms written in upper case and prefixed by '***PPP_Flag_***' prefix.

2 Software library

2.1 Package description

The software library is supplied in one single zip package. The extraction of the zip file will give the one folder ***"STR91x\StdLib"*** containing the following sub-directories:

Figure 1. Software Library Directory Structure



2.2 Examples

This directory contains for each peripheral sub-directory, the minimum set of files needed to run a typical example on how to use a peripheral:

- ***Readme.txt***: a brief text file describing the example and how to make it work,
- ***91x_conf.h****:* the header file to configure the used peripherals and miscellaneous defines,
- ***91x_it.c***: the source file containing the interrupt handlers (the function bodies may be empty if not used),
- ***main.c***: the example program,

Note: All examples are independent from any software tool chain.

2.3 Library

This directory contains all the subdirectories and files that form the core of the library:

- ***inc*** sub-directory contains the software library header files that do not need to be modified by the user:
  - ***91x_type.h***: Contains the common data types and enumeration used in all other files,
  - ***91x_map.h***: Contains the peripheral memory mapping and register data structures,
  - ***91x_lib.h***: Main header file including all other headers,
  - ***91x_ppp.h*** (one header file per peripheral): contains the function prototypes, data structures and enumeration.
- ***src*** sub-directory contains the software library source files that do not need to be modified by the user:
  - ***91x_ppp.c*** (one source file per peripheral): contains the function bodies of each peripheral.

Note: All library files are independent from any software toolchain.

2.4 Project

This directory contains a standard template project program that compiles all library files and also all the user modifiable files needed to create a new project:

- ***91x_conf.h***: The configuration header file with all peripherals defined by default.
- ***91x_it.c***: The source file containing the interrupt handlers (the function bodies are empty in this template).
- ***main.c***: The main program body.

- ***EWARM***, ***RVDK, RIDE, RVMDK***: For each toolchain usage, refer to the Readme.txt file available in the same sub-directory.

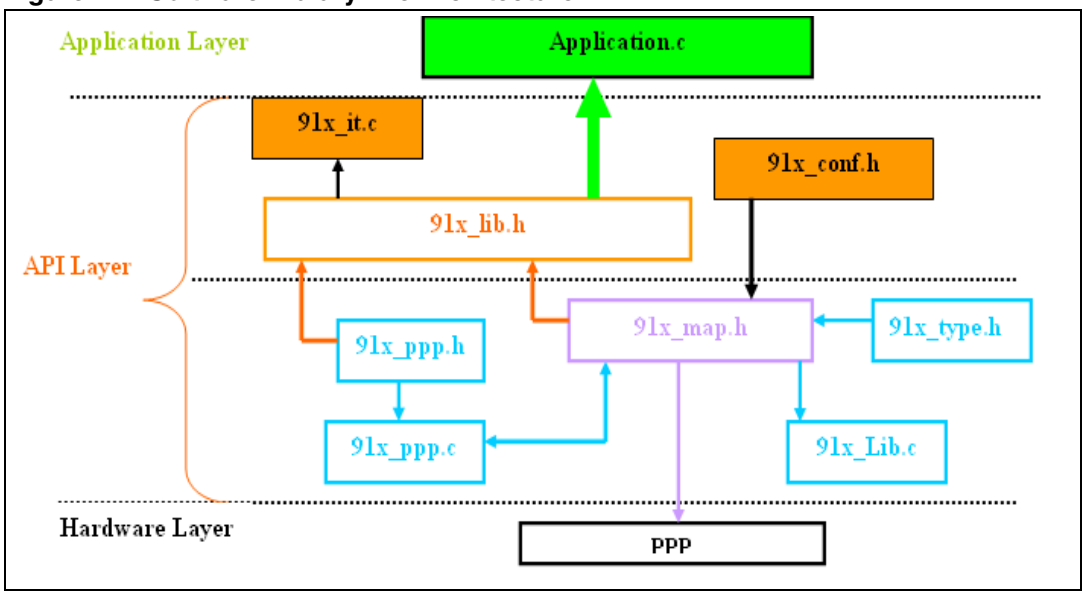
2.5 File description

Several files are used in the software library. The following tables enumerate and describe the different files used in the software library.

File name	Description
91x_conf.h	Parameter configuration file. It should be modified by the user to specify several parameters to interface with the library before running any application. You can enable or disable peripherals if you use the template.
main.c	The main example program body.
91x_it.c	Peripheral interrupt functions file. You can modify it by including the code of interrupt functions used in your application. In case of multiple interrupt requests mapped on the same interrupt vector, the function polls the interrupt flags of the peripheral to establish the exact source of the interrupt. The names of these functions are already provided in the software library.
91x_lib.h	Header file including all the peripheral header files. It is the only file to be included in the user application to interface with the library.
91x_lib.c	Debug mode initialization file. It includes the definition of variable pointers each one pointing to the first address of a specific peripheral and the definition of one function called when you choose to enter debug mode. This function initializes the defined pointers.
91x_map.h	This file implements memory mapping and physical registers address definition for both development and debug modes. This file is supplied with all peripherals.
91x_type.h	Common declarations file. It includes common types and constants used by all peripheral drivers.
91x_ppp.c	Driver source code file of PPP peripheral written in C language.
91x_ppp.h	Header file of PPP peripheral. It includes the definition of PPP peripheral functions and variables used within these functions.

The software library architecture and file inclusion relationship are shown in *Figure 2*.

Figure 2. Software Library File Architecture



Each peripheral has a source code file ***91x_ppp.c*** and a header file ***91x_ppp.h***. The ***91x_ppp.c*** file contains all the software functions required to use the corresponding peripheral. A single memory mapping file ***91x_map.h*** is supplied for all peripherals. This file contains all the register declarations for both development and debug modes.

The header file ***91x_lib.h*** includes all the peripheral header files. This is the only file that needs to be included in the user application to interface with the library.

2.6 How to use the Library

This section describes step-by-step how to configure and initialize a PPP peripheral.

- In your main application file, declare a ***PPP_InitTypeDef*** structure, e.g:

```
PPP_InitTypeDef  PPP_InitStructure;
```

The `PPP_InitStructure` is a working variable located in data memory that allows you to initialize one or more PPP instances.

- Fill the `PPP_InitStructure` variable with the allowed values of the structure member.

There are two ways of doing this:

– Configuration of the whole structure: in this case you should proceed as follows:

```
PPP_InitStructure.member1 = val1;
PPP_InitStructure.member2 = val2;
PPP_InitStructure.memberN = valN; /* where N is the number of
                                     the structure members */
```

Note: The previous initialization could be merged in only one line like the following:

```
PPP_InitTypeDef  PPP_InitStructure = { val1, val2, …, valN}
```

This reduces and optimizes code size.

– Configuration of a few members of a structure: in this case you should modify the `PPP_InitStructure` variable that has been already filled by a call to the ***PPP_StructInit(..)*** function. This ensures that the other members of the `PPP_InitStructure` variable have appropriate values (in most case their default values).

```
PPP_StructInit(&PPP_InitStructure);
PPP_InitStructure.memberX = valX;
PPP_InitStructure.memberY = valY; /* where X and Y are the only members
                                     that you want to configure */
```

- You have to initialize the PPP peripheral by calling the ***PPP_Init(..)*** function.

```
PPP_Init(PPP, &PPP_InitStructure);
```

- At this stage the PPP peripheral is initialized and can be enabled (if applicable) by making a call to ***PPP_Cmd(..)*** function.

```
PPP_Cmd(PPP, ENABLE);
```

To use the PPP peripheral, you can use a set of dedicated functions. These functions are specific to the peripheral and for more details refer to *Section 3 on page 22*.

Note: 1 Before configuring a peripheral, you have to enable its clock by calling the following function:

```
SCU_APBPeriphClockConfig(__PPP, ENABLE); /* For APB Peripheral */
```

OR:

```
SCU_AHBPeriphClockConfig(__PPP, ENABLE); /* For AHB Peripheral */
```

*2 **PPP_DeInit(..)** function can be used to set all PPP peripheral registers to their reset values:*

```
PPP_DeInit(PPP);
```

3 If after peripheral configuration, you want to modify one or more peripheral settings you should proceed as follows:

```
PPP_InitStucture.memberX = valX;
PPP_InitStructure.memberY = valY; /* where X and Y are the only members
                                     that user wants to modify*/
PPP_Init(PPP, &PPP_InitStructure);
```

3 Peripheral software overview

The following chapters describe each peripheral software library in detail. The related functions are fully documented. An example of use of the function is given and some important considerations are also provided.

Functions are described in the format below:

Function name	The name of the peripheral function
Function prototype	Prototype declaration
Behavior Description	Brief explanation of how the functions are executed
Input Parameter {x}	Description of the input parameters
Output parameter {x}	Description of the output parameters
Return Value	Value returned by the function
Required Preconditions	Requirements before calling the function
Called Functions	Other library functions called by the function

4 Flash Memory Interface (FMI)

The Flash Memory Interface of the STR91xFA contains two banks with a total capacity of 256+32 or 512+32 Kbytes, can be 32-bit burst read accessed and 16-bit write accessed.

4.1 FMI register structure

4.1.1 FMI register structure

The FMI register structure *FMI_TypeDef* is defined in the ***91x_map.h*** file as follows:

```
typedef struct
{
  vu32 BBSR;
  vu32 NBBSR;
  vu32 EMPTY1;
  vu32 BBADR;
  vu32 NBBADR;
  vu32 EMPTY2;
  vu32 CR;
  vu32 SR;
  vu32 BCE5ADDR;
} FMI_TypeDef;
```

The following table presents the FMI registers:

Register	Description
BBSR	Boot Bank Size Register
NBBSR	Non-Boot Bank Size Register
BBADR	Boot Bank Base Address Register
NBBADR	Non-Boot Bank Base Address Register
CR	Control Register
SR	Status Register
BCE5ADDR	BC Fifth Entry Target Address Register

The FMI interface are declared in the same file:

```
#ifndef EXT
  #define EXT extern
#endif /* EXT */

#define AHB_FMI_U          (0x54000000) /* FMI Unbuffered Space */
#define AHB_FMI_B          (0x44000000) /* FMI buffered Space   */
...
#ifndef Buffered
...
#define FMI_BASE           (AHB_FMI_U)
...
#else /* Buffered */
...
#define FMI_BASE           (AHB_FMI_B)
```

```
...
#endif /* Buffered */
...
#ifndef DEBUG
...
#define FMI                ((FMI_TypeDef *)FMI_BASE)
...
#else
...
#ifdef _FMI
EXT FMI_TypeDef            *FMI;
#endif /* _FMI */
...
#endif
```

When debug mode is used, FMI pointer is initialized in ***91x_lib.c*** file:

```
#ifdef _FMI
  FMI = (FMI_TypeDef *)FMI_BASE
#endif /* _FMI */
```

_FMI must be defined, in ***91x_conf.h*** file, to access the peripheral registers as follows:

```
...
#define _FMI
...
```

4.2 Software library functions

The following table enumerates the different functions of the FMI library.

Function Name	Description
FMI_BankRemapConfig	Configures the sizes and addresses of bank 0 and bank 1.
FMI_Config	Configures the FMI.
FMI_EraseSector	Erases a sector.
FMI_EraseBank	Erases a bank.
FMI_WriteHalfWord	Writes a halfword to a Flash memory address.
FMI_WriteOTPHalfWord	Writes a halfword to an OTP sector address.
FMI_ReadWord	Reads the corresponding data.
FMI_ReadOTPData	Reads data from the OTP sector.
FMI_GetFlagStatus	Checks whether the specified FMI flag is set or not.
FMI_GetReadWaitStateValue	Gets the current Read wait state value.
FMI_GetWriteWaitStateValue	Gets the current write wait state value.
FMI_SuspendEnable	Enables the Suspend command.
FMI_ResumeEnable	Resumes the suspended command.
FMI_ClearFlag	Clears the FMI flags on the corresponding bank.
FMI_WriteProtectionCmd	Enables or disables the write protection for the specified sector.
FMI_GetWriteProtectionStatus	Gets the write protection status for the specified sector.
FMI_WaitForLastOperation	Waits until the last Operation (Write halfword, Erase sector and Erase bank) completion.

4.2.1 FMI_BankRemapConfig

Function Name	FMI_BankRemapConfig
Function Prototype	`void FMI_BankRemapConfig(u8 FMI_BootBankSize, u8 FMI_NonBootBankSize, u32 FMI_BootBankAddress, u32 FMI_NonBootBankAddress)`
Behavior Description	Configures the addresses and sizes of bank 0 and bank 1.
Input Parameter1	FMI_BootBankSize: specifies the boot bank size. Refer to section "*FMI_BootBankSize on page 26*" for more details on the allowed values of this parameter.
Input Parameter2	FMI_NonBootBankSize: specifies the non boot bank size. Refer to section "*FMI_NonBootBankSize on page 26*" for more details on the allowed values of this parameter.
Input Parameter3	FMI_BootBankAddress: specifies the boot bank address.
Input Parameter4	FMI_BootBankAddress: specifies the non boot bank address.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

FMI_BootBankSize

To select the FMI boot bank size, use one of the following values:

FMI_BootBankSize	Meaning
0	32 KBytes
1	64 KBytes
2	128 KBytes
3	256 KBytes
4	512 KBytes
...	...
0xB	64 MBytes

FMI_NonBootBankSize

To select the FMI non boot bank size, use one of the following values:

FMI_NonBootBankSize	Meaning
0	8 KBytes
1	16 KBytes
2	32 KBytes
3	64 KBytes
...	...
0xD	64 MBytes

```
Example:
/*To configure bank 1 (32KByte) at address 0x0 and bank 0 (512KBytes) at address
0x80000 , where bank 0 is the boot bank*/
FMI_BankConfig(0x4, 0x2, 0x80000, 0x0);
```

Note: When bank 1 is hardware remapped to address 0 (bank 1 is the boot bank), in the 91x_fmi.h file, the //#define Remap_Bank_1 line should be decommented.

4.2.2 FMI_Config

Function Name	FMI_Config
Function Prototype	`void FMI_Config(u16 FMI_ReadWaitState, u32 FMI_WriteWaitState, u16 FMI_PWD, u16 FMI_LVDEN, u16 FMI_FreqRange)`
Behavior Description	Configures the FMI.
Input Parameter 1	FMI_ReadWaitState: specifies the read wait state value. Refer to section "*FMI_ReadWaitState on page 27*" for more details on the allowed values of this parameter.
Input Parameter 2	FMI_WriteWaitState: specifies the read wait state value. Refer to section "*FMI_WriteWaitState on page 28*" for more details on the allowed values of this parameter.
Input Parameter 3	FMI_PWD: specifies the power down mode status. Refer to section "*FMI_PWD on page 28*" for more details on the allowed values of this parameter.
Input Parameter 4	FMI_LVDEN: specifies the low voltage detector mode status. Refer to section "*FMI_LVDEN on page 28*" for more details on the allowed values of this parameter.
Input Parameter 5	FMI_FreqRange: specifies the working frequency range. Refer to section "*FMI_FreqRange on page 28*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

FMI_ReadWaitState

The FMI read wait states that can be selected are listed in the following table:

FMI_ReadWaitState	Meaning
FMI_READ_WAIT_STATE_1	1 read wait state
FMI_READ_WAIT_STATE_2	2 read wait states
FMI_READ_WAIT_STATE_3	3 read wait states

FMI_WriteWaitState

The FMI write wait states that can be selected are listed in the following table:

FMI_WriteWaitState	Meaning
FMI_WRITE_WAIT_STATE_0	0 write wait state
FMI_WRITE_WAIT_STATE_1	1 write wait states

FMI_PWD

FMI power down mode can be selected as listed in the following table:

FMI_PWD	Meaning
FMI_PWD_ENABLE	Enable the PWD
FMI_PWD_DISABLE	Disable the PWD

FMI_LVDEN

The FMI low voltage detector can be selected as listed in the following table:

FMI_LVDEN	Meaning
FMI_LVD_ENABLE	Enable the LVD
FMI_LVD_DISABLE	Disable the LVD

FMI_FreqRange

The FMI frequency ranges that can be selected are listed in the following table:

FMI_FreqRange	Meaning
FMI_FREQ_LOW	Low working frequency (up to 66 MHz)
FMI_FREQ_HIGH	High working frequency (above 66 MHz)

Example:

```
/*To configure the FMI as follows: 2 read wait states, 1 write wait state, PWD
enabled, LVD enabled and a low working frequency*/
FMI_Config(FMI_READ_WAIT_STATE_2, FMI_WRITE_WAIT_STATE_1, FMI_PWD_ENABLE,
FMI_LVD_ENABLE, FMI_FREQ_LOW);
```

4.2.3 FMI_EraseSector

Function Name	FMI_EraseSector
Function Prototype	`void FMI_EraseSector(vu32 FMI_Sector)`
Behavior Description	Erases a sector.
Input Parameter	FMI_Sector: specifies the sector to be erased. Refer to section "*FMI_Sectors on page 29*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

FMI_Sectors

To select the FMI sectors, use one of the following values:

FMI_Sector	Meaning
FMI_B0S0	FMI bank 0 sector 0
FMI_B0S1	FMI bank 0 sector 1
FMI_B0S2	FMI bank 0 sector 2
FMI_B0S3	FMI bank 0 sector 3
FMI_B0S4	FMI bank 0 sector 4
FMI_B0S5	FMI bank 0 sector 5
FMI_B0S6	FMI bank 0 sector 6
FMI_B0S7	FMI bank 0 sector 7
FMI_B1S0	FMI bank 1 sector 0
FMI_B1S1	FMI bank 1 sector 1
FMI_B1S2	FMI bank 1 sector 2
FMI_B1S3	FMI bank 1 sector 3

Example:

/*To erase sector 3 in bank 0*/

```
u8 FMI_Timeout_Status;
FMI_EraseSector(FMI_B0S3);
FMI_Timeout_Status = FMI_WaitForLastOperation(FMI_BANK_0);
```

4.2.4 FMI_EraseBank

Function Name	FMI_EraseBank
Function Prototype	`void FMI_EraseBank(vu32 FMI_Bank)`
Behavior Description	Erases a bank.
Input Parameter	FMI_Bank: specifies the bank to be erased. Refer to section "*FMI_Bank on page 30*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

FMI_Bank

To select the FMI bank, use one of the following values:

FMI_Bank	Meaning
FMI_BANK_0	FMI bank 0
FMI_BANK_1	FMI bank 1

Example:

```
/*To erase bank 0*/
u8 FMI_Timeout_Status;
FMI_EraseBank(FMI_BANK_0);
FMI_Timeout_Status = FMI_WaitForLastOperation(FMI_BANK_0);
```

4.2.5 FMI_WriteHalfWord

Function Name	FMI_WriteHalfWord
Function Prototype	`void FMI_WriteHalfWord(u32 FMI_Address, u16 FMI_Data)`
Behavior Description	Writes a halfword to a Flash memory address.
Input Parameter1	FMI_Address: specifies the address offset where the data are to be written.
Input Parameter2	FMI_Data: the data to be written.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

Example:

```
/*To write the data 0x1234 to the address 0x4000*/
u8 FMI_Timeout_Status;
FMI_WriteHalfWord(0x4000,0x1234);
FMI_Timeout_Status = FMI_WaitForLastOperation(FMI_BANK_0);
```

4.2.6 FMI_WriteOTPHalfWord

Function Name	FMI_WriteOTPHalfWord
Function Prototype	`void FMI_WriteOTPHalfWord(u8 FMI_OTPHWAddress, u16 FMI_OTPData)`
Behavior Description	Writes a halfword to the specified OTP sector address.
Input Parameter1	FMI_OTPHWAddress: specifies the address offset where the data is to be written. Refer to section "*FMI_OTPHWAddress on page 31*" for more details on the allowed values of this parameter.
Input Parameter2	FMI_OTPData: the data to be written.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

FMI_OTPHWAddress

To select the FMI OTP halfword addresses, use one of the following values:

FMI_OTPAddress	Meaning
FMI_OTP_LOW_HALFWORD_0	FMI OTP low halfword 0
FMI_OTP_HIGH_HALFWORD_0	FMI OTP high halfword 0
FMI_OTP_LOW_HALFWORD_1	FMI OTP low halfword 1
FMI_OTP_HIGH_HALFWORD_1	FMI OTP high halfword 1
FMI_OTP_LOW_HALFWORD_2	FMI OTP low halfword 2
FMI_OTP_HIGH_HALFWORD_2	FMI OTP high halfword 2
FMI_OTP_LOW_HALFWORD_3	FMI OTP low halfword 3
FMI_OTP_HIGH_HALFWORD_3	FMI OTP high halfword 3
FMI_OTP_LOW_HALFWORD_4	FMI OTP low halfword 4
FMI_OTP_HIGH_HALFWORD_4	FMI OTP high halfword 4
FMI_OTP_LOW_HALFWORD_5	FMI OTP low halfword 5
FMI_OTP_HIGH_HALFWORD_5	FMI OTP high halfword 5
FMI_OTP_LOW_HALFWORD_6	FMI OTP low halfword 6
FMI_OTP_HIGH_HALFWORD_6	FMI OTP high halfword 6
FMI_OTP_LOW_HALFWORD_7	FMI OTP low halfword 7
FMI_OTP_HIGH_HALFWORD_7	FMI OTP high halfword 7

Example:

```
/*To write the data 0x1234 to the LSB of the word 2*/
u8 FMI_Timeout_Status;
FMI_WriteOTPHalfWord(FMI_OTP_LOW_HALFWORD_2,0x1234);
FMI_Timeout_Status = FMI_WaitForLastOperation(FMI_BANK_1);
```

4.2.7 FMI_ReadWord

Function Name	FMI_ReadWord
Function Prototype	`u32 FMI_ReadWord(u32 FMI_Address)`
Behavior Description	Reads the corresponding data.
Input Parameter	FMI_Address: specifies the address of the word to be read.
Output Parameter	None
Return Parameter	The data contained in the specified address.
Required preconditions	...
Called functions	None

Example:

```
/*To read the data contained in the address 0x6000*/
u32 DATA;
DATA = FMI_ReadWord(0x6000);
```

4.2.8 FMI_ReadOTPData

Function Name	FMI_ReadOTPData
Function Prototype	`u32 FMI_ReadOTPData(u8 FMI_OTPAddress)`
Behavior Description	Reads data from the OTP sector.
Input Parameter	FMI_OTPAddress: specifies the address of the data to be read. Refer to section "*FMI_OTPAddress on page 32*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The data to be read from the OTP sector.
Required preconditions	...
Called functions	None

FMI_OTPAddress

To select the FMI OTP addresses, use one of the following values:

FMI_OTPAddress	Meaning
FMI_OTP_WORD_0	FMI OTP word 0
FMI_OTP_WORD_1	FMI OTP word 1
FMI_OTP_WORD_2	FMI OTP word 2
FMI_OTP_WORD_3	FMI OTP word 3
FMI_OTP_WORD_4	FMI OTP word 4
FMI_OTP_WORD_5	FMI OTP word 5
FMI_OTP_WORD_6	FMI OTP word 6
FMI_OTP_WORD_7	FMI OTP word 7

Example:

```
/*To read the 2nd word from the OTP sector*/
vu16 OTP_Data;
OTP_Data= FMI_ReadOTPData(FMI_OTP_WORD_2);
```

4.2.9 FMI_GetFlagStatus

Function Name	FMI_GetFlagStatus
Function Prototype	`FlagStatus FMI_GetFlagStatus(u8 FMI_Flag, vu32 FMI_Bank)`
Behavior Description	Checks whether the specified FMI flag is set or not.
Input Parameter1	FMI_Flag: specifies the flag to check. Refer to section "*FMI_Flag on page 33*" for more details on the allowed values of this parameter.
Input Parameter2	FMI_Bank: specifies the corresponding bank. Refer to section "*FMI_Bank on page 33*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of FMI_Flag (SET or RESET).
Required preconditions	...
Called functions	None

FMI_Flag

The FMI flags that can be read are listed in the following table:

FMI_Flag	Meaning
FMI_FLAG_SPS	Sector Protection status
FMI_FLAG_PSS	Program suspend status
FMI_FLAG_PS	Program status
FMI_FLAG_ES	Erase status
FMI_FLAG_ESS	Erase suspend status
FMI_FLAG_PECS	FPEC status

FMI_Bank

The FMI banks containing the flags to be read are listed in the following table:

FMI_Bank	Meaning
FMI_BANK_0	FMI bank 0
FMI_BANK_1	FMI bank 1

Example:

```
/*To get the FMI program status for bank 1*/
FlagStatus FMI_Flag;
FMI_Flag = GetFlagStatus(FMI_FLAG_PS, FMI_BANK_1);
```

4.2.10 FMI_GetReadWaitStateValue

Function Name	FMI_GetReadWaitStateValue
Function Prototype	`u16 FMI_GetReadWaitStateValue(void)`
Behavior Description	Gets the current read wait state value.
Input Parameter	None
Output Parameter	None
Return Parameter	The current read wait state value.
Required preconditions	...
Called functions	None

Example:

```
/*To get the current read wait state value*/
u16 FMI_ReadWaitState;
FMI_ReadWaitState = FMI_GetReadWaitStateValue();
```

4.2.11 FMI_GetWriteWaitStateValue

Function Name	FMI_GetWriteWaitStateValue
Function Prototype	`u16 FMI_GetWriteWaitStateValue(void)`
Behavior Description	Gets the current write wait state value.
Input Parameter	None
Output Parameter	None
Return Parameter	The current write wait state value.
Required preconditions	...
Called functions	None

Example:

```
/*To get the current write wait state value*/
u16 FMI_WriteWaitState;
FMI_WriteWaitState = FMI_GetWriteWaitStateValue();
```

4.2.12 FMI_SuspendEnable

Function Name	FMI_SuspendEnable
Function Prototype	`void FMI_SuspendEnable(vu32 FMI_Bank)`
Behavior Description	Enables the Suspend command.
Input Parameter	FMI_Bank: specifies the bank to be suspended. Refer to section "*FMI_Bank on page 33*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

FMI_Bank

To select the FMI bank, use one of the following values:

FMI_Bank	Meaning
FMI_BANK_0	FMI bank 0
FMI_BANK_1	FMI bank 1

Example:

```
/*To suspend the current command in the bank 1*/
FMI_SuspendEnanble(FMI_BANK_1);
```

4.2.13 FMI_ResumeEnable

Function Name	FMI_ResumeEnable
Function Prototype	`void FMI_ResumeEnable(vu32 FMI_Bank)`
Behavior Description	Resumes the suspended command.
Input Parameter	FMI_Bank: specifies the suspended bank. Refer to section "*FMI_Bank on page 33*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

FMI_Bank

To select the FMI bank, use one of the following values:

FMI_Bank	Meaning
FMI_BANK_0	FMI bank 0
FMI_BANK_1	FMI bank 1

Example:

```
/*To resume the suspended command in the bank 1*/
FMI_ResumeEnanble(FMI_BANK_1);
```

4.2.14 FMI_ClearFlag

Function Name	FMI_ClearFlag
Function Prototype	`void FMI_ClearFlag(vu32 FMI_Bank)`
Behavior Description	Clears the FMI flags in the corresponding bank.
Input Parameter	FMI_Bank: specifies the corresponding bank. Refer to section "*FMI_Bank on page 33*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

FMI_Bank

The FMI banks that contain the flags to be cleared are listed in the following table:

FMI_Bank	Meaning
FMI_BANK_0	FMI bank 0
FMI_BANK_1	FMI bank 1

Example:

```
/*To clear the status register on bank 0*/
FMI_ClearFlag(FMI_BANK_0);
```

4.2.15 FMI_WriteProtectionCmd

Function Name	FMI_WriteProtectionCmd
Function Prototype	`void FMI_WrieProtectionCmd(vu32 FMI_Sector, FunctionalState FMI_NewState)`
Behavior Description	Enables or disables the write protection for a specified sector.
Input Parameter1	FMI_Sector: specifies the sector to be protected or unprotected. Refer to section "*FMI_Sector on page 37*" for more details on the allowed values of this parameter.
Input Parameter2	NewState: specifies the protection status. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

FMI_Sector

To select the FMI sectors, use one of the following values:

FMI_Sector	Meaning
FMI_B0S0	FMI bank 0 sector 0
FMI_B0S1	FMI bank 0 sector 1
FMI_B0S2	FMI bank 0 sector 2
FMI_B0S3	FMI bank 0 sector 3
FMI_B0S4	FMI bank 0 sector 4
FMI_B0S5	FMI bank 0 sector 5
FMI_B0S6	FMI bank 0 sector 6
FMI_B0S7	FMI bank 0 sector 7
FMI_B1S0	FMI bank 1sector 0
FMI_B1S1	FMI bank 1sector 1
FMI_B1S2	FMI bank 1sector 2
FMI_B1S3	FMI bank 1sector 3

Example:

```
/*To disable the write protection of the sector 3*/
FMI_WriteProtectionCmd(FMI_B0S3, DISABLE);
```

4.2.16 FMI_GetWriteProtectionStatus

Function Name	FMI_GetWriteProtectionStatus
Function Prototype	`FlagStatus FMI_GetWriteProtectionStatus(u32 FMI_Sector_Mask)`
Behavior Description	Gets the write protection status for the specified sector.
Input Parameter	FMI_Sector_Mask: The sector mask. Refer to section "*FMI_Sector on page 37*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The write protection status for the specified sector. It can be SET or RESET. - SET: the write protection is enabled. - RESET: the write protection is disabled.
Required preconditions	...
Called functions	None

FMI_Sector_Mask

To select the FMI sector Mask, use one of the following values:

FMI_Sector	Meaning
FMI_B0S0_Mask	FMI bank 0 sector 0
FMI_B0S1_Mask	FMI bank 0 sector 1
FMI_B0S2_Mask	FMI bank 0 sector 2
FMI_B0S3_Mask	FMI bank 0 sector 3
FMI_B0S4_Mask	FMI bank 0 sector 4
FMI_B0S5_Mask	FMI bank 0 sector 5
FMI_B0S6_Mask	FMI bank 0 sector 6
FMI_B0S7_Mask	FMI bank 0 sector 7
FMI_B1S0_Mask	FMI bank 1 sector 0
FMI_B1S1_Mask	FMI bank 1 sector 1
FMI_B1S2_Mask	FMI bank 1 sector 2
FMI_B1S3_Mask	FMI bank 1 sector 3

Example:

```
/*To get the write protection status of sector 2 of bank 0*/
FlagStatus FMI_WriteProtectionStatus;
FMI_WriteProtectionStatus = FMI_GetWriteProtectionStatus(FMI_B0S2_MASK);
```

4.2.17 FMI_WaitForLastOperation

Function Name	FMI_WaitForLastOperation
Function Prototype	`u8 FMI_WaitForLastOperation(vu32 FMI_Bank)`
Behavior Description	Waits for the last operation (Write halfword, Write OTP halfword, Erase sector and Erase bank) to be completed.
Input Parameter	FMI_Bank: specifies the corresponding bank. Refer to section "*FMI_Bank on page 39*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The timeout status. This parameter can be one of the following values: - FMI_TIME_OUT_ERROR: Timeout error occurred. - FMI_NO_TIME_OUT_ERROR: No timeout error.
Required preconditions	...
Called functions	None

FMI_Bank

To select the FMI banks, use one of the following values:

FMI_Bank	Meaning
FMI_BANK_0	FMI bank 0
FMI_BANK_1	FMI bank 1

Example:

```
/*To wait until the write operation completion*/
u8 FMI_Timeout_Status;
FMI_WriteProtectionCmd(FMI_B1S0, DISABLE);
FMI_WriteHalfWord(0x80000,0x1234);
FMI_Timeout_Status = FMI_WaitForLastOperation(FMI_BANK_1);
```

5 External Memory Interface (EMI)

The EMI provides an interface between the AHB system bus and external memory devices supporting up to four memory banks that you can configure independently. Each memory bank supports: SRAM, ROM and Flash EPROM.

You can configure each memory bank to use 8 or 16-bit data paths.

You can configure the EMI memory banks to support:

Non-burst read and write accesses.

Asynchronous page mode read accesses supported in 8-bit non-multiplexed EMI configuration.

The first section describes the data structure used in the EMI software library. The second one presents the software library functions.

Note: Some EMI-related functions are defined in the SCU module (see Section 6.2 on page 50).

5.1 EMI register structure

5.1.1 EMI register structure

The EMI register structure *EMI_Bank_TypeDef is* defined in the ***91x_map.h*** file as follows:

```
typedef struct
{
   vu32 EMI_ICR;
   vu32 EMI_RCR;
   vu32 EMI_WCR;
   vu32 EMI_OECR;
   vu32 EMI_WECR;
   vu32 EMI_BCR;
} EMI_Bank_TypeDef;
```

This structure defines the registers of one bank.

There are 4 banks in the EMI interface, so there are 24 registers without taking into account the SCU registers used to configure the EMI clock and GPIO mode for EMI bus.

The following table presents the EMI registers:

Register	Description
EMI_ICRx	Bankx Idle Cycle Control Register
EMI_RCRx	Bankx Read Wait State Control Register
EMI_WCRx	Bankx Write Wait State Control Register
EMI_OECRx	Bankx Output Enable Assertion Delay Control Register
EMI_WECRx	Bankx Write Enable Assertion Delay Control Register
EMI_BCRx	Bankx Control Register

The EMI interface is declared in the same file:

```
#ifndef EXT
  #define EXT extern
#endif /* EXT */

#define AHB_EMI_U          (0x74000000) /* EMI UnBuffered Space */
#define AHB_EMI_B          (0x64000000) /* EMI Buffered Space   */

#define AHB_EMIB3_OFST      (0x00000040)   /* Offset of EMI bank3 */
#define AHB_EMIB2_OFST      (0x00000020)   /* Offset of EMI bank2 */
#define AHB_EMIB1_OFST      (0x00000000)   /* Offset of EMI bank1 */
#define AHB_EMIB0_OFST      (0x000000E0)   /* Offset of EMI bank0 */
...
#ifndef Buffered
#define EMI_BASE           (AHB_EMI_U)
...
#else /* Buffered */

#define EMI_BASE           (AHB_EMI_B)
...
#endif /* Buffered */

/* Bank Base Address definition*/
#define EMI_Bank0_BASE  (EMI_BASE + AHB_EMIB0_OFST)
#define EMI_Bank1_BASE  (EMI_BASE + AHB_EMIB1_OFST)
#define EMI_Bank2_BASE  (EMI_BASE + AHB_EMIB2_OFST)
#define EMI_Bank3_BASE  (EMI_BASE + AHB_EMIB3_OFST)

...

#ifndef DEBUG
...

/* EMI peripheral declaration*/

#define EMI_Bank0         ((EMI_Bank_TypeDef *)EMI_Bank0_BASE)
#define EMI_Bank1         ((EMI_Bank_TypeDef *)EMI_Bank1_BASE)
#define EMI_Bank2         ((EMI_Bank_TypeDef *)EMI_Bank2_BASE)
#define EMI_Bank3         ((EMI_Bank_TypeDef *)EMI_Bank3_BASE)

#else   /* DEBUG */
...
#ifdef _EMI_Bank0
EXT EMI_Bank_TypeDef       *EMI_Bank0;
#endif /* _EMI_Bank0 */

#ifdef _EMI_Bank1
EXT EMI_Bank_TypeDef       *EMI_Bank1;
#endif /* _EMI_Bank1 */

#ifdef _EMI_Bank2
EXT EMI_Bank_TypeDef       *EMI_Bank2;
#endif /* _EMI_Bank2 */

#ifdef _EMI_Bank3
EXT EMI_Bank_TypeDef       *EMI_Bank3;
#endif /* _EMI_Bank3 */
```

When debug mode is used, the EMI pointers are initialized in ***91x_lib.c*** file:

```
#ifdef _EMI_Bank0
  EMI_Bank0= (EMI_Bank_TypeDef *)EMI_Bank0_BASE;
#endif /* _EMI_Bank0 */

#ifdef _EMI_Bank1
  EMI_Bank1=       (EMI_Bank_TypeDef *)EMI_Bank1_BASE;
#endif /* _EMI_Bank1 */

#ifdef _EMI_Bank2
  EMI_Bank2 =      (EMI_Bank_TypeDef *)EMI_Bank2_BASE;
#endif /* _EMI_Bank2 */

#ifdef _EMI_Bank3
 EMI_Bank3 =      (EMI_Bank_TypeDef *)EMI_Bank3_BASE;
 #endif /* _EMI_Bank3 */
```

_EMI, _EMI_Bankl0, _EMI_Bank1, _EMI_Bank2 and EMI_Bank3 must be defined, in ***91x_conf.h*** file, to access the peripheral registers as follows:

```
#define _EMI
#define _EMI_Bank0
#define _EMI_Bank1
#define _EMI_Bank2
#define __EMI_Bank3

...
```

5.2 Software library functions

The following table enumerates the different functions of the EMI library.

Function Name	Description
EMI_DeInit	Initializes the EMI peripheral registers to their default reset values.
EMI_Init	Initializes the EMI Bankx according to the specified parameters in the EMI_InitStruct.
EMI_StructInit	Fills each EMI_InitStruct member with its reset value.

5.2.1 EMI_DeInit

Function Name	EMI_DeInit
Function Prototype	`void EMI_DeInit()`
Behavior Description	Initializes the EMI peripheral registers to their default reset values.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	`SCU_AHBPeriphReset()`

Example:

```
...
EMI_DeInit(); /* set all EMI Peripheral registers to their reset value */
...
```

5.2.2 EMI_Init

Function Name	EMI_Init
Function Prototype	`void EMI_Init(EMI_Bank_TypeDef* EMI_Bankx, EMI_InitTypeDef* EMI_InitStruct)`
Behavior Description	Initializes the EMI_Bankx according to the specified parameters in the EMI_InitStruct.
Input Parameter1	EMI_Bankx: where x can be 0,1,2 or 3 to select the EMI Bank.
Input Parameter2	EMI_InitStruct: pointer to a EMI_InitTypeDef structure that contains the configuration information for the specified EMI_Bankx. Refer to section "*EMI_InitTypeDef on page 44*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

EMI_InitTypeDef

The EMI_InitTypeDef structure is defined in the ***91x_map.h*** file:

```
typedef struct
{
u32 EMI_Bank_IDCY;
u32 EMI_Bank_WSTRD;
u32 EMI_Bank_WSTWR;
u32 EMI_Bank_WSTROEN;
u32 EMI_Bank_WSTWEN;
u32 EMI_Bank_MemWidth ;
u32 EMI_Bank_WriteProtect ;
u32 EMI_PageModeRead_TransferLength;
u32 EMI_PageModeRead_Selection;
} EMI_InitTypeDef;
```

Example:

```
...

   EMI_InitTypeDef EMI_InitStruct;
   EMI_InitStruct.EMI_Bank_IDCY =0x2;
   EMI_InitStruct.EMI_Bank_WSTRD =0x1;
   EMI_InitStruct.EMI_Bank_WSTWR =0x4;
   EMI_InitStruct.EMI_Bank_WSTROEN=0x5;
   EMI_InitStruct.EMI_Bank_WSTWEN=0x07;
   EMI_InitStruct.EMI_Bank_MemWidth=EMI_Width_HalfWord;
   EMI_InitStruct.EMI_Bank_WriteProtection= EMI_Bank_WriteProtect;
   EMI_InitStruct.EMI_PageModeRead_TransferLength=EMI_4Data;
   EMI_InitStruct.EMI_PageModeRead_Selection=EMI_PageModeRead;
   EMI_Init (EMI_Bank0,&EMI_InitStruct);
...
```

EMI_Bank_IDCY

This member controls the number of bus turnaround cycles added between read and write accesses. It is coded with 4 bits (16 values: 0x01, 0x02, 0x03, ... 0xF)

EMI_Bank_WSTRD

For SRAM and ROM, this member controls the number of wait states for read accesses, and the external wait assertion timing for reads. For burst ROM, it controls the number of wait states for the first read access only. It is coded with 5 bits (32 values: 0x01, 0x02, 0x03, ... 0x1F)

EMI_Bank_WSTWR

For SRAM, this member controls the number of wait states for write accesses, and the external wait assertion timing for writes.

It does not apply to read-only devices such as ROM.

It is coded with 5 bits (32 values: 0x01, 0x02, 0x03, ... 0x1F)

EMI_Bank_WSTROEN

Output enable assertion delay from chip select assertion.

It is coded with 4 bits (16 values: 0x01, 0x02, 0x03, ...0xF)

EMI_Bank_WSTWEN

Write Enable Assertion Delay From Chip Select Assertion.

It is coded with 4 bits (16 values: 0x01, 0x02, 0x03, ...0xF)

EMI_PageModeRead_TransferLength

This member controls the page transfer length for page mode read.

It can be one of the following values:

EMI_PageModeRead_TransferLength	Value	Meaning
EMI_4Data	0x00000000	4 -transfer burst
EMI_8Data	0x00000400	8-8transfer burst

EMI_PageModeRead_Selection

Select or deselect page mode read.

This member can be one of the following values:

EMI_PageModeRead_Selection	Value	Meaning
EMI_NormalMode	0x00000000	Normal Mode
EMI_PageModeRead	0x00000100	Page Mode Read

EMI_Bank_MemWidth

This member controls the memory width.

This member can be one of the following values:

EMI_Bank_MemWidth	Value	Meaning
EMI_Width_Byte	0x00000000	8-bit width
EMI_Width_HalfWord	0x00000010	16-bit width

EMI_Bank_WriteProtect

This member controls the write protection feature.

This member can be one of the following values:

EMI_Bank_WriteProtect	Value	Meaning
EMI_Bank_NonWriteProtect	0x00000000	No write protection,
EMI_Bank_WriteProtection	0x00000008	Bank is write protected.

5.2.3 EMI_StructInit

Function Name	EMI_StructInit
Function Prototype	`void EMI_StructInit(EMI_InitTypeDef* EMI_InitStruct)`
Behavior Description	Fills each EMI_InitStruct member with its reset value.
Input Parameter	EMI_InitStruct: pointer to a EMI_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
...
EMI_InitTypeDef EMI_InitStruct;
EMI_StructInit(&EMI_InitStruct);

...
```

6 System Control Unit (SCU)

The System Control Unit (SCU) provides the control logic for the STR91xFA power, reset and clocks, also it controls a large number of miscellaneous features.

6.1 Register structure

6.1.1 SCU register structure

The SCU register structure *SCU_TypeDef* is defined in the *91x_map.h* file as follows:

```
typedef struct
{
  vu32 CLKCNTR;      /* Clock Control Register                        */
  vu32 PLLCONF;      /* PLL Configuration Register                    */
  vu32 SYSSTATUS;    /* System Status Register                        */
  vu32 PWRMNG;       /* Power Management Register                     */
  vu32 ITCMSK;       /* Interrupt Mask Register                       */
  vu32 PCGRO;        /* Peripheral Clock Gating Register 0            */
  vu32 PCGR1;        /* Peripheral Clock Gating Register 1            */
  vu32 PRR0;         /* Peripheral Software Reset Register 0          */
  vu32 PRR1;         /* Peripheral Software Reset Register 1          */
  vu32 MGR0;         /* Mask Gating Register 0                        */
  vu32 MGR1;         /* Mask Gating Register 1                        */
  vu32 PECGR0;       /* Peripheral Emulation Clock Gating Register 0 */
  vu32 PECGR1;       /* Peripheral Emulation Clock Gating Register 1 */
  vu32 SCR0;         /* System Configuration Register 0               */
  vu32 SCR1;         /* System Configuration Register 1               */
  vu32 SCR2;         /* System Configuration Register 2               */
  u32  EMPTY1;
  vu32 GPIOOUT[8];   /* GPIO Output Registers                         */
  vu32 GPIOIN[8];    /* GPIO Input Registers                          */
  vu32 GPIOTYPE[10]; /* GPIO Type Registers                           */
  vu32 GPIOEMI;      /* GPIO EMI Selector Register                    */
  vu32 WKUPSEL;      /* Wake-Up Selection Register                    */
  u32  EMPTY2[2];
  vu32 GPIOANA;      /* GPIO Analag mode                              */
} SCU_TypeDef;
```

The following table presents the SCU registers:

Register	Description
CLKCNTR	Clock Control Register
PLLCONF	PLL Configuration Register
SYSSTATUS	System Status Register
PWRMNG	Power Management Register
ITMSK	Interrupt Mask Register
PCGRn	Peripheral Clock Gating Register (n =[0:1])
PRRn	Peripheral Reset Register (n=[0:1])
MGRn	Idle mode Mask Gating Register (n=[0:1])
PECGRn	Peripheral Emulation Clock Gating (n=[0:1])
GPIOOUTn	GPIO Output Register (n=[0:7])
GPIOINn	GPIO Input Register (n=[0:7])
GPIOTYPEn	GPIO Type Register (n=[0:9])
GPIOEMI	GPIO External Memory Interface selector Register
SCRn	System Configuration Register (n=[0:2])
WKUPSEL	WakeUp Select Register
GPIOANA	GPIO Analog mode Register

The SCU is declared in the file below

```
#ifndef EXT
 #Define EXT extern
#endif
...
#define AHB_APB_BRDG1_U    (0x5C000000) /* AHB/APB Bridge 1 UnBuffered Space */
#define AHB_APB_BRDG1_B    (0x4C000000) /* AHB/APB Bridge 1 Buffered Space   */
...
#define APB_SCU_OFST       (0x00002000) /* Offset of SCU */
...
#ifndef Buffered
#define AHBAPB1_BASE           (AHB_APB_BRDG1_U)
...
#else /* Buffered */
...
#define AHBAPB1_BASE           (AHB_APB_BRDG1_B)

/* SCU Base Address definition*/
#define SCU_BASE          (AHBAPB1_BASE + APB_SCU_OFST)


...
/* SCU peripheral declaration*/

#ifndef DEBUG
...
#define SCU         ((SCU_TypeDef *) SCU_BASE)
...
#else
```

```
...
#ifdef _SCU
EXT SCU_TypeDef           *SCU;
#endif /* _SCU */
...

#endif
```

When debug mode is used, SCU pointer is initialized in ***91x_lib.c*** file:

```
#ifdef _SCU
  SCU = (SCU_TypeDef *)SCU_BASE
#endif /* _SCU */
```

_SCU must be defined, in ***91x_conf.h*** file, to access the peripheral registers as follows:

```
#define _SCU

         ...
```

6.2 Software library functions

The following table enumerates the different functions of the SCU library.

Function Name	Description
SCU_MCLKSourceConfig	Selects the Master clock source: OSC, PLL, or RTC
SCU_PLLFactorsConfig	Configures the PLL factors M,N and P
SCU_PLLCmd	Enables or disables the PLL
SCU_RCLKDivisorConfig	Configures the RCLK divisor: 1,2,4,8,16 or 1024
SCU_HCLKDivisorConfig	Configures the HCLK divisor: 1,2 or 4
SCU_PCLKDivisorConfig	Configures the PCLK divisor: 1,2,4 or 8
SCU_FMICLKDivisorConfig	Configures the FMI clock divisor: 1 or 2
SCU_EMIBCLKDivisorConfig	Configures the EMI bus clock divisor : 1 or 2
SCU_BRCLKDivisorConfig	Selects the Baud Rate clock divisor: 1 or 2
SCU_TIMCLKSourceConfig	Configures the TIMx clock source : External clock or MCLK divided by prescaler
SCU_TIMPresConfig	Configures the TIMx clock prescaler
SCU_USBCLKConfig	Selects the USBClock source: external 48MHz, MCLK or MCLK/2
SCU_PHYCLKConfig	Enables or disables the output PHY clock
SCU_APBPeriphClockConfig	Enables or disables the APB peripheral clocks
SCU_AHBPeriphClockConfig	Enables or disables the AHB peripheral clocks
SCU_APBPeriphIdleConfig	Enables or disables the APB peripheral clocks during Idle mode
SCU_AHBPeriphIdleConfig	Enables or disables the AHB peripherals clocks during Idle mode
SCU_APBPeriphDebugConfig	Enables or disables the APB peripherals clocks during debug mode
SCU_AHBPeriphDebugConfig	Enables or disables AHB peripheral clocks during debug mode
SCU_APBPeriphReset	Enables or disables Reset state for APB peripherals
SCU_AHBPeriphReset	Enables or disables Reset state for AHB peripherals
SCU_EMIModeConfig	Configures EMI mode : Multiplexed or Demultiplexed mode
SCU_EMIALEConfig	Configures EMI ALE signal length and polarity
SCU_GetPLLFreqValue	Gets the current PLL frequency value (unit = KHz)
SCU_GetMCLKFreqValue	Gets the current MCLK frequency value (unit = KHz)
SCU_GetHCLKFreqValue	Gets the current HCLK frequency value (unit = KHz)
SCU_GetPCLKFreqValue	Gets the current PCLK frequency value (unit = KHz)
SCU_GetRCLKFreqValue	Gets the current RCLK frequency value (unit = KHz)
SCU_WakeUpLineConfig	Configures an external interrupt as wake-up line
SCU_SpecIntRunModeConfig	Enables or disables the Special interrupt Run mode
SCU_EnterIdleMode	Puts the MCU in Idle mode
SCU_EnterSleepMode	Puts the MCU in Sleep mode
SCU_UARTIrDASelect	Selects UARTx operation : UART or IrDA

Function Name	Description
SCU_PFQBCCmd	Enables or disables PFQBC
SCU_ITConfig	Enables or disables SCU interrupts
SCU_GetFlagStatus	Gets a flag status
SCU_ClearFlag	Clears a status flag

Note: The SCU_GPIOOUT, SCU_GPIOIN, SCU_GPIOTYPE, SCU_GPIOEMI and SCU_GPIOANA registers are not supported by the SCU library functions, they are supported by the GPIO driver functions.

6.2.1 SCU_MCLKSourceConfig

Function Name	SCU_MCLKSourceConfig
Function Prototype	`ErrorStatus SCU_MCLKSourceConfig(u32 MCLK_Source)`
Behavior Description	Selects the MCLK clock source: OSC, RTC, or PLL
Input Parameter	MCLK_Source Refer to *MCLK_Source on page 51* section for details about allowed values.
Output Parameter	None
Return Parameter	ErrorStatus: ERROR or SUCCESS – Function returns ERROR when selecting the PLL clock as MCLK source while PLL is either disabled or not locked.
Required preconditions	When selecting the PLL as MCLK clock source, make sure that the PLL is enabled and locked, you can do this using the *SCU_PLLCmd(ENABLE)* function.
Called functions	None

MCLK_Source

MCLK_Source	Meaning
SCU_MCLK_PLL	MCLK source = PLL clock
SCU_MCLK_RTC	MCLK source = RTC clock
SCU_MCLK_OSC	MCLK source = Oscillator clock

6.2.2 SCU_PLLFactorsConfig

Function Name	SCU_PLLFactorsConfig
Function Prototype	`ErrorStatus SCU_PLLFactorsConfig(u8 PLLN, u8 PLLM, u8 PLLP)`
Behavior Description	Configures the PLL factors
Input Parameter1	PLLN: PLL Feedback divider
Input Parameter2	PLLM: PLL Pre-divider
Input Parameter3	PLLP: PLL Post-divider
Output Parameter	None
Return Parameter	ErrorStatus: ERROR or SUCCESS – Function returns ERROR when trying to set new PLL factors while PLL selected as MCLK source clock
Required preconditions	PLL must not be selected as MCLK clock
Called functions	SCU_PLLCmd(DISABLE) : Disables the PLL

Note: This function disables the PLL before programming the PLL factors. To enable the PLL use the SCU_PLLCmd(ENABLE) function after programming the PLL factors.

6.2.3 SCU_PLLCmd

Function Name	SCU_PLLCmd
Function Prototype	`ErrorStatus SCU_PLLConfig(FunctionnalState NewState)`
Behavior Description	Enables or disables the PLL
Input Parameter	NewState : ENABLE or DISABLE
Output Parameter	None
Return Parameter	ErrorStatus: ERROR or SUCCESS The function returns ERROR when: – Disabling the PLL while PLL is selected as MCLK source – Enabling the PLL while PLL already enabled & locked
Required preconditions	– When enabling the PLL, you must have already configured the PLL factors using the *SCU_PLLFactorsConfig* function. – When disabling the PLL, make sure that the PLL is not selected as MCLK.
Called functions	None

Note: After enabling the PLL, the PLL lock bit is polled to ensure that the PLL is locked before exiting the function.

Before disabling the PLL a “security” delay is inserted, this to guarantee that the system clock has already switched to OSC or RTC before disabling the PLL.

6.2.4 SCU_RCLKDivisorConfig

Function Name	SCU_RCLKDivisorConfig
Function Prototype	`void SCU_RCLKDivisorConfig(u32 RCLK_Divisor)`
Behavior Description	Selects the RCLK divisor 1, 2, 4, 8, 16 or 1024
Input Parameter	RCLK_Divisor Refer to *RCLK_Divisor on page 53* section for details about allowed values.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

RCLK_Divisor

RCLK_Divisor	Meaning
SCU_RCLK_Div1	RCLK Divisor = 1
SCU_RCLK_Div2	RCLK Divisor = 2
SCU_RCLK_Div4	RCLK Divisor = 4
SCU_RCLK_Div8	RCLK Divisor = 8
SCU_RCLK_Div16	RCLK Divisor = 16
SCU_RCLK_Div1024	RCLK Divisor = 1024

6.2.5 SCU_HCLKDivisorConfig

Function Name	SCU_HCLKDivisorConfig
Function Prototype	`void SCU_HCLKDivisorConfig(u32 HCLK_Divisor)`
Behavior Description	Selects the HCLK divisor: 1,2 or 4.
Input Parameter	HCLK_Divisor Refer to *HCLK_Divisor on page 53* section for details about allowed values.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

HCLK_Divisor

HCLK_Divisor	Meaning
SCU_HCLK_Div1	HCLK Divisor = 1
SCU_HCLK_Div2	HCLK Divisor = 2
SCU_HCLK_Div4	HCLK Divisor = 4

6.2.6 SCU_PCLKDivisorConfig

Function Name	SCU_PCLKDivisorConfig
Function Prototype	`void SCU_PCLKDivisorConfig(u32 PCLK_Divisor)`
Behavior Description	Selects the PCLK divisor: 1,2,4 or 8
Input Parameter	PCLK_Divisor Refer to *PCLK_Divisor on page 54* section for details about allowed values.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

PCLK_Divisor

PCLK_Divisor	Meaning
SCU_PCLK_Div1	PCLK Divisor = 1
SCU_PCLK_Div2	PCLK Divisor = 2
SCU_PCLK_Div4	PCLK Divisor = 4
SCU_PCLK_Div8	PCLK Divisor = 8

6.2.7 SCU_FMICLKDivisorConfig

Function Name	SCU_FMICLKDivisorConfig
Function Prototype	`void SCU_FMICLKDivisorConfig(u32 FMICLK_Divisor)`
Behavior Description	Selects the FMI clock Divisor: 1 or 2
Input Parameter	FMICLK_Divisor Refer to *FMICLK_Divisor on page 54* section for details about allowed values.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

FMICLK_Divisor

FMICLK_Divisor	Meaning
SCU_FMICLK_Div1	FMICLK Divisor = 1
SCU_FMICLK_Div2	FMICLK Divisor = 2

6.2.8 SCU_EMIBCLKDivisorConfig

Function Name	SCU_EMIBCLKDivisorConfig
Function Prototype	`void SCU_EMIBCLKDivisorConfig(u32 EMIBCLK_Divisor)`
Behavior Description	Selects the EMI Bus clock Divisor: 1 or 2
Input Parameter	EMIBCLK_Divisor Refer to *EMIBCLK_Divisor on page 55* section for details about allowed values.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

EMIBCLK_Divisor

EMIBCLK_Divisor	Meaning
SCU_EMIBCLK_Div1	EMIBCLK Divisor = 1
SCU_EMIBCLK_Div2	EMIBCLK Divisor = 2

6.2.9 SCU_BRCLKDivisorConfig

Function Name	SCU_BRCLKDivisorConfig
Function Prototype	`void SCU_BRCLKDivisorConfig(u32 BRCLK_Divisor)`
Behavior Description	Selects the Baud rate clock Divisor : 1 or 2
Input Parameter	BRCLK_Divisor Refer to *BRCLK_Divisor on page 55* section for details about allowed values.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

BRCLK_Divisor

BRCLK_Divisor	Meaning
SCU_BRCLK_Div1	BRCLK Divisor = 1
SCU_BRCLK_Div2	BRCLK Divisor = 2

6.2.10 SCU_TIMCLKSourceConfig

Function Name	SCU_TIMCLKSourceConfig
Function Prototype	`void SCU_TIMCLKSourceConfig(u8 TIMx, u32 TIMCLK_Source)`
Behavior Description	Selects the TIM[0:3] clock source: External or MCLK/Prescaler
Input Parameter1	TIMx Refer to *TIMx on page 56* section for details about allowed values.
Input Parameter2	TIMCLK_Source: External or Internal Refer to *TIMCLK_Source on page 56* section for details about allowed values.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

TIMCLK_Source

ClockSource	Meaning
SCU_TIMCLK_EXT	TIMx clock = TIMx External
SCU_TIMCLK_INT	TIMx clock = Internal clock

TIMx

ClockSource	Meaning
SCU_TIM01	TIMER 0 & 1
SCU_TIM23	TIMER 2 & 3

6.2.11 SCU_TIMPresConfig

Function Name	SCU_TIMPresConfig
Function Prototype	`void SCU_TIMPresConfig(u8 TIMx, u16 Prescaler)`
Behavior Description	Configures the TIM[0:3] prescaler
Input Parameter1	TIMx Refer to *TIMx on page 56* section for details about allowed values.
Input Parameter2	Prescaler value (16bits value)
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

6.2.12 SCU_USBCLKConfig

Function Name	SCU_USBCLKConfig
Function Prototype	`void SCU_USBCLKConfig(u32 USBCLK_Source)`
Behavior Description	Selects the USB clock source
Input Parameter	USBCLK_Source Refer to *USBCLK_Source on page 57* section for details about allowed values.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

USBCLK_Source

ClockSource	Meaning
SCU_USBCLK_MCLK	USB clock = MCLK
SCU_USBCLK_MCLK2	USB clock = MCLK/2
SCU_USBCLK_EXT	USB clock = External 48MHz

6.2.13 SCU_PHYCLKConfig

Function Name	SCU_PHYCLKConfig
Function Prototype	`void SCU_PHYCLKConfig(FunctionnalState NewState)`
Behavior Description	Enables/disables the PHYCLK output
Input Parameter	NewState: ENABLE or DISABLE
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

6.2.14 SCU_APBPeriphClockConfig

Function Name	SCU_APBPeriphClockConfig
Function Prototype	`void SCU_APBPeriphClockConfig(u32 APBPeriph, FunctionalState NewState)`
Behavior Description	Disables/ enables a clock for a peripheral or combination of peripherals (uses logical OR) on APB bus
Input Parameter1	APBPeriph: __PPP, example: __RTC , __GPIO0, ... Refer to *APBPeriph on page 58* section for details about allowed values.
Input Parameter2	NewState: ENABLE or DISABLE
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

APBPeriph

__TIM01, __TIM23, __MC, __UART0, __UART1, __UART2, __I2C0, __I2C1, __SSP0, __SSP1, __CAN, __ ADC, __WDG, __WIU, __GPIO0, __GPIO1, __GPIO2, __GPIO3, __GPIO4, __GPIO5, __GPIO6, __GPIO7, __GPIO8, __GPIO9, __RTC.

6.2.15 SCU_AHBPeriphClockConfig

Function Name	SCU_AHBPeriphClockConfig
Function Prototype	`void SCU_AHBPeriphClockConfig(u32 AHBPeriph, FunctionalState NewState)`
Behavior Description	Disables / enables the clock for a peripheral or combination of peripherals (use logical OR) on AHB bus
Input Parameter1	AHBPeriph: __PPP, example: __DMA, __USB, ... Refer to *AHBPeriph on page 59* section for details about allowed values.
Input Parameter2	NewState: ENABLE or DISABLE
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

AHBPeriph

__FMI, __PFQBC, __SRAM, __SRAM_ARBITER, __VIC, __EMI, __EXT_MEM_CLK, __DMA,

__USB, __USB48M , __ENET.

6.2.16 SCU_APBPeriphDebugConfig

Function Name	SCU_APBPeriphDebugConfig
Function Prototype	`void SCU_APBPeriphDebugConfig(u32 APBPeriph, FunctionalState NewState)`
Behavior Description	Enables/disables the clock for a peripheral during CPU Debug mode
Input Parameter1	APBPeriph Refer to *APBPeriph on page 58* section for details about allowed values.
Input Parameter2	NewState: ENABLE or DISABLE
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

6.2.17 SCU_AHBPeriphDebugConfig

Function Name	SCU_AHBPeriphDebugConfig
Function Prototype	`void SCU_AHBPeriphDebugConfig(u32 AHBPeriph, FunctionalState NewState)`
Behavior Description	Enables/disables the clock for a peripheral during CPU Debug mode

Input Parameter1	AHBPeriph Refer to *AHBPeriph on page 59* section for details about allowed values.
Input Parameter2	NewState: ENABLE or DISABLE
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

6.2.18 SCU_APBPeriphIdleConfig

Function Name	SCU_APBPeriphIdleConfig
Function Prototype	`void SCU_APBPeriphIdleConfig(u32 APBPeriph, FunctionalState NewState)`
Behavior Description	Enables/disables the clock for a peripheral during during Idle mode
Input Parameter1	APBPeriph Refer to *APBPeriph on page 58* section for details about allowed values.
Input Parameter2	NewState: ENABLE or DISABLE
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

6.2.19 SCU_AHBPeriphIdleConfig

Function Name	SCU_AHBPeriphIdleConfig
Function Prototype	`void SCU_AHBPeripIdleConfig(u32 AHBPeriph, FunctionalState NewState)`
Behavior Description	Enables/ disables the clock for a peripheral during Idle mode
Input Parameter1	AHBPeriph Refer to *AHBPeriph on page 59* section for details about allowed values.
Input Parameter2	NewState: ENABLE or DISABLE
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

6.2.20 SCU_APBPeriphReset

Function Name	SCU_APBPeriphReset
Function Prototype	`void SCU_APBPeriphReset(u32 APBPeriph, FunctionalState NewState)`
Behavior Description	Forces a peripheral into RESET
Input Parameter1	APBPeriph Refer to *APBPeriph on page 58* section for details about allowed values.
Input Parameter2	NewState: ENABLE or DISABLE
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

6.2.21 SCU_AHBPeriphReset

Function Name	SCU_AHBPeriphReset
Function Prototype	`void SCU_AHBPeriphReset(u32 AHBResetPeriph, FunctionalState NewState)`
Behavior Description	Forces a peripheral into RESET
Input Parameter1	AHBResetPeriph Refer to *AHBResetPeriph on page 62* section for details about allowed values.
Input Parameter2	NewState: ENABLE or DISABLE
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

AHBResetPeriph

__FMI, __PFQBC, __SRAM, __SRAM_ARBITER, __VIC, __DMA,__USB, __ENET, __PFQBC_AHB.

6.2.22 SCU_EMIModeConfig

Function Name	SCU_EMIModeConfig
Function Prototype	`void SCU_EMIModeConfig(u32 SCU_EMIMODE)`
Behavior Description	Configures the EMI mode: multiplexed or demultiplexed
Input Parameter	SCU_EMIMODE Refer to *SCU_EMIMODE on page 63* section for details about allowed values.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

SCU_EMIMODE

SCU_EMIMODE	Meaning
SCU_EMI_MUX	EMI mode = Multiplexed
SCU_EMI_DEMUX	EMI mode = Demultiplexed

6.2.23 SCU_EMIALEConfig

Function Name	SCU_EMIALEConfig
Function Prototype	`void SCU_EMIALEConfig(u32 SCU_EMIALE_LEN, u32 SCU_EMIALE_POL)`
Behavior Description	Configures EMI ALE signal length and polarity
Input Parameter1	SCU_EMIALE_LEN Refer to *SCU_EMIALE_LEN on page 64* section for details about allowed values.
Input Parameter2	SCU_EMIALE_POL Refer to *SCU_EMIALE_POL on page 64* section for details about allowed values.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

SCU_EMIALE_LEN

mode	Meaning
SCU_EMIALE_LEN1	EMI ALE Length = 1 clock cycle
SCU_EMIALE_LEN2	EMI ALE length = 2 clock cycles

SCU_EMIALE_POL

mode	Meaning
SCU_EMIALE_POLLow	EMI ALE Polarity = Low
SCU_EMIALE_POLHigh	EMI ALE Polarity = High

6.2.24 SCU_GetPLLFreqValue

Function Name	SCU_GetPLLFreqValue
Function Prototype	`u32 SCU_GetPLLFreqValue(void)`
Behavior Description	Gets the current generated PLL frequency
Input Parameter	None
Output Parameter	None
Return Parameter	PLL frequency value (unit = KHz)
Required preconditions	None
Called functions	None

6.2.25 SCU_GetMCLKFreqValue

Function Name	SCU_GetMCLKFreqValue
Function Prototype	`u32 SCU_GetMCLKFreqValue(void)`
Behavior Description	Gets current MCLK frequency value
Input Parameter	None
Output Parameter	None
Return Parameter	MCLK frequency value (unit = KHz)
Required preconditions	None
Called functions	None

6.2.26 SCU_GetHCLKFreqValue

Function Name	SCU_GetHCLKFreqValue
Function Prototype	`u32 SCU_GetHCLKFreqValue(void)`
Behavior Description	Gets current HCLK frequency value
Input Parameter	None
Output Parameter	None
Return Parameter	HCLK frequency value (unit = KHz)
Required preconditions	None
Called functions	None

6.2.27 SCU_GetPCLKFreqValue

Function Name	SCU_GetPCLKFreqValue
Function Prototype	`u32 SCU_GetPCLKFreqValue(void)`
Behavior Description	Gets current PCLK frequency value
Input Parameter	None
Output Parameter	None
Return Parameter	PCLK frequency value (unit = KHz)
Required preconditions	None
Called functions	None

6.2.28 SCU_GetRCLKFreqValue

Function Name	SCU_GetRCLKFreqValue
Function Prototype	`u32 SCU_GetRCLKFreqValue(void)`
Behavior Description	Gets current RCLK frequency value
Input Parameter	None
Output Parameter	None
Return Parameter	RCLK frequency (unit = KHz)
Required preconditions	None
Called functions	None

6.2.29 SCU_WakeUpLineConfig

Function Name	SCU_WakeUpLineConfig
Function Prototype	`u32 SCU_WakeUpLineConfig(u8 EXTint)`
Behavior Description	Configures an external interrupt as wakeup line
Input Parameter	EXTint (value from 0 to 31)
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

6.2.30 SCU_EnterIdleMode

Function Name	SCU_EnterIdleMode
Function Prototype	`void SCU_EnterIdleMode(void)`
Behavior Description	Puts MCU in Idle mode
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

6.2.31 SCU_EnterSleepMode

Function Name	SCU_EnterSleepMode
Function Prototype	`void SCU_EnterSleepMode(void)`
Behavior Description	Puts MCU in Sleep mode
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

6.2.32 SCU_UARTIrDASelect

Function Name	SCU_UARTIrdaSelect
Function Prototype	`void SCU_UARTIrDASelect(u8 SCU_UARTx, u8 UART_IrDA_Mode)`
Behavior Description	Configures UARTx as UART or IrDA
Input Parameter1	SCU_UARTx: SCU_UART[0:3]

Input Parameter2	UART_IrDA Refer to *UART_IrDA on page 67* section for details about allowed values.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

UART_IrDA

mode	Meaning
SCU_UARTMode_IrDA	UARTMode = IrDA
SCU_UARTMode_UART	UARTMode = UART

SCU_UARTx

mode	Meaning
SCU_UART0	UART0
SCU_UART1	UART1
SCU_UART2	UART2

6.2.33 SCU_PFQBCCmd

Function Name	SCU_PFQBCCmd
Function Prototype	`void SCU_PFQBCCmd(FunctionalState NewState)`
Behavior Description	Enables or Disables the PFQBC
Input Parameter	NewState: ENABLE or DISABLE
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

6.2.34 SCU_ITConfig

Function Name	SCU_ITConfig
Function Prototype	`void SCU_ITConfig(u32 SCU_IT, FunctionalState NewState)`
Behavior Description	Enables or disables the specified SCU interrupts.
Input Parameter1	SCU_IT: specifies the SCU interrupts sources to be enabled or disabled. Refer to *SCU_IT on page 68* section for more details on the allowed values of this parameter.
Input Parameter2	NewState: new state of the specified SCU interrupts. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

SCU_IT

SCU_IT	Meaning
SCU_IT_LVD_RST	LVD Reset interrupt
SCU_IT_SRAM_ERROR	SRAM error interrupt
SCU_IT_ACK_PFQBC	ACK_PFQBC interrupt
SCU_IT_LOCK_LOST	PLL LOCK Lost interrupt
SCU_IT_LOCK	PLL lock interrupt

6.2.35 SCU_GetFlagStatus

Function Name	SCU_GetFlagStatus
Function Prototype	`FlagStatus SCU_GetFlagStatus(u32 SCU_FLAG)`
Behavior Description	Checks whether the specified SCU flag is set or not.
Input Parameter	SCU_FLAG: specifies the flag to check. Refer to *SCU_FLAG on page 69* section for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of SCU_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

SCU_FLAG

SCU_FLAG	Meaning
SCU_FLAG_SRAM_ERR	SRAM error flag
SCU_FLAG_ACK_PFQBC	ACK_PFQBC flag
SCU_FLAG_LVD_RESET	LVD RESET flag
SCU_FLAG_WDG_RST	Watchdog RESET flag
SCU_FLAG_LOCK_LOST	PLL Lock lost flag
SCU_FLAG_LOCK	PLL Lock flag

6.2.36 SCU_ClearFlag

Function Name	SCU_ClearFlag
Function Prototype	`void SCU_ClearFlag(u32 SCU_FLAG)`
Behavior Description	Clears an SCU flag
Input Parameter	SCU_FLAG: specifies the flag to clear. Refer to *SCU_FLAG on page 69* section for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

7 General Purpose I/O Ports (GPIO)

The GPIO driver may be used for several purposes, including pin configuration, reading a port pin and writing data into the port pin.

The first section describes the data structures used in the GPIO software library. The second one presents the software library functions.

7.1 GPIO register structure

The GPIO register structure *GPIO_TypeDef* is defined in the ***91x_map.h*** file as follows:

```
typedef struct
{

vu8 DR[1021]; /* Data Register */
vu32 DDR; /* Data Direction Register */
} GPIO_TypeDef;
```

The following table presents the GPIO registers:

Register	Description
GPIODATA	Data register
GPIODIR	Data direction register

The ten GPIO interfaces are declared in the same file:

```
#ifndef EXT
 #Define EXT extern
#endif
...
#define AHB_APB_BRDG0_U    (0x58000000) /* AHB/APB Bridge 0 UnBuffered Space */
#define AHB_APB_BRDG0_B    (0x48000000) /* AHB/APB Bridge 0 Buffered Space   */
...
#define APB_GPIO0_OFST       (0x00006000)   /* Offset of GPIO0 */
#define APB_GPIO1_OFST       (0x00007000)   /* Offset of GPIO1 */
#define APB_GPIO2_OFST       (0x00008000)   /* Offset of GPIO2 */
#define APB_GPIO3_OFST       (0x00009000)   /* Offset of GPIO3 */
#define APB_GPIO4_OFST       (0x0000A000)   /* Offset of GPIO4 */
#define APB_GPIO5_OFST       (0x0000B000)   /* Offset of GPIO5 */
#define APB_GPIO6_OFST       (0x0000C000)   /* Offset of GPIO6 */
#define APB_GPIO7_OFST       (0x0000D000)   /* Offset of GPIO7 */
#define APB_GPIO8_OFST       (0x0000E000)   /* Offset of GPIO8 */
#define APB_GPIO9_OFST       (0x0000F000)   /* Offset of GPIO9 */

...
#ifndef Buffered
#define AHBAPB0_BASE           (AHB_APB_BRDG0_U)
...
#else /* Buffered */
...
#define AHBAPB0_BASE           (AHB_APB_BRDG0_B)
```

```
/* GPIO Base Address definition*/

#define GPIO0_BASE          (AHBAPB0_BASE + APB_GPIO0_OFST)
#define GPIO1_BASE          (AHBAPB0_BASE + APB_GPIO1_OFST)
#define GPIO2_BASE          (AHBAPB0_BASE + APB_GPIO2_OFST)
#define GPIO3_BASE          (AHBAPB0_BASE + APB_GPIO3_OFST)
#define GPIO4_BASE          (AHBAPB0_BASE + APB_GPIO4_OFST)
#define GPIO5_BASE          (AHBAPB0_BASE + APB_GPIO5_OFST)
#define GPIO6_BASE          (AHBAPB0_BASE + APB_GPIO6_OFST)
#define GPIO7_BASE          (AHBAPB0_BASE + APB_GPIO7_OFST)
#define GPIO8_BASE          (AHBAPB0_BASE + APB_GPIO8_OFST)
#define GPIO9_BASE          (AHBAPB0_BASE + APB_GPIO9_OFST)
...
/* GPIO peripheral declaration*/

#ifndef DEBUG
...
#define GPIO0    ((GPIO_TypeDef *) GPIO0_BASE)
#define GPIO1    ((GPIO_TypeDef *) GPIO1_BASE)
#define GPIO2    ((GPIO_TypeDef *) GPIO2_BASE)
#define GPIO3    ((GPIO_TypeDef *) GPIO3_BASE)
...
#define GPIO2    ((GPIO_TypeDef *) GPIO8_BASE)
#define GPIO3    ((GPIO_TypeDef *) GPIO9_BASE)

#else
...

#ifdef _GPIO0
EXT GPIO_TypeDef           *GPIO0;
#endif /* _GPIO0 */

#ifdef _GPIO1
EXT GPIO_TypeDef           *GPIO1;
#endif /* _GPIO1 */

#ifdef _GPIO2
EXT GPIO_TypeDef           *GPIO2;
#endif /* _GPIO2 */

#ifdef _GPIO3
EXT GPIO_TypeDef           *GPIO3;
#endif /* _GPIO3 */

...


#ifdef _GPIO8
EXT GPIO_TypeDef           *GPIO8;
#endif /* _GPIO8 */

#ifdef _GPIO9
EXT GPIO_TypeDef           *GPIO9;
#endif /* _GPIO9 */
```

When debug mode is used, the GPIO pointers are initialized in the ***91x_lib.c*** file:

```
#ifdef _GPIO0
GPIO0 = (GPIO_TypeDef *)GPIO0_BASE;
#endif /*_GPIO0 */

#ifdef _GPIO1
GPIO0 = (GPIO_TypeDef *)GPIO1_BASE;
#endif /*_GPIO1 */

#ifdef _GPIO2
GPIO0 = (GPIO_TypeDef *)GPIO2_BASE;
#endif /*_GPIO2 */

#ifdef _GPIO3
GPIO1 = (GPIO_TypeDef *)GPIO3_BASE;
#endif /*_GPIO3 */

...

#ifdef _GPIO8
GPIO1 = (GPIO_TypeDef *)GPIO8_BASE;
#endif /*_GPIO8 */

#ifdef _GPIO9
GPIO1 = (GPIO_TypeDef *)GPIO9_BASE;
#endif /*_GPIO9 */
```

_GPIO, _GPIO0,_GPIO1,_GPIO2 ...._GPIO9 must be defined, in ***91x_conf.h*** file, to access the peripheral registers as follows:

```
#define _GPIO
#define _GPIO0
#define _GPIO1
#define _GPIO2
...
#define _GPIO8
#define _GPIO9

...
```

7.2 Software library functions

The following table enumerates the different functions of the GPIO library.

Function Name	Description
GPIO_DeInit	Deinitializes the GPIOx peripheral registers to their default reset values.
GPIO_Init	Initializes the GPIOx peripheral according to the specified parameters in the GPIO_InitStruct.
GPIO_StructInit	Fills each GPIO_InitStruct member with its reset value.
GPIO_Read	Reads the specified GPIO data port
GPIO_ReadBit	Reads the specified port pin
GPIO_WriteBit	Sets or clears the selected data port bit.
GPIO_Write	Write data to the specified GPIO data port
GPIO_ANAPinConfig	Enables or disables pins from GPIO 4 in Analog mode.
GPIO_EMIConfig	Enables or disables GPIO 8 and 9 in EMI mode.

7.2.1 GPIO_DeInit

Function Name	GPIO_DeInit
Function Prototype	`void GPIO_DeInit(GPIO_TypeDef* GPIOx)`
Behavior Description	Deinitializes the GPIOx peripheral registers to their default reset values.
Input Parameter	GPIOx: where x can be 0,1,..,9 to select the GPIO peripheral.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	SCU_APBPeriphReset()

Example:

```
/*Deinitializes the GPIO0 peripheral registers to their default reset values*/
GPIO_DeInit(GPIO0);
```

7.2.2 GPIO_Init

Function Name	GPIO_Init
Function Prototype	`void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)`
Behavior Description	Initializes the GPIOx peripheral according to the specified parameters in the GPIO_InitStruct .
Input Parameter1	GPIOx: where x can be 0,1, .., 9 to select the GPIO peripheral.
Input Parameter2	GPIO_InitStruct: pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral. Refer to section "*GPIO_InitTypeDef on page 74*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

GPIO_InitTypeDef

The GPIO_InitTypeDef structure is defined in the ***91x_GPIO.h*** file:

```
typedef struct
{
vu8 GPIO_Pin;
vu8 GPIO_Direction;
vu8 GPIO_Type;
vu8 GPIO_IPConnected;
vu16 GPIO_Alternate;
} GPIO_InitTypeDef;
```

GPIO_Pin

Specifies GPIO pins to configure, (allows multiple pin configuration with the | operator).

This member can be one of the following values:

GPIO_Pin	Meaning
GPIO_Pin_None	No pin selected
GPIO_Pin_0	Pin 0 Selected
GPIO_Pin_1	Pin 1 Selected
GPIO_Pin_2	Pin 2 Selected
GPIO_Pin_3	Pin 3 Selected
GPIO_Pin_4	Pin 4 Selected
GPIO_Pin_5	Pin 5 Selected
GPIO_Pin_6	Pin 6 Selected
GPIO_Pin_7	Pin 7 Selected
GPIO_Pin_All	All pins Selected

GPIO_Direction

Specifies GPIO pin direction.

This member can be one of the following values:

GPIO_Direction	Meaning
GPIO_PinOutput	Configures corresponding pin to be an output.
GPIO_PinInput	Configures corresponding pin to be an input

GPIO_Type

Specifies the pin output type.

This member can be one of the following values:

GPIO_Type	Meaning
GPIO_Type_PushPull	Configures the GPIO in push pull type.
GPIO_Type_OpenCollector	Configures the GPIO in open collector type.

GPIO_IPConnected

Specifies the IP function to receive the input from a port pin. Only GPIO pins on P0 thru P7 use this function.

This member can be one of the following values:

GPIO_IPConnected	Meaning
GPIO_IPConnected_Enable	IP connected to the input
GPIO_IPConnected_Disàable	IP unconnected to the input.

GPIO_Alternate

Specifies the IP function to be assigned to port pin. Only GPIO pins on P0 thru P7 use this function.

This member can be one of the following values:

GPIO_Out	Meaning
GPIO_InputAlt1	Configures the GPIO pin in input
GPIO_OutputAlt1	Configures the GPIO pin in output alternate function 1
GPIO_OutputAlt2	Configures the GPIO pin in output alternate function 2
GPIO_OutputAlt3	Configures the GPIO pin in output alternate function 3

Example:

```
/* Configure all the GPIO0 in Input Push pull mode*/
GPIO_InitTypeDef GPIO_InitStruct;
GPIO_InitStruct.GPIO_Pin = GPIO_Pin_All;
GPIO_InitStruct.GPIO_Direction = GPIO_PinInput;
GPIO_InitStruct.GPIO_Type = GPIO_Type_PushPull;
GPIO_InitStruct.GPIO_IPConnected = GPIO_IPConnected_Disable;
GPIO_InitStruct.GPIO_Alternate = GPIO_InputAlt1;
GPIO_Init (GPIO0, &GPIO_InitStruct);
```

7.2.3 GPIO_StructInit

Function Name	GPIO_StructInit
Function Prototype	`void GPIO_StructInit(GPIO_InitTypeDef* GPIO_InitStruct)`
Behavior Description	Fills each GPIO_InitStruct member with its reset value.
Input Parameter	GPIO_InitStruct: pointer to a GPIO_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/*Initialize the GPIO Init Structure parameters*/
GPIO_InitTypeDef GPIO_InitStruct;
GPIO_StructInit(&GPIO_InitStruct);
```

7.2.4 GPIO_ReadBit

Function Name	GPIO_ReadBit
Function Prototype	`u8 GPIO_ReadBit(GPIO_TypeDef* GPIOx, u8 GPIO_Pin)`
Behavior Description	Reads the specified data port bit.
Input Parameter1	GPIOx: where x can be 0,1,...,9 to select the GPIO peripheral.
Input Parameter2	GPIO_Pin: Specifies the port bit to read.
Output Parameter	None
Return Parameter	The port pin value
Required preconditions	None
Called functions	None

Example:

```
/* Reads the seventh pin of the GPIO1 and store it in Read_Value variable*/
u8 Read_Value;
Read_Value = GPIO_ReadBit(GPIO1, GPIO_Pin_7);
```

7.2.5 GPIO_Read

Function Name	GPIO_Read
Function Prototype	`u8 GPIO_Read(GPIO_TypeDef* GPIOx)`
Behavior Description	Reads the specified GPIO data port
Input Parameter	GPIOx: where x can be 0,1,...,9 to select the GPIO peripheral.
Output Parameter	None
Return Parameter	GPIO data port byte value.
Required preconditions	None
Called functions	None

Example:

```
/* Read the GPIO2 data port and store it in Read_Value variable*/
u8 Read_Value;
Read_Value = GPIO_Read(GPIO2);
```

7.2.6 GPIO_WriteBit

Function Name	GPIO_WriteBit
Function Prototype	`void GPIO_WriteBit(GPIO_TypeDef* GPIOx,u8 GPIO_Pin, BitAction BitVal)`
Behavior Description	Sets or clears the selected data port bit
Input parameter1	GPIOx: where x can be 0,1, ..., 9 to select the GPIO peripheral.
Input parameter2	GPIO_Pin: specifies the port bit to be written.
Input Parameter3	BitVal: this parameter specifies the value to be written to the selected bit. Port_Val must be one of the BitAction enum values: Bit_RESET: to clear the port pin Bit_SET: to set the port pin
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/*Set the GPIO0 port pin 7*/
GPIO_WriteBit(GPIO0, GPIO_Pin_7, Bit_SET);
```

7.2.7 GPIO_Write

Function Name	GPIO_Write
Function Prototype	`void GPIO_Write(GPIO_TypeDef* GPIOx, u8 Port_Val)`
Behavior Description	Write the passed value in to the selected data GPIOx port register
Input Parameter1	GPIOx: where x can be 0,1, ..., 9 to select the GPIO peripheral.
Input parameter2	Port_Val: The value to be written to the data port register.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/*Write data to GPIO0 data port*/
GPIO_Write(GPIO0, 0xFD);
```

7.2.8 GPIO_ANAPinConfig

Function Name	GPIO_ANAPinConfig
Function Prototype	`void GPIO_ANAPinConfig(u8 GPIO_ANAChannel, FunctionalState NewState)`
Behavior Description	Enables or disables pins from GPIO 4 in Analog mode.
Input Parameter1	GPIO_ANAChannel: Specifies the port bit to be configured in Analog input. Refer to section "*GPIO_ANAChannel on page 79*" for more details on the allowed values of this parameter.
Input Parameter2	NewState: new state of the specified analog channel. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

GPIO_ANAChannel

Specifies the analog channel input.

This member can be one of the following values:

GPIO_ANAChannel	Meaning
GPIO_ANAChannel0	Configure the analog channel 0
GPIO_ANAChannel1	Configure the analog channel 1
GPIO_ANAChannel2	Configure the analog channel 2
GPIO_ANAChannel3	Configure the analog channel 3
GPIO_ANAChannel4	Configure the analog channel 4
GPIO_ANAChannel5	Configure the analog channel 5
GPIO_ANAChannel6	Configure the analog channel 6
GPIO_ANAChannel7	Configure the analog channel 7
GPIO_ANAChannelALL	Configure the all analog channel

Example:

```
/* Enable Analog channel 3 */
GPIO_ANAPinConfig(GPIO_ANAChannel3, ENABLE);
```

7.2.9 GPIO_EMIConfig

Function Name	GPIO_EMIConfig
Function Prototype	`void GPIO_EMIConfig(FunctionalState NewState)`
Behavior Description	Enables or disables GPIO 8 and 9 in EMI mode.
Input Parameter3	NewState: new state of the specified GPIO 8 and 9 is EMI interface. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable GPIO 8 and 9 in EMI mode */
GPIO_EMIConfig(ENABLE);
```

8 Vectored Interrupt Controller (VIC)

The driver may be used for several purposes, such as enabling and disabling interrupts (*IRQ*) and fast interrupts (*FIQ*), enabling and disabling individual *IRQ* channels, changing *IRQ* channel priorities, saving and restoring context and installing *IRQ* handlers.

The first section describes the data structures used in the VIC software library. The second one presents the software library functions.

8.1 VIC register structure

The VIC register structure *VIC_TypeDef* is defined in the ***91x_map.h*** file as follows:

```
typedef struct
{
vu32 ISR;
vu32 FSR;
vu32 RINTSR;
vu32 INTSR;
vu32 INTER;
vu32 INTECR;
vu32 SWINTR
vu32 SWINTCR;
vu32 PER;
vu32 EMPTY1[3];
vu32 VAR;
vu32 DVAR;
vu32 EMPTY2[50];
vu32 VAiR[16];
vu32 EMPTY3[48];
vu32 VCiR[16];
} VIC_TypeDef;
```

The following table presents the VIC registers:

Register	Description
ISR	IRQ Status Register
FSR	FIQ Status Register
RINTSR	Raw Interrupt Status Register
INTSR	Interrupt Select Register
INTER	Interrupt Enable Register
INTECR	Interrupt Enable Clear Register
SWINTR	Software Interrupt Register
SWINTCR	Software Interrupt clear Register
PER	Protection Enable Register
VAR	Vector Address Register
DVAR	Default Vector Address Register
VAiR	Vector Address 0-15 Register
VCiR	Vector Control 0-15 Register

The 2 VIC interfaces are declared in the same file:

```
#ifndef EXT
  #define EXT extern
#endif /* EXT */

#define AHB_VIC1_U          (0xFC000000) /* Secondary VIC1 UnBuffered Space */
#define AHB_VIC0_U          (0xFFFFF000) /* Primary VIC0 UnBuffered Space   */
...
#define VIC0_BASE           (AHB_VIC0_U)
#define VIC1_BASE           (AHB_VIC1_U)

#ifndef DEBUG
...
#define VIC0                ((VIC_TypeDef *)VIC0_BASE)
#define VIC1                ((VIC_TypeDef *)VIC1_BASE)
...
#else
#ifdef _VIC0
EXT VIC_TypeDef             *VIC0;
#endif /* _VIC0 */

#ifdef _VIC1
EXT VIC_TypeDef             *VIC1;
#endif /* _VIC1 */
...
#endif
```

When debug mode is used, VIC pointer is initialized in ***91x_lib.c*** file:

```
#ifdef __VIC0
  VIC0 = (VIC_TypeDef *)VIC0_BASE
#endif /* __VIC0 */

#ifdef __VIC1
  VIC1 = (VIC_TypeDef *)VIC1_BASE
#endif /* __VIC1 */
```

_VIC, _VIC0 and _VIC1 must be defined, in *91x_conf.h* file, to access the peripheral registers as follows:

```
#define _VIC
#define _VIC0
#define _VIC1
...
```

8.2 Software library functions

The following table enumerates the different functions of the VIC library.

Function Name	Description
VIC_DeInit	Deinitializes the VIC module registers to their default reset values.
VIC_GetIRQStatus	Gets the status of interrupts after IRQ masking.
VIC_GetFIQStatus	Gets the status of interrupts after FIQ masking.
VIC_GetSourceITStatus	Gets the status of the source interrupts before masking.
VIC_ITCmd	Enables or disables the interrupt request lines.
VIC_SWITConfig	Generates a software interrupt for the specific source interrupt before interrupt masking.
VIC_ProtectionCmd	Enables or disables the register access protection.
VIC_GetCurrentISRAdd	Gets the address of the currently active ISR.
VIC_GetISRVectAdd	Gets the ISR vector addresses.
VIC_Config	Configures the ISR, the line, the mode and the priority for each interrupt.

8.2.1 VIC_DeInit

Function Name	VIC_DeInit
Function Prototype	`void VIC_DeInit(void)`
Behavior Description	Deinitializes the VIC module registers to their default reset values.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	`SCU_AHBPeriphReset()`

Example:

```
/* Deinitialize the VIC */
VIC_DeInit();
```

8.2.2 VIC_GetIRQStatus

Function Name	VIC_GetIRQStatus
Function Prototype	`FlagStatus VIC_GetIRQStatus(u16 VIC_Source)`
Behavior Description	Gets the status of interrupts after IRQ masking.
Input Parameter	VIC_Source: specifies the number of the source line. Refer to section "*VIC_Source on page 85*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The status of the IRQ interrupt after masking (SET or RESET).
Required preconditions	...
Called functions	None

VIC_Source

The VIC_Source can take one of the following values defined in the ***91x_vic.h*** file:

VIC_Source	Meaning
WDG_ITLine	WDG interrupt line
SW_ITLine	Software interrupt line
ARMRX_ITLine	ARM RX interrupt line
ARMTX_ITLine	ARM TX interrupt line
TIM0_ITLine	TIM0 interrupt line
TIM1_ITLine	TIM1 interrupt line
TIM2_ITLine	TIM2 interrupt line
TIM3_ITLine	TIM3 interrupt line
USBHP_ITLine	High priority USB interrupt line
USBLP_ITLine	Low priority USB interrupt line
SCU_ITLine	SCU interrupt line
MAC_ITLine	MAC interrupt line
DMA_ITLine	DMA interrupt line
CAN_ITLine	CAN interrupt line
MC_ITLine	Motor Control interrupt line
ADC_ITLine	ADC interrupt line
UART0_ITLine	UART0 interrupt line
UART1_ITLine	UART1 interrupt line
UART2_ITLine	UART2 interrupt line
I2C0_ITLine	I2C0 interrupt line
I2C1_ITLine	I2C1 interrupt line
SSP0_ITLine	SSP0 interrupt line
SSP1_ITLine	SSP1 interrupt line
LVD_ITLine	LVD interrupt line
RTC_ITLine	RTC interrupt line
WIU_ITLine	WIU interrupt line
EXTIT0_ITLine	EXTIT0 interrupt line
EXTIT1_ITLine	EXTIT1 interrupt line
EXTIT2_ITLine	EXTIT2 interrupt line
EXTIT3_ITLine	EXTIT3 interrupt line
USBWU_ITLine	USBWU interrupt line
PFQBC_ITLine	PFQBC interrupt line

Example:

```
FlagStatus RTC_IT_Status;
```

```
/* Get the RTC interrupt status after IRQ masking */
RTC_IT_Status = VIC_GetIRQStatus(RTC_ITLine);
```

8.2.3 VIC_GetFIQStatus

Function Name	VIC_GetFIQStatus
Function Prototype	`FlagStatus VIC_GetFIQStatus(u16 VIC_Source)`
Behavior Description	Gets the status of interrupts after FIQ masking.
Input Parameter	VIC_Source: specifies the number of the source line. Refer to section "*VIC_Source on page 85*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The status of the FIQ interrupt after masking (SET or RESET).
Required preconditions	...
Called functions	None

VIC_Source

The VIC_Source can take one of the following values defined in the ***91x_vic.h*** file:

VIC_Source	Meaning
WDG_ITLine	WDG interrupt line
SW_ITLine	Software interrupt line
ARMRX_ITLine	ARM RX interrupt line
ARMTX_ITLine	ARM TX interrupt line
TIM0_ITLine	TIM0 interrupt line
TIM1_ITLine	TIM1 interrupt line
TIM2_ITLine	TIM2 interrupt line
TIM3_ITLine	TIM3 interrupt line
USBHP_ITLine	High priority USB interrupt line
USBLP_ITLine	Low priority USB interrupt line
SCU_ITLine	SCU interrupt line
MAC_ITLine	MAC interrupt line
DMA_ITLine	DMA interrupt line
CAN_ITLine	CAN interrupt line
MC_ITLine	Motor Control interrupt line
ADC_ITLine	ADC interrupt line
UART0_ITLine	UART0 interrupt line
UART1_ITLine	UART1 interrupt line
UART2_ITLine	UART2 interrupt line
I2C0_ITLine	I2C0 interrupt line

VIC_Source	Meaning
I2C1_ITLine	I2C1 interrupt line
SSP0_ITLine	SSP0 interrupt line
SSP1_ITLine	SSP1 interrupt line
LVD_ITLine	LVD interrupt line
RTC_ITLine	RTC interrupt line
WIU_ITLine	WIU interrupt line
EXTIT0_ITLine	EXTIT0 interrupt line
EXTIT1_ITLine	EXTIT1 interrupt line
EXTIT2_ITLine	EXTIT2 interrupt line
EXTIT3_ITLine	EXTIT3 interrupt line
USBWU_ITLine	USBWU interrupt line
PFQBC_ITLine	PFQBC interrupt line

Example:

```
FlagStatus SSP1_IT_Status;
/* Get the SSP1 interrupt status after FIQ masking */
SSP1_IT_Status = VIC_GetFIQStatus(SSP1_ITLine);
```

8.2.4 VIC_GetSourceITStatus

Function Name	VIC_GetSourceITStatus
Function Prototype	`FlagStatus VIC_GetSourceITStatus(u16 VIC_Source)`
Behavior Description	Gets the status of the source interrupts before masking.
Input Parameter	VIC_Source: specifies the number of the source line. Refer to section "*VIC_Source on page 85*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The status of the source interrupt before masking (SET or RESET).
Required preconditions	...
Called functions	None

VIC_Source

The VIC_Source can take one of the following values defined in the ***91x_vic.h*** file:

VIC_Source	Meaning
WDG_ITLine	WDG interrupt line
SW_ITLine	Software interrupt line
ARMRX_ITLine	ARM RX interrupt line
ARMTX_ITLine	ARM TX interrupt line
TIM0_ITLine	TIM0 interrupt line
TIM1_ITLine	TIM1 interrupt line
TIM2_ITLine	TIM2 interrupt line
TIM3_ITLine	TIM3 interrupt line
USBHP_ITLine	High priority USB interrupt line
USBLP_ITLine	Low priority USB interrupt line
SCU_ITLine	SCU interrupt line
MAC_ITLine	MAC interrupt line
DMA_ITLine	DMA interrupt line
CAN_ITLine	CAN interrupt line
MC_ITLine	Motor Control interrupt line
ADC_ITLine	ADC interrupt line
UART0_ITLine	UART0 interrupt line
UART1_ITLine	UART1 interrupt line
UART2_ITLine	UART2 interrupt line
I2C0_ITLine	I2C0 interrupt line
I2C1_ITLine	I2C1 interrupt line
SSP0_ITLine	SSP0 interrupt line
SSP1_ITLine	SSP1 interrupt line
LVD_ITLine	LVD interrupt line
RTC_ITLine	RTC interrupt line
WIU_ITLine	WIU interrupt line
EXTIT0_ITLine	EXTIT0 interrupt line
EXTIT1_ITLine	EXTIT1 interrupt line
EXTIT2_ITLine	EXTIT2 interrupt line
EXTIT3_ITLine	EXTIT3 interrupt line
USBWU_ITLine	USBWU interrupt line
PFQBC_ITLine	PFQBC interrupt line

Example:

```
FlagStatus RTC_IT_Status;
```

```
/* Get the RTC interrupt status before masking */
RTC_IT_Status = VIC_GetSourceITStatus(RTC_ITLine);
```

8.2.5 VIC_ITCmd

Function Name	VIC_ITCmd
Function Prototype	`void VIC_ITCmd(u16 VIC_Source, FunctionalState VIC_NewState)`
Behavior Description	Enables or disables the interrupt request lines.
Input Parameter1	VIC_Source: specifies the number of the source line. Refer to section "*VIC_Source on page 85*" for more details on the allowed values of this parameter.
Input Parameter2	VIC_NewState: specifies the line status. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

VIC_Source

The VIC_Source can take one of the following values defined in the ***91x_vic.h*** file:

VIC_Source	Meaning
WDG_ITLine	WDG interrupt line
SW_ITLine	Software interrupt line
ARMRX_ITLine	ARM RX interrupt line
ARMTX_ITLine	ARM TX interrupt line
TIM0_ITLine	TIM0 interrupt line
TIM1_ITLine	TIM1 interrupt line
TIM2_ITLine	TIM2 interrupt line
TIM3_ITLine	TIM3 interrupt line
USBHP_ITLine	High priority USB interrupt line
USBLP_ITLine	Low priority USB interrupt line
SCU_ITLine	SCU interrupt line
MAC_ITLine	MAC interrupt line
DMA_ITLine	DMA interrupt line
CAN_ITLine	CAN interrupt line
MC_ITLine	Motor Control interrupt line
ADC_ITLine	ADC interrupt line
UART0_ITLine	UART0 interrupt line
UART1_ITLine	UART1 interrupt line
UART2_ITLine	UART2 interrupt line
I2C0_ITLine	I2C0 interrupt line
I2C1_ITLine	I2C1 interrupt line
SSP0_ITLine	SSP0 interrupt line
SSP1_ITLine	SSP1 interrupt line
LVD_ITLine	LVD interrupt line
RTC_ITLine	RTC interrupt line
WIU_ITLine	WIU interrupt line
EXTIT0_ITLine	EXTIT0 interrupt line
EXTIT1_ITLine	EXTIT1 interrupt line
EXTIT2_ITLine	EXTIT2 interrupt line
EXTIT3_ITLine	EXTIT3 interrupt line
USBWU_ITLine	USBWU interrupt line
PFQBC_ITLine	PFQBC interrupt line

Example:

```
/* Enable the SSP1 interrupt request line */
```

```
VIC_ITCmd(SSP1_ITLine, ENABLE);
/* Disable the SSP2 interrupt request line */
VIC_ITCmd(SSP2_ITLine, DISABLE);
```

8.2.6 VIC_SWITCmd

Function Name	VIC_SWITCmd
Function Prototype	`void VIC_SWITCmd(u16 VIC_Source, FunctionalState VIC_NewState)`
Behavior Description	Generates a software interrupt for the specific source interrupt before interrupt masking.
Input Parameter1	VIC_Source: specifies the number of the source line. Refer to section "*VIC_Source on page 85*" for more details on the allowed values of this parameter.
Input Parameter2	VIC_NewState: specifies the software interrupt status. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

VIC_Source

The VIC_Source can take one of the following values defined in the ***91x_vic.h*** file:

VIC_Source	Meaning
WDG_ITLine	WDG interrupt line
SW_ITLine	Software interrupt line
ARMRX_ITLine	ARM RX interrupt line
ARMTX_ITLine	ARM TX interrupt line
TIM0_ITLine	TIM0 interrupt line
TIM1_ITLine	TIM1 interrupt line
TIM2_ITLine	TIM2 interrupt line
TIM3_ITLine	TIM3 interrupt line
USBHP_ITLine	High priority USB interrupt line
USBLP_ITLine	Low priority USB interrupt line
SCU_ITLine	SCU interrupt line
MAC_ITLine	MAC interrupt line
DMA_ITLine	DMA interrupt line
CAN_ITLine	CAN interrupt line
MC_ITLine	Motor Control interrupt line
ADC_ITLine	ADC interrupt line
UART0_ITLine	UART0 interrupt line
UART1_ITLine	UART1 interrupt line
UART2_ITLine	UART2 interrupt line
I2C0_ITLine	I2C0 interrupt line
I2C1_ITLine	I2C1 interrupt line
SSP0_ITLine	SSP0 interrupt line
SSP1_ITLine	SSP1 interrupt line
LVD_ITLine	LVD interrupt line
RTC_ITLine	RTC interrupt line
WIU_ITLine	WIU interrupt line
EXTIT0_ITLine	EXTIT0 interrupt line
EXTIT1_ITLine	EXTIT1 interrupt line
EXTIT2_ITLine	EXTIT2 interrupt line
EXTIT3_ITLine	EXTIT3 interrupt line
USBWU_ITLine	USBWU interrupt line
PFQBC_ITLine	PFQBC interrupt line

Example:

```
/* Generate a software interrupt in the SSP1 interrupt request line before masking */
```

```
VIC_SWITCmd(SSP1_ITLine, ENABLE);
/* Generate a software interrupt in the SSP2 interrupt request line before masking */
VIC_SWITCmd(SSP2_ITLine, ENABLE);
```

8.2.7 VIC_ProtectionCmd

Function Name	VIC_ProtectionCmd
Function Prototype	`void VIC_ProtectionCmd(FuncionalState VIC_NewState)`
Behavior Description	Enables or disables the register access protection.
Input Parameter	VIC_NewState: specifies the protection status. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

Example:

```
/* Enable the registers access protection */
VIC_ProtectionCmd(ENABLE);
```

8.2.8 VIC_GetCurrentISRAdd

Function Name	VIC_GetCurrentISRAdd
Function Prototype	`u32 VIC_GetCurrentISRAdd(VIC_TypeDef* VICx)`
Behavior Description	Gets the address of the current active ISR.
Input Parameter	VICx: specifies the VIC peripheral. This parameter can one of the following values: - VIC0: To select VIC0. - VIC1: To select VIC1.
Output Parameter	None
Return Parameter	The address of the active ISR.
Required preconditions	...
Called functions	None

Example:

```
u32 Address_Active_ISR
/* Get the address of the current active ISR for the VIC0 */
Address_Active_ISR = VIC_GetCurrentISRAdd(VIC0);
```

Note: Lines 0 to 15 are mapped on VIC0 and Lines 16 to 31 are mapped on VIC1.

8.2.9 VIC_GetISRVectAdd

Function Name	VIC_GetISRVectAdd
Function Prototype	`u32 VIC_GetISRVectAdd(u16 VIC_Source)`
Behavior Description	Configuration of the ISR vector addresses.
Input Parameter	VIC_Source: specifies the number of the source line. Refer to section "*VIC_Source on page 85*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The corresponding ISR vector address.
Required preconditions	...
Called functions	None

VIC_Source

The VIC_Source can take one of the following values defined in the ***91x_vic.h*** file:

VIC_Source	Meaning
WDG_ITLine	WDG interrupt line
SW_ITLine	Software interrupt line
ARMRX_ITLine	ARM RX interrupt line
ARMTX_ITLine	ARM TX interrupt line
TIM0_ITLine	TIM0 interrupt line
TIM1_ITLine	TIM1 interrupt line
TIM2_ITLine	TIM2 interrupt line
TIM3_ITLine	TIM3 interrupt line
USBHP_ITLine	High priority USB interrupt line
USBLP_ITLine	Low priority USB interrupt line
SCU_ITLine	SCU interrupt line
MAC_ITLine	MAC interrupt line
DMA_ITLine	DMA interrupt line
CAN_ITLine	CAN interrupt line
MC_ITLine	Motor Control interrupt line
ADC_ITLine	ADC interrupt line
UART0_ITLine	UART0 interrupt line
UART1_ITLine	UART1 interrupt line
UART2_ITLine	UART2 interrupt line
I2C0_ITLine	I2C0 interrupt line
I2C1_ITLine	I2C1 interrupt line
SSP0_ITLine	SSP0 interrupt line
SSP1_ITLine	SSP1 interrupt line
LVD_ITLine	LVD interrupt line
RTC_ITLine	RTC interrupt line
WIU_ITLine	WIU interrupt line
EXTIT0_ITLine	EXTIT0 interrupt line
EXTIT1_ITLine	EXTIT1 interrupt line
EXTIT2_ITLine	EXTIT2 interrupt line
EXTIT3_ITLine	EXTIT3 interrupt line
USBWU_ITLine	USBWU interrupt line
PFQBC_ITLine	PFQBC interrupt line

Example:

```
u32 ISR_Address;
/* Get the ISR vector address of the RTC */
ISR_Address = VIC_GetISRVectAdd(RTC_ITLine);
```

8.2.10 VIC_Config

Function Name	VIC_Config
Function Prototype	`void VIC_Config(u16 VIC_Source, VIC_ITLineMode VIC_LineMode, u8 VIC_Priority)`
Behavior Description	Configures the ISR, the line, the mode and the priority for each interrupt.
Input Parameter1	VIC_Source: specifies the number of the source line. Refer to section "*VIC_Source on page 85*" for more details on the allowed values of this parameter.
Input Parameter2	VIC_LineMode: specifies the type of interrupt of the source line. This parameter can be one of the following values: - VIC_IRQ: To configure the line as IRQ. - VIC_FIQ: To configure the line as FIQ.
Input Parameter3	VIC_Priority: specifies the priority of the interrupt, it can be a value from 0 to 15, 0 is the highest priority.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	VIC_ITModeConfig VIC_ISRVecAddConfig VIC_VecEnableConfig VIC_ITSourceConfig

VIC_Source

The VIC_Source can take one of the following values defined in the ***91x_vic.h*** file:

VIC_Source	Meaning
WDG_ITLine	WDG interrupt line
SW_ITLine	Software interrupt line
ARMRX_ITLine	ARM RX interrupt line
ARMTX_ITLine	ARM TX interrupt line
TIM0_ITLine	TIM0 interrupt line
TIM1_ITLine	TIM1 interrupt line
TIM2_ITLine	TIM2 interrupt line
TIM3_ITLine	TIM3 interrupt line
USBHP_ITLine	High priority USB interrupt line
USBLP_ITLine	Low priority USB interrupt line
SCU_ITLine	SCU interrupt line
MAC_ITLine	MAC interrupt line
DMA_ITLine	DMA interrupt line
CAN_ITLine	CAN interrupt line
MC_ITLine	Motor Control interrupt line
ADC_ITLine	ADC interrupt line
UART0_ITLine	UART0 interrupt line
UART1_ITLine	UART1 interrupt line
UART2_ITLine	UART2 interrupt line
I2C0_ITLine	I2C0 interrupt line
I2C1_ITLine	I2C1 interrupt line
SSP0_ITLine	SSP0 interrupt line
SSP1_ITLine	SSP1 interrupt line
LVD_ITLine	LVD interrupt line
RTC_ITLine	RTC interrupt line
WIU_ITLine	WIU interrupt line
EXTIT0_ITLine	EXTIT0 interrupt line
EXTIT1_ITLine	EXTIT1 interrupt line
EXTIT2_ITLine	EXTIT2 interrupt line
EXTIT3_ITLine	EXTIT3 interrupt line
USBWU_ITLine	USBWU interrupt line
PFQBC_ITLine	PFQBC interrupt line

Example:

```
void SSP1_ITHandler(void)
{
...
}
/* Configure the SSP1 interrupt as IRQ and set its priority to 5 */
VIC_Config(SSP1_ITLine,VIC_IRQ,5);
```

9 Wake-Up Interrupt Unit (WIU)

The WIU driver may be used for several purposes, such as enabling and disabling interrupt lines, selecting the edge sensitivity, interrupt or wake-up mode.

9.1 WIU register structure

The WIU register structure *WIU_TypeDef* is defined in the ***91x_map.h*** file as follows:

```
typedef struct
{
vu32  CTRL;   /* Control Register            */
vu32  MR;     /* Mask Register               */
vu32  TR;     /* Trigger Register            */
vu32  PR;     /* Pending Register            */
vu32  INTR;   /* Software Interrupt Register */
} WIU_TypeDef;
```

The following table presents the WIU registers:

Register	Description
CTRL	WIU Control register: used to enable input as wake-up or interrupt
MR	WIU Mask Register: used to mask generation of the interrupt or Wake-up event
TR	WIU Trigger register: used to specify whether the wake-up event is rising or falling edge triggered
PR	WIU Pending register: It is set when a wake-up input is pending
INTR	WIU Software Interrupt register: Software initiated interrupt

The WIU is declared in the file below

```
#ifndef EXT
 #Define EXT extern
#endif
...
#define AHB_APB_BRDG0_U    (0x58000000) /* AHB/APB Bridge 0 UnBuffered Space */
#define AHB_APB_BRDG0_B    (0x48000000) /* AHB/APB Bridge 0 Buffered Space   */
...
#define APB_WIU_OFST       (0x00001000) /* Offset of WIU */
...
#ifndef Buffered
#define AHBAPB0_BASE           (AHB_APB_BRDG0_U)
...
#else /* Buffered */
...
#define AHBAPB0_BASE           (AHB_APB_BRDG0_B)

/* WIU Base Address definition*/
#define WIU_BASE          (AHBAPB0_BASE + APB_WIU_OFST)


...
/* WIU peripheral declaration*/
```

```
#ifndef DEBUG
...
#define WIU          ((WIU_TypeDef *) WIU_BASE)
...
#else
...
EXT  WIU_TypeDef           *WIU;
...

#endif

When debug mode is used, WIU pointer is initialized in 91x_lib.c file:
#ifdef _WIU
  WIU = (WIU_TypeDef *)WIU_BASE
#endif /* _WIU */
```

_WIU must be defined, in ***91x_conf.h*** file, to access the peripheral registers as follows:

```
#define _WIU
...
```

9.2 Software library functions

The following table enumerates the different functions of the WIU Library.

Function Name	Description
WIU_Init	Initializes the WIU according to the specified parameters in the WIU_InitTypeDef structure
WIU_DeInit	Deinitializes the WIU registers to their default reset values
WIU_StructInit	Fills each WIU_InitStruct member with its reset value.
WIU_Cmd	Enables or disables the WIU peripheral.
WIU_GetITStatus	Checks whether the specified WIU line is asserted or not
WIU_ClearITPendingBit	Clears the pending bit of the selected WIU line
WIU_GenerateSWInterrupt	Generates a Software interrupt for the specified line
WIU_GetFlagStatus	Checks whether the specified WIU line flag is set or not.
WIU_ClearFlag	Clears the WIU line pending flag.

9.2.1 WIU_Init

Function Name	WIU_Init
Function Prototype	`void WIU_Init(WIU_InitTypeDef* WIU_InitStruct)`
Behavior Description	Initializes WIU peripheral according to the specified parameters in the WIU_InitTypeDef structure.
Input Parameter	WIU_InitStruct: Pointer to a WIU_InitTypeDef structure that contains the configuration information for the WIU peripheral. Refer to section "*WIU_initTypeDef on page 101*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

WIU_initTypeDef

The WIU_InitTypeDef structure defines the control setting for the WIU cell, it is defined in the ***91x_WIU.h***

```
typedef struct
{
u8 WIU_TriggerEdge;
u32 WIU_Line;
}WIU_InitTypeDef;WIU_TriggerEdge
```

WIU_TriggerEdge

Specifies the triggering edge of the Wake-up line.

This member can be one of the following values.

WIU_TriggerEdge	Meaning
WIU_FallingEdge	Wake-up lines trigger on falling edge.
WIU_RisingEdge	Wake-up lines trigger on rising edge

WIU_Line

Specifies the Wake-up line to be configured, it can be one of the following values:

Lines value	Corresponding Line
WIU_Line0	Wake-up line 0
WIU_Line1	Wake-up line 1
WIU_Line2	Wake-up line 2
WIU_Line3	Wake-up line 3
WIU_Line4	Wake-up line 4
WIU_Line5	Wake-up line 5
WIU_Line6	Wake-up line 6

Lines value	Corresponding Line
WIU_Line7	Wake-up line 7
WIU_Line8	Wake-up line 8
WIU_Line9	Wake-up line 9
WIU_Line10	Wake-up line 10
WIU_Line11	Wake-up line 11
WIU_Line12	Wake-up line 12
WIU_Line13	Wake-up line 13
WIU_Line14	Wake-up line 14
WIU_Line15	Wake-up line 15
WIU_Line16	Wake-up line 16
WIU_Line17	Wake-up line 17
WIU_Line18	Wake-up line 18
WIU_Line19	Wake-up line 19
WIU_Line20	Wake-up line 20
WIU_Line21	Wake-up line 21
WIU_Line22	Wake-up line 22
WIU_Line23	Wake-up line 23
WIU_Line24	Wake-up line 24
WIU_Line25	Wake-up line 25
WIU_Line26	Wake-up line 26
WIU_Line27	Wake-up line 27
WIU_Line28	Wake-up line 28
WIU_Line29	Wake-up line 29
WIU_Line30	Wake-up line 30
WIU_Line31	Wake-up line 31

Example:

The following example illustrates how to configure the WIU unit :

```
{
...
/* Set the WIU_InitTypeDef structure with the needed configuration */
WIU_InitStructure.WIU_Line = WIU_Line1;
WIU_InitStructure.WIU_TriggerEdge = WIU_FallingEdge;
/* Configure the WIU unit */
WIU_Init(&WIU_InitStructure);

...
}
```

9.2.2 WIU_DeInit

Function Name	WIU_DeInit
Function Prototype	`void WIU_DeInit(void)`
Behavior Description	Deinitalizes WIU peripheral registers to their default reset values
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	`SCU_APBPeriphReset()`

9.2.3 WIU_StructInit

Function Name	WIU_StructInit
Function Prototype	`void WIU_StructInit(WIU_InitTypeDef* WIU_InitStruct)`
Behavior Description	Fills each WIU_InitStruct member with its reset value.
Input Parameter	WIU_InitStruct: pointer to a WIU_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

9.2.4 WIU_Cmd

Function Name	WIU_Cmd
Function Prototype	`void WIU_Cmd(FunctionalState NewState)`
Behavior Description	Enables or disables the specified WIU peripheral.
Input Parameter1	NewState: new state of the WIU peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

9.2.5 WIU_GenerateSWInterrupt

Function Name	WIU_GenerateSWInterrupt
Function Prototype	`void WIU_GenerateSWInterrupt(u32 WIU_Line)`
Behavior Description	Generates a software interrupt for the specified line.
Input Parameter2	WIU_Line : Wake-up line.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
WIU_GenerateSWInterrupt (WIU_Line0);
```

9.2.6 WIU_GetFlagStatus

Function Name	WIU_GetFlagStatus
Function Prototype	`void WIU_GetFlagStatus(u32 WIU_Line)`
Behavior Description	Checks whether the specified WIU line flag is set or not
Input Parameter2	WIU_Line: Wake-up line.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
WIU_GetFlagStatus (WIU_Line0);
```

9.2.7 WIU_ClearFlag

Function Name	WIU_ClearFlag
Function Prototype	`void WIU_ClearFlag(u32 WIU_Line)`
Behavior Description	Clears the WIU's line pending flag.
Input Parameter2	WIU_Line : Wake-up line.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
WIU_ClearFlag (WIU_Line0);
```

9.2.8 WIU_GetITStatus

Function Name	WIU_GetITStatus
Function Prototype	`void WIU_GetITStatus(u32 WIU_Line)`
Behavior Description	Checks whether the specified WIU line is asserted or not.
Input Parameter2	WIU_Line: Wake-up line
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
WIU_GetITStatus(WIU_Line0);
```

9.2.9 WIU_ClearITPendingBit

Function Name	WIU_ClearITPendingBit
Function Prototype	`void WIU_ClearITPendingBit(u32 WIU_Line)`
Behavior Description	Clears the pending bit of the selected WIU line
Input Parameter2	WIU_Line: Wake-up line to clear its pending bit.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

The following example clears the WIU line0 pending bit:

```
WIU_ClearITPendingBit(WIU_Line0);
```

10 Real Time Clock (RTC)

The RTC block combines a complete time of day clock with alarm, periodic interrupt, tamper detection and 9999-year calendar. The time is in 24 hour mode, and time/calendar values are stored in binary-coded decimal format.

10.1 RTC register structure

The RTC register structure *RTC_TypeDef* is defined in the ***91x_map.h*** file as follows:

```
typedef struct
{
  vu32 TR;          /* Time Register       */
  vu32 DTR;         /* Date Register       */
  vu32 ATR;         /* Alarm time Register */
  vu32 CR;          /* Control Register    */
  vu32 SR;          /* Status Register     */
  vu32 MILR;        /* Milliseconds Register   */
}RTC_TypeDef;
```

The following table presents the RTC registers:

Register	Description
TR	Time register
DTR	Date register
ATR	Alarm Time register
CR	Control register
SR	Status register
MILR	Milliseconds register

The RTC is declared in the file below

```
#ifndef EXT
 #Define EXT extern
#endif
...
#define AHB_APB_BRDG1_U    (0x5C000000) /* AHB/APB Bridge 1 UnBuffered Space */
#define AHB_APB_BRDG1_B    (0x4C000000) /* AHB/APB Bridge 1 Buffered Space   */
...
#define APB_RTC_OFST       (0x00001000) /* Offset of RTC */
...
#ifndef Buffered
#define AHBAPB1_BASE           (AHB_APB_BRDG1_U)
...
#else /* Buffered */
...
#define AHBAPB1_BASE           (AHB_APB_BRDG1_B)

/* RTC Base Address definition*/
#define RTC_BASE          (AHBAPB1_BASE + APB_RTC_OFST)
```

```
...
/* RTC peripheral declaration*/

#ifndef DEBUG
...
#define RTC         ((RTC_TypeDef *) RTC_BASE)
...
#else
...
#ifdef _RTC
EXT RTC_TypeDef           *RTC;
#endif /* _RTC */
...

#endif
```

When debug mode is used, RTC pointer is initialized in ***91x_lib.c*** file:

```
#ifdef _RTC
  RTC = (RTC_TypeDef *)RTC_BASE
#endif /* _RTC */
```

_RTC must be defined, in ***91x_conf.h*** file, to access the peripheral registers as follows:

```
#define _RTC
...
```

10.2 Software library functions

The following table enumerates the different functions of the RTC library.

Function Name	Description
RTC_DeInit	Resets RTC registers
RTC_SetDate	Sets Date (weekday, day, month, year* and century*) *example : 2006 , century = 20, year = 06.
RTC_SetTime	Sets Time (milliseconds, seconds, minutes, hours)
RTC_SetAlarm	Sets Alarm (alarm seconds, alarm minutes, alarm hour, alarm day)
RTC_GetDate	Gets current RTC date
RTC_GetTime	Gets current RTC time
RTC_GetAlarm	Gets configured alarm time
RTC_TamperConfig	Configures Tamper detection mode (level or edge) and polarity (high or low)
RTC_TamperCmd	Enables or disables Tamper detection
RTC_AlarmCmd	Enables or disables Alarm
RTC_CalibClockCmd	Enables or disables RTC Calibration Clock output
RTC_SRAMBattPowerCmd	Enables or Disables SRAM backup with VBAT
RTC_PeriodicIntConfig	Configures Periodic interrupt frequency
RTC_ITConfig	Enables or disables RTC interrupts
RTC_GetFlagStatus	Gets a flag status
RTC_ClearFlag	Clears a status flag

10.2.1 RTC_DeInit

Function Name	RTC_DeInit
Function Prototype	`void RTC_DeInit(void)`
Behavior Description	Resets RTC registers
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	`SCU_APBPeriphReset()`

10.2.2 RTC_SetDate

Function Name	RTC_SetDate
Function Prototype	`void RTC_SetDate (RTC_DATE Date)`
Behavior Description	Set Date : weekday, day, month, year and century.
Input Parameter	Date: structure of type *RTC_DATE*
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

RTC_DATE

```
typedef struct
{
  u8 century;
  u8 year;
  u8 month;
  u8 day;
  u8 weekday;
}RTC_DATE;
```

Structure members:

member	Range
century	0- 99
year	0 - 99
month	1-12
day	1- 31
weekday	1- 7

10.2.3 RTC_SetTime

Function Name	RTC_SetTime
Function Prototype	`void RTC_SetTime(RTC_TIME Time)`
Behavior Description	Set Time: milliseconds, seconds, minutes, hours
Input Parameter	TIme: structure of type *RTC_TIME*
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

RTC_TIME

```
typedef struct
{
  u8 hours;
  u8 minutes;
  u8 seconds;
  u16 milliseconds;
}RTC_TIME;
```

Structure members:

member	Range
hours	0- 23
minutes	0- 59
seconds	0- 59
milliseconds	0 - 999
weekday	1-7

10.2.4 RTC_SetAlarm

Function Name	RTC_SetAlarm
Function Prototype	`void RTC_SetAlarm(RTC_ALARM Alarm)`
Behavior Description	Sets the Alarm time
Input Parameter	Alarm: structure of type *RTC_ALARM*
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

RTC_ALARM

```
typedef struct
{
  u8 day;
  u8 hours;
  u8 minutes;
  u8 seconds;
}RTC_ALARM;
```

Structure members:

member	Range
day	1 - 31
hours	0 - 23
minutes	0 - 59
seconds	0 - 59

10.2.5 RTC_GetDate

Function Name	RTC_GetDate
Function Prototype	`void RTC_GetDate(u8 Format, RTC_DATE * Date)`
Behavior Description	Returns current RTC date
Input1 Parameter	Format : BINARY or BCD
Input2 Parameter	Date: pointer to structure of type *RTC_DATE* to be filled by function
Output Parameter	Date
Return Parameter	None
Required preconditions	None
Called functions	None

Format

mode	Meaning
BINARY	Binary coded values are returned
BCD	BCD coded values are returned

10.2.6 RTC_GetTime

Function Name	RTC_GetDate
Function Prototype	`void RTC_GetTime(u8 Format, RTC_TIME * Time)`
Behavior Description	Returns current RTC time
Input1 Parameter	Format: BINARY or BCD
Input2 Parameter	Time: pointer to a structure of type *RTC_TIME* to be filled by function
Output Parameter	Time
Return Parameter	None
Required preconditions	None
Called functions	None

10.2.7 RTC_GetAlarm

Function Name	RTC_GetAlarm
Function Prototype	`void RTC_GetAlarm(u8 Format, RTC_ALARM * Alarm)`
Behavior Description	Returns current RTC configured alarm time
Input1 Parameter	Format: BINARY or BCD
Input2 Parameter	Alarm: pointer to a structure of type *RTC_ALARM* to be filled by function
Output Parameter	Alarm
Return Parameter	None
Required preconditions	None
Called functions	None

10.2.8 RTC_TamperConfig

Function Name	RTC_TamperConfig
Function Prototype	`void RTC_TamperConfig(u32 TamperMode, u32 TamperPol)`
Behavior Description	Configures Tamper detection mode (level or edge) and polarity (high or low)
Input Parameter1	TamperMode : Tamper detection mode (level or edge) Refer to *TamperMode on page 114* section for details about allowed values
Input Parameter2	TamperPol: Tamper detection polarity (high or Low) Refer to *TamperPol on page 114* section for details about allowed values
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

TamperMode

TamperMode	Meaning
RTC_TamperMode_Edge	Tamper detection is on edge
RTC_TamperMode_Level	Tamper detection is on level

TamperPol

TamperPol	Meaning
RTC_TamperPol_High	Tamper event triggered when Tamper input goes high
RTC_TamperPol_Low	Tamper event triggered when Tamper input goes high

10.2.9 RTC_TamperCmd

Function Name	RTC_TamperCmd
Function Prototype	`void RTC_TamperCmd(FunctionlState NewState)`
Behavior Description	Enables or disables RTC Tamper detection
Input Parameter	NewState: ENABLE or DISABLE
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

10.2.10 RTC_AlarmCmd

Function Name	RTC_AlarmCmd
Function Prototype	`void RTC_AlarmCmd(FunctionlState NewState)`
Behavior Description	Enables or disables RTC Tamper detection
Input Parameter	NewState: ENABLE or DISABLE
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

10.2.11 RTC_CalibClockCmd

Function Name	RTC_CalibClockCmd
Function Prototype	`void RTC_CalibClockCmd(FunctionalState NewState)`
Behavior Description	Enables or disables the RTC claibration clock output
Input Parameter	NewState: ENABLE or DISABLE
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

10.2.12 RTC_SRAMBattPowerCmd

Function Name	RTC_SRAMBattPowerCmd
Function Prototype	`void RTC_SRAMBattPowerCmd(FunctionalState NewState)`
Behavior Description	Enables or disables SRAM power backup by VBAT
Input Parameter	NewState: ENABLE or DISABLE
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

10.2.13 RTC_PeriodicIntConfig

Function Name	RTC_PeriodicIntConfig
Function Prototype	`void RTC_PeriodicIntConfig(u32 PeriodicClock)`
Behavior Description	Configures the Periodic clock frequency
Input Parameter	PeriodicClock: Periodic Clock frequency refer to *PeriodicClock on page 116* section for details about allowed values
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

PeriodicClock

PeriodicClock	Meaning
RTC_Periodic_2Hz	Periodic Clock = 2 Hz
RTC_Periodic_16Hz	Periodic Clock = 16 Hz
RTC_Periodic_128Hz	Periodic Clock = 128 Hz
RTC_Periodic_1024Hz	Periodic Clock = 1024 Hz
RTC_Periodic_DISABLE	Periodic clock generation disabled

10.2.14 RTC_ITConfig

Function Name	RTC_ITConfig
Function Prototype	`void RTC_ITConfig(u32 RTC_IT, FunctionalState NewState)`
Behavior Description	Enables or disables the specified RTC interrupts.
Input Parameter1	RTC_IT: specifies the RTC interrupts sources to be enabled or disabled.
Input Parameter2	NewState: new state of the specified RTC interrupts. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

RTC_IT

RTC_IT	Meaning
RTC_IT_Per	Periodic interrupt
RTC_IT_Alarm	Alarm interrupt
RTC_IT_Tamper	Tamper interrupt

10.2.15 RTC_GetFlagStatus

Function Name	RTC_GetFlagStatus
Function Prototype	`FlagStatus RTC_GetFlagStatus(u32 RTC_FLAG)`
Behavior Description	Checks whether the specified RTC flag is set or not.
Input Parameter	RTC_FLAG: specifies the flag to check. Refer to *RTC_FLAG on page 117* section for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of RTC_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

RTC_FLAG

RTC_FLAG	Meaning
RTC_FLAG_Per	Periodic interrupt flag
RTC_FLAG_Alarm	Alarm flag
RTC_FLAG_Tamper	Tamper flag

10.2.16 RTC_ClearFlag

Function Name	RTC_ClearFlag
Function Prototype	`void RTC_ClearFlag(u32 RTC_FLAG)`
Behavior Description	Clears a RTC flag
Input Parameter2	RTC_FLAG: specifies the flag to clear. Refer to *RTC_FLAG on page 117* section for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

11 Watchdog Timer (WDG)

The Watchdog Timer peripheral can be used as free-running timer or as Watchdog to resolve processor malfunctions due to hardware or software failure.

The Watchdog supports the following features:

16-bit down-counter

8-bit clock prescaler

Safe reload sequence

Free-running timer mode

End of Count interrupt generation

The first section describes the data structure used in the WDG software library. The second one presents the software library functions.

11.1 WDG register structure

The WDG register structure *WDG_TypeDef* is defined in the ***91x_map.h*** file as follows:

```
typedef struct
{
  vu16 WDG_CR;         /* Control Register        */
  vu16 EMPTY1;
  vu16 WDG_PR;         /* Presclar Register       */
  vu16 EMPTY2;
  vu16 WDG_VR;         /* Pre-load Value Register */
  vu16 EMPTY3;
  vu16 WDG_CNT;        /* Counter Register        */
  vu16 EMPTY4;
  vu16 WDG_SR;         /* Status Register         */
  vu16 EMPTY5;
  vu16 WDG_MR;         /* Mask Register           */
  vu16 EMPTY6;
  vu16 WDG_KR;         /* Key Register            */
  vu16 EMPTY7;
} WDG_TypeDef;
```

The following table presents the WDG registers:

Register	Description
WDG_CR	This register controls and enables Watchdog Timer operations
WDG_PR	8-bit prescaler, WDG clock divider
WDG_VR	This register contains the 16-bit preload value to the WDG
WDG_CNT	This register contains the current counter value
WDG_SR	WDG Status register
WDG_MR	WDG Mask register
WDG_KR	Key Register: The WDG is loaded with VR value when the Key Register is written twice.

The WDG interface is declared in the same file:

```
#ifndef EXT
 #Define EXT extern
#endif
...
#define AHB_APB_BRDG1_U    (0x5C000000) /* AHB/APB Bridge 1 UnBuffered Space */
#define AHB_APB_BRDG1_B    (0x4C000000) /* AHB/APB Bridge 1 Buffered Space   */
...
#define APB_WDG_OFST       (0x0000B000) /* Offset of WDG */
...
#ifndef Buffered
#define AHBAPB1_BASE           (AHB_APB_BRDG1_U)
...
#else /* Buffered */
...
#define AHBAPB1_BASE           (AHB_APB_BRDG1_B)

/* WDG Base Address definition*/
#define WDG_BASE          (AHBAPB1_BASE + APB_WDG_OFST)
...

#ifndef DEBUG
...

/* WDG peripheral declaration*/

#define WDG         ((WDG_TypeDef *) WDG_BASE)
...
#else   /* DEBUG */
...
#ifdef _WDG

EXT WDG_TypeDef           *WDG;

#endif /* _WDG */
...

#endif
```

When debug mode is used, WDG pointer is initialized in ***91x_lib.c*** file:

```
#ifdef _WDG
  WDG = (WDG_TypeDef *)WDG_BASE
#endif /* _WDG */
```

_WDG must be defined, in ***91x_conf.h*** file, to access the peripheral registers as follows:

```
#define _WDG
```

11.2 Software library functions

The following table enumerates the different functions of the WDG library.

Function Name	Description
WDG_DeInit	Initializes the WDG peripheral registers to their default reset values.
WDG_Init	Initializes the WDG peripheral according to the specified parameters in the *WDG_InitStruct.*
WDG_StructInit	Fills each *WDG_InitStruct* member with its reset value.
WDG_Cmd	Enables or disables the WDG peripheral.
WDG_ITConfig	Enables or disables the specified WDG interrupts.
WDG_GetITStatus	Checks if the WDG End of Count(EC) interrupt is occurred or not.
WDG_ClearITPendingBit	Clears the WDG End of Count(EC) interrupt pending bit.
WDG_GetCounter	Gets the WDG counter value.
WDG_GetFlagStatus	Checks whether the WDG End of Count(EC) flag is set or not.
WDG_ClearFlag	Clears the WDG End of Count(EC) Flag.

11.2.1 WDG_DeInit

Function Name	WDG_DeInit
Function Prototype	`void WDG_DeInit(void)`
Behavior Description	Initializes the WDG peripheral registers to their default reset values.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	`SCU_APBPeriphReset()`

Example:

```
...
WDG_DeInit(); /* set all WDG Peripheral registers to their reset value */
...
```

11.2.2 WDG_Init

Function Name	WDG_Init
Function Prototype	`void WDG_Init(void)`
Behavior Description	Initializes the WDG peripheral according to the specified parameters in the WDG_InitStruct.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

WDG_InitTypeDef

The WDG_InitTypeDef structure is defined in the ***91x_wdg.h*** file:

```
typedef struct
{
   u16 WDG_Mode;
   u16 WDG_ClockSource;
   u16 WDG_Prescaler;
   u16 WDG_Preload;

} WDG_InitTypeDef;
```

WDG_Mode

This field specifies the WDG running mode:Free-running Timer mode or watchdog mode.

This member can be one of the following values:

WDG_Mode	Value	Meaning
WDG_Mode_Wdg	0X0001	WDG configured to run in watchdog mode.
WDG_Mode_Timer	0XFFFE	WDG configured to be in Free-running Timer mode.

WDG_ClockSource

This field specifies if the External clock (32 kHz RTC clock) or the APB clock signal will be used as counting clock.

This member can be one of the following values:

WDG_ClockSource	Value	Meaning
WDG_ClockSource_Rtc	0x0004	External clock (32 kHz RTC clock) will be used as counting clock.
WDG_ClockSource_Apb	0x0000	The APB clock signal will be used as counting clock.

WDG_Prescaler

Specifies the Prescaler value to divide the clock source. The clock of the Watchdog Timer Counter is divided by PR[7:0] + 1.

This member must be a number between 0x00 and 0xFF.

WDG_Preload

This value is loaded in the WDG Counter when it starts counting. The time (µs) needed to reach the end of count is given by:

$$\frac{\text{Prescaler} \times \text{Preload} \times \text{Tclk}}{1000}$$

where Tclk is the Clock period measured in ns.

This member must be a number between 0x0000 and 0xFFFF.

Example:

```
{
...
   WDG_InitTypeDef Init_Structure;
   Init_Structure.WDG_Mode = WDG_Mode_Timer
   Init_Structure.WDG_Prescaler = 0xF0;
   Init_Structure.WDG_Preload = 0xFF00;
   WDG_Init (&Init_Structure);
  ...
}
```

11.2.3 WDG_StructInit

Function Name	WDG_StructInit
Function Prototype	`void WDG_StructInit(WDG_InitTypeDef* WDG_InitStruct)`
Behavior Description	Fills each WDG_InitStruct member with its reset value.
Input Parameter	WDG_InitStruct: pointer to a WDG_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
...
   WDG_InitTypeDef WDG_InitStructure;
   WDG_StructInit(&WDG_InitStructure);
 ...
```

11.2.4 WDG_Cmd

Function Name	WDG_Cmd
Function Prototype	`void WDG_Cmd(FunctionalState NewState)`
Behavior Description	Enables or disables the specified WDG peripheral.
Input Parameter1	NewState: new state of the WDG peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

The following example illustrates how to enable the *WDG* peripheral:

```
...
WDG_Cmd(ENABLE);
...
```

11.2.5 WDG_ITConfig

Function Name	WDG_ITConfig
Function Prototype	`void WDG_ITConfig(FunctionalState NewState))`
Behavior Description	Enables or disables the WDG End of Count(EC) interrupt.
Input Parameter1	NewState: This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
...
WDG_ITConfig(ENABLE);
...
```

11.2.6 WDG_GetITStatus

Function Name	WDG_GetFlagStatus
Function Prototype	`ITStatus WDG_GetITStatus(void)`
Behavior Description	Checks whether the WDG End of Count(EC) interrupt has occurred or not.
Input Parameter1	None
Output Parameter	None
Return Parameter	The new state of WDG End of Count(EC) interrupt (SET or RESET).
Required preconditions	None
Called functions	None

Example:

The following example illustrates how to test if the *end of count* flag is set or reset:

```
...
if (WDG_GetFlagStatus(WDG_U,WDG_IT_ECM)== SET)
...
```

11.2.7 WDG_ClearITPendingBit

Function Name	WDG_ClearITPendingBit
Function Prototype	`void WDG_ClearITPendingBit(void)`
Behavior Description	Clears the WDG End of Count(EC) interrupt pending bit.
Input Parameter1	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

The following example illustrates how to clear the end of count flag:

```
...
    WDG_ClearITPendingBit(WDG,WDG_FLAG_EC);
...
```

11.2.8 WDG_GetCounter

Function Name	WDG_GetCounter
Function Prototype	`u16 WDG_GetCounter(void)`
Behavior Description	Gets the WDG current counter value.
Input Parameter	None
Output Parameter	None
Return Parameter	WDG current counter value.
Required preconditions	None
Called functions	None

Example:

The following example illustrates how to get the counter value of the WDG timer:

```
...
u16 CounterValue = WDG_GetCounter();
```

11.2.9 WDG_GetFlagStatus

Function Name	WDG_GetFlagStaus
Function Prototype	`FlagStatus WDG_GetFlagStatus(void)`
Behavior Description	Checks whether the WDG End of Count(EC) flag is set or not.
Input Parameter	None
Output Parameter	None
Return Parameter	None

Required preconditions	None
Called functions	None

Example:

Example illustrating the use of WDG_GetFlagStaus function:

```
/*wait for the end of count on mode free running timer*/
while(WDG_GetFlagStatus()== RESET);
...
```

11.2.10 WDG_GetFlagStatus

Function Name	WDG_ClearFlag
Function Prototype	`void WDG_ClearFlag(void)`
Behavior Description	Clears the WDG End of Count(EC) Flag.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
...
WDG_ClearFlag();
...
```

12 16-bit Timer (TIM)

The TIM driver may be used for a variety of purposes, including timing operation, Input capture, output compare and PWM generation.

12.1 TIM register structure

The TIM register structure *TIM_TypeDef* is defined in the ***91x_map.h*** file as follows:

```
typedef struct
{
  vu16 IC1R;
  vu16 EMPTY1;
  vu16 IC2R;
  vu16 EMPTY2;
  vu16 OC1R;
  vu16 EMPTY3;
  vu16 OC2R;
  vu16 EMPTY4;
  vu16 CNTR;
  vu16 EMPTY5;
  vu16 CR1;
  vu16 EMPTY6;
  vu16 CR2;
  vu16 EMPTY7;
  vu16 SR;
  vu16 EMPTY8;
} TIM_TypeDef;
```

The following table presents the TIM registers:

Register	Description
IC1R	Input Capture 1 Register
IC2R	Input Capture 2 Register
OC1R	Output Compare 1 Register
OC2R	Output Compare 2 Register
CNTR	Counter Register
CR1	Control Register 1
CR2	Control Register 2
SR	Status Register

The four TIM interfaces are declared in the same file:

```
#ifndef EXT
 #Define EXT extern
#endif
...
#define AHB_APB_BRDG0_U    (0x58000000) /* AHB/APB Bridge 0 UnBuffered Space */
#define AHB_APB_BRDG0_B    (0x48000000) /* AHB/APB Bridge 0 Buffered Space   */
...
#define APB_TIM0_OFST       (0x00002000) /* Offset of TIM0 */
```

```
#define APB_TIM1_OFST       (0x00003000) /* Offset of TIM1 */
#define APB_TIM2_OFST       (0x00004000) /* Offset of TIM0 */
#define APB_TIM3_OFST       (0x00005000) /* Offset of TIM1 */

...
#ifndef Buffered
#define AHBAPB0_BASE           (AHB_APB_BRDG0_U)
...
#else /* Buffered */
...
#define AHBAPB0_BASE           (AHB_APB_BRDG0_B)

/* TIM Base Address definition*/
#define TIM0_BASE          (AHBAPB0_BASE + APB_TIM0_OFST)
#define TIM1_BASE          (AHBAPB0_BASE + APB_TIM1_OFST)
#define TIM2_BASE          (AHBAPB0_BASE + APB_TIM2_OFST)
#define TIM3_BASE          (AHBAPB0_BASE + APB_TIM3_OFST)

...
/* TIM peripheral declaration*/

#ifndef DEBUG
...
#define TIM0    ((TIM_TypeDef *) TIM0_BASE)
#define TIM1    ((TIM_TypeDef *) TIM1_BASE)
#define TIM2    ((TIM_TypeDef *) TIM2_BASE)
#define TIM3    ((TIM_TypeDef *) TIM3_BASE)
...
#else
...
#ifdef _TIM0
EXT TIM_TypeDef            *TIM0;
#endif /* _TIM0 */

#ifdef _TIM1
EXT TIM_TypeDef            *TIM1;
#endif /* _TIM1 */

#ifdef _TIM2
EXT TIM_TypeDef            *TIM2;
#endif /* _TIM2 */

#ifdef _TIM3
EXT TIM_TypeDef            *TIM3;
#endif /* _TIM3 */
...

#endif
```

When debug mode is used, TIM pointers are initialized in ***91x_lib.c*** file:

```
#ifdef _TIM0
TIM0 = (TIM_TypeDef *)TIM0_BASE;
#endif /*_TIM0 */

#ifdef _TIM1
TIM0 = (TIM_TypeDef *)TIM1_BASE;
#endif /*_TIM1 */

#ifdef _TIM2
TIM0 = (TIM_TypeDef *)TIM2_BASE;
#endif /*_TIM2 */
```

```
#ifdef _TIM3
TIM1 = (TIM_TypeDef *)TIM3_BASE;
#endif /*_TIM3 */
```

_TIM, _TIM0,_TIM1,_TIM2 and _TIM3 must be defined, in ***91x_conf.h*** file, to access the peripheral registers as follows:

```
#define _TIM
#define _TIM0
#define _TIM1
#define _TIM2
#define _TIM3

...
```

12.2 Software library functions

The following table enumerates the different functions of the TIM library.

Function Name	Description
TIM_DeInit	Deinitializes TIM peripheral registers to their default reset values.
TIM_Init	Initializes TIM peripheral according to the specified parameters in the TIM_InitTypeDef structure.
TIM_StructInit	Fills a TIM_InitTypeDef structure with the reset value of each parameter (which depend on the register reset value)
TIM_ClockSourceConfig	Configures the TIM clock source.
TIM_PrescalerConfig	Configures the TIM prescaler Value.
TIM_GetPrescalerValue	Gets the TIM prescaler Value.
TIM_GetICAP1Value	Reads and returns the Input Capture 1 value.
TIM_GetICAP2Value	Reads and returns the Input Capture 2 value.
TIM_GetPWMIPulse	Reads and returns the PWM input pulse value.
TIM_GetPWMIPeriod	Reads and returns the PWM input period value.
TIM_CounterCmd	Configures the TIM counter.
TIM_SetPulse	Set the new pulse value.
TIM_GetFlagStatus	Checks whether the specified TIM flag is set or not.
TIM_ClearFlag	Clears the specified TIM flag.
TIM_ITConfig	Configures the TIM interrupts.
TIM_GetCounterValue	Gets the TIM counter value.
TIM_DMAConfig	Configures the TIM DMA source.
TIM_DMACmd	Enables or disables the TIMx DMA interface.

12.2.1 TIM_DeInit

Function Name	TIM_DeInit
Function Prototype	`void TIM_DeInit(TIM_TypeDef *TIMx);`
Behavior Description	Deinitializes the TIMx peripheral registers to their default reset values.
Input Parameter	TIMx: where x can be 0,1, 2 or 3 to select the TIM peripheral.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	`SCU_APBPeriphReset()`

Example:

```
/*To deinitialize the TIM0 and TIM1*/
TIM_DeInit (TIM0);
TIM_DeInit (TIM1);
```

12.2.2 TIM_Init

Function Name	TIM_Init
Function Prototype	`void TIM_Init(TIM_TypeDef* TIMx, TIM_InitTypeDef* TIM_InitStruct)`
Behavior Description	Initializes the TIMx peripheral according to the specified parameters in the TIM_InitStruct .
Input Parameter1	TIMx: where x can be 0,1, 2 or 3 to select the TIM peripheral.
Input Parameter2	TIM_InitStruct: pointer to a TIM_InitTypeDef structure that contains the configuration information for the specified TIM peripheral. Refer to section "*TIM_InitTypeDef on page 131*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

TIM_InitTypeDef

The TIM_InitTypeDef structure is defined in the ***91x_tim.h*** file:

```
{
  u16 TIM_Mode;             /* Timer mode                                    */
  u16 TIM_OC1_Modes;        /* Output Compare 1 Mode: Timing or Wave         */
  u16 TIM_OC2_Modes;        /* Output Compare 2 Mode: Timing or Wave         */
  u16 TIM_Clock_Source;     /* Timer Clock source APB/SCU/EXTERNAL           */
  u16 TIM_Clock_Edge;       /* Timer Clock Edge: Rising or Falling Edge      */
  u16 TIM_OPM_INPUT_Edge;   /* Timer Input Capture 1 Edge used in OPM Mode   */
  u16 TIM_ICAP1_Edge;       /* Timer Input Capture 1 Edge used in ICAP1 Mode */
  u16 TIM_ICAP2_Edge;       /* Timer Input Capture 2 Edge used in ICAP2 Mode */
  u8  TIM_Prescaler;        /* Timer Prescaler factor                        */
  u16 TIM_Pulse_Level_1;    /* Level applied on the Output Compare Pin 1     */
  u16 TIM_Pulse_Level_2;    /* Level applied on the Output Compare Pin 2     */
  u16 TIM_Period_Level;     /* Level applied during the Period of a PWM Mode */
  u16 TIM_Pulse_Length_1;   /* Pulse 1 Length used in Output Compare 1 Mode  */
  u16 TIM_Pulse_Length_2;   /* Pulse 2 Length used in Output Compare 2 Mode  */
  u16 TIM_Full_Period;      /* Period Length used in PWM Mode                */
} TIM_InitTypeDef;
```

TIM_Mode

Specifies the TIM operating mode. This parameter can be one of the following values:

TIM_Mode	Meaning
TIM_PWM	Pulse Width Modulation mode
TIM_OPM	One Pulse Mode
TIM_PWMI	Pulse Width Modulation Input Mode
TIM_OCM_CHANNEL_1	Output Compare Channel 1 Mode
TIM_OCM_CHANNEL_2	Output Compare Channel 2 Mode
TIM_OCM_CHANNEL_12	Output Compare Channels 1 & 2 Mode
TIM_ICAP_CHANNEL_1	Input Capture Channel 1 Mode
TIM_ICAP_CHANNEL_2	Input Capture Channel 2 Mode
TIM_ICAP_CHANNEL_12	Input Capture Channel 1 & 2 Mode

*TIM_OC1_Mode*s

Specifies the operating mode of the OCMP1 pin. This member can be one of the following values:

TIM_OC1_Modes	Meaning
TIM_Timing	OCMP1 pin is a general I/O
TIM_Wave	OCMP1 pin is dedicated to the OC1 capability of the TIM

TIM_OC2_Modes

Specifies the operating mode of the OCMP2 pin. This member can be one of the following values:

TIM_OC2_Modes	Meaning
TIM_Timing	OCMP2 pin is a general I/O
TIM_Wave	OCMP2 pin is dedicated to the OC2 capability of the TIM

TIM_ICAP1_EDGE

Specifies the trigger mode for Input Capture 1. This member can be one of the following values:

TIM_ICAP1_EDGE	Meaning
TIM_ICAP1_EDGE_RISING	A rising edge triggers the capture
TIM_ICAP1_EDGE_FALLING	A falling edge triggers the capture

TIM_ICAP2_EDGE

Specifies the trigger mode for Input Capture 1. This member can be one of the following values:

TIM_ICAP2_EDGE	Meaning
TIM_ICAP2_EDGE_RISING	A rising edge triggers the capture
TIM_ICAP2_EDGE_FALLING	A falling edge triggers the capture

TIM_OPM_INPUT_Edge

Specifies the input edges for one pulse mode. This member can be one of the following values:

TIM_OPM_INPUT_Edge	Meaning
TIM_Rising	A rising edge triggers the counter initialization
TIM_Falling	A falling edge triggers the counter initialization

TIM_Clock_Source

Specifies the clock source of the TIM peripheral. This member can be one of the following values:

TIM_Clock_Source	Meaning
TIM_CLK_EXTERNAL	Reference clock source is used
TIM_CLK_APB	APB clock source is used
TIM_CLK_SCU	SCU clock source is used

TIM_Clock_Edge

Specifies which type of level transition on the reference clock will trigger the counter. This member can be one of the following values:

TIM_Clock_Edge	Meaning
TIM_CLK_EDGE_RISING	A rising edge triggers the counter
TIM_CLK_EDGE_FALLING	A falling edge triggers the counter

TIM_Prescaler

Specifies the Prescaler value to divide the APB clock. Timer clock will be equal to

$$\frac{f_{CPU}}{(Prescaler + 1)}$$

TIM_Pulse_Level_1

When using PWM, OPM, OCM1 or OCM12 mode, this parameter specifies the pulse level of the signal generated on the OCMP1 pin. This member can be one of the following values:

TIM_Pulse_Level_1	Meaning
TIM_High	OLVL1 is High Level
TIM_Low	OLVL2 is High Level

TIM_Pulse_Level_2

When using OCM2 or OCM12 mode, this parameter specifies the pulse level of the signal generated on OCMP2 pin. This member can be one of the following values:

TIM_Pulse_Level_2	Meaning
TIM_High	OLVL2 is High Level
TIM_Low	OLVL1 is High Level

TIM_Period_Level

When using OPM mode, this parameter specifies the signal level after the pulse generated on the OCMP1 pin. This member can be one of the following values:

TIM_Period_Level	Meaning
TIM_High	OLVL1 is High Level
TIM_Low	OLVL1 is Low Level

TIM_Pulse_Length_1

When using PWM, OPM, OCM1 or OCM1&2 mode, this parameter specifies the pulse length to be loaded in the OC1R register.

TIM_Pulse_Length_2

When using OCM1 or OCM12 mode, this parameter specifies the pulse length to be loaded in the OC2R register.

TIM_Full_Period

Specifies the period to be loaded in the OC2R register, when PWM mode is used.

Example:

```
/* To configure the TIM3 peripheral as Output Compare Mode on channel 2 */
TIM_InitStruct      TIM_InitStructure;
TIM_InitStructure.TIM_Mode = TIM_OCM_CHANNEL_2;
TIM_InitStructure.TIM_Clock_Source = TIM_CLK_APB;
TIM_InitStructure.TIM_Prescaler = 0xFF;
TIM_InitStructure.TIM_OC2_Modes = TIM_WAVE ;
TIM_InitStructure.TIM_Pulse_Level_2 = TIM_HIGH;
TIM_InitStructure.TIM_Pulse_Length_2 = 0xFF00;
TIM_Init (TIM3, &TIM_InitStructure);
/*To configure the TIM2 peripheral as Input Capture Mode on channel1*/

TIM_InitStruct      TIM_InitStructure;
TIM_InitStructure.TIM_Mode = TIM_ICAP_CHANNEL_1;
TIM_InitStructure.TIM_Clock_Source = TIM_CLK_APB;
TIM_InitStructure.TIM_Prescaler = 0xFF;
TIM_InitStructure.TIM_ICAP1_Edge = TIM_ICAP1_EDGE_RISING;

TIM_Init (TIM2, &TIM_InitStructure);
/*To configure the TIM0 peripheral in PWM Mode*/
TIM_InitStruct      TIM_InitStructure;
TIM_InitStructure.TIM_Mode = TIM_PWM;
TIM_InitStructure.TIM_Clock_Source = TIM_CLK_APB;
TIM_InitStructure.TIM_Prescaler = 0xFF;
TIM_InitStructure.TIM_Pulse_Level_1 = TIM_HIGH;
TIM_InitStructure.TIM_Period_Level = TIM_LOW;
TIM_InitStructure.TIM_Pulse_Length_1 = 0x3FF;
TIM_InitStructure.TIM_Full_Period = 0xFFF;
TIM_Init (TIM0, &TIM_InitStructure);

/*To configure the TIM1 peripheral in PWMI Mode*/
TIM_InitStruct      TIM_InitStructure;
TIM_InitStructure.TIM_Mode = TIM_PWMI;
TIM_InitStructure.TIM_Clock_Source = TIM_CLK_APB;
TIM_InitStructure.TIM_Prescaler = 0x7F;
TIM_InitStructure.TIM_ICAP1_Edge = TIM_ICAP1_EDGE_RISING;
TIM_Init (TIM1, &TIM_InitStructure);

/*To configure the TIM2 peripheral in OPM Mode*/
TIM_InitStruct      TIM_InitStructure;
TIM_InitStructure.TIM_Mode = TIM_OPM;
TIM_InitStructure.TIM_OPM_INPUT_Edge = TIM_OPM_EDGE_RISING;
TIM_InitStructure.TIM_Clock_Source = TIM_CLK_APB;
TIM_InitStructure.TIM_Prescaler = 0xFF;
TIM_InitStructure.TIM_Pulse_Level_1 = TIM_HIGH;
TIM_InitStructure.TIM_Period_Level = TIM_LOW;
TIM_InitStructure.TIM_Pulse_Length_1 = 0xFFF;
TIM_Init (TIM2, &TIM_InitStructure);
```

12.2.3 TIM_StructInit

Function Name	TIM_StructInit
Function Prototype	`void TIM_StructInit(TIM_InitTypeDef* TIM_InitStruct)`
Behavior Description	Fills each TIM_InitStruct member with its reset value.
Input Parameter	TIM_InitStruct: pointer to a TIM_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

Example:

```
/* To initialize the TIM structure */
TIM_StructInit (&TIM_InitStruct);
```

12.2.4 TIM_PrescalerConfig

Function Name	TIM_PrescalerConfig
Function Prototype	`void TIM_PrescalerConfig (TIM_TypeDef *TIMx, u8 TIM_Prescaler);`
Behavior Description	This routine is used to configure the TIM prescaler value.
Input Parameter1	TIMx: where x can be 0,1, 2 or 3 to select the TIM peripheral.
Input Parameter2	TIM_Prescaler: specifies the TIM prescaler value.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

Example:

```
/*To configure the TIM3 prescaler value to 0x7F*/
TIM_PrescalerConfig(TIM3, 0x7F)
```

12.2.5 TIM_GetPrescalerValue

Function Name	TIM_GetPrescalerValue
Function Prototype	`u8 TIM_GetPrescalerValue (TIM_TypeDef *TIMx)`
Behavior Description	This routine is used to get the Prescaler value.
Input Parameter	TIMx: where x can be 0,1, 2 or 3 to select the TIM peripheral.
Output Parameter	None
Return Parameter	The prescaler value.
Required preconditions	...
Called functions	None

Example:

```
/*To get the Prescaler value of the TIM2 timer*/
u8 MyPrescaler;
MyPrescaler = TIM_GetPrescalerValue(TIM2);
```

12.2.6 TIM_GetICAP1Value

Function Name	TIM_GetICAP1Value
Function Prototype	`u8 TIM_GetICAPAValue (TIM_TypeDef *TIMx)`
Behavior Description	This routine is used to get the input capture 1 value.
Input Parameter	TIMx: where x can be 0,1, 2 or 3 to select the TIM peripheral.
Output Parameter	None
Return Parameter	The input capture 1 value.
Required preconditions	...
Called functions	None

Example:

```
/*To get the input capture 1value of the TIM2 timer*/
u16 MyValue;
MyValue = TIM_GetICAP1Value(TIM2);
```

12.2.7 TIM_GetICAP2Value

Function Name	TIM_GetICAP2Value
Function Prototype	`u8 TIM_GetICAP2Value (TIM_TypeDef *TIMx)`
Behavior Description	This routine is used to get the input capture 2 value.
Input Parameter	TIMx: where x can be 0,1, 2 or 3 to select the TIM peripheral.
Output Parameter	None
Return Parameter	The input capture 2 value.
Required preconditions	...
Called functions	None

Example:

```
/*To get the input capture 2 value of the TIM2 timer*/
u16 MyValue;
MyValue = TIM_GetICAP2Value(TIM2);
```

12.2.8 TIM_GetPWMIPulse

Function Name	TIM_GetPWMIPulse
Function Prototype	`u16 TIM_GetPWMIPulse (TIM_TypeDef *TIMx);`
Behavior Description	This routine is used to get the PWM input pulse value.
Input Parameter	TIMx: where x can be 0,1, 2 or 3 to select the TIM peripheral.
Output Parameter	None
Return Parameter	The pulse of the external signal.
Required preconditions	...
Called functions	None

Example:

```
/*To get the pulse of PWM input signal of the TIM3 timer*/
u16 MyPWMIPulse;
MyPWMIPulse = TIM_GetPWMIPulse(TIM3);
```

12.2.9 TIM_GetPWMIPeriod

Function Name	TIM_GetPWMIPeriod
Function Prototype	`u16 TIM_GetPWMIPeriod (TIM_TypeDef *TIMx);`
Behavior Description	This routine is used to get the PWM input period value.
Input Parameter	TIMx: where x can be 0,1, 2 or 3 to select the TIM peripheral.
Output Parameter	None
Return Parameter	The period of the external signal.
Required preconditions	...
Called functions	None

Example:

```
/*To get the period of the PWM input signal of the TIM3 timer*/
u16 MyPWMIPeriod;
MyPWMIPeriod = TIM_GetPWMIPeriod(TIM3);
```

12.2.10 TIM_CounterCmd

Function Name	TIM_CounterCmd
Function Prototype	`void TIM_CounterCmd (TIM_TypeDef *TIMx, CounterOperations TIM_Operation);`
Behavior Description	This routine is used to control the TIM counter.
Input Parameter1	TIMx: where x can be 0,1, 2 or 3 to select the TIM peripheral.
Input Parameter2	TIM_Operation: specifies the operation of the counter. - TIM_STOP: Stops the TIM counter. - TIM_ENABLE: Enable or resume the TIM counter. - TIM_CLEAR: Clear the TIM counter value.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

Example:

```
/*To stop the TIM2 timer counter*/
TIM_CounterCmd(TIM2, TIM_STOP);
```

12.2.11 TIM_SetPulse

Function Name	TIM_SetPulse
Function Prototype	`void TIM_SetPulse(TIM_TypeDef *TIMx, u16 TIM_Channel, u16 TIM_Pulse);`
Behavior Description	This routine is used to set the new pulse value.
Input Parameter1	TIMx: where x can be 0,1, 2 or 3 to select the TIM peripheral.
Input Parameter2	TIM_Channel: specifies the channel. This parameter can be one of the following: - TIM_PWM_OC1_Channel: PWM/Output Compare 1 Channel - TIM_OC2_Channel: Output Compare 2 Channel.
Input Parameter3	TIM_Pulse: specifies the new TIM pulse value.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

Example:

```
/*To set the pulse value of TIM2 on the channel 2 to 0xFFAA*/
TIM_SetPulse(TIM2, TIM_OC2_Channel,0xFFAA);
```

12.2.12 TIM_GetFlagStatus

Function Name	TIM_GetFlagStatus
Function Prototype	`FlagStatus TIM_GetFlagStatus(TIM_TypeDef* TIMx, u16 TIM_FLAG)`
Behavior Description	Checks whether the specified TIM flag is set or not.
Input Parameter1	TIMx: where x can be 0,1, 2 or 3 to select the TIM peripheral.
Input Parameter2	TIM_FLAG: specifies the flag to check. Refer to section "*TIM_FLAG on page 140*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of TIM_FLAG (SET or RESET).
Required preconditions	...
Called functions	None

TIM_FLAG

TIM_FLAG	Meaning
TIM_FLAG_IC1	Input Capture Flag channel 1
TIM_FLAG_OC1	Output Compare Flag channel 1
TIM_FLAG_TO	Timer Overflow
TIM_FLAG_IC2	Input Capture Flag channel 2
TIM_FLAG_OC2	Output Compare Flag channel 2

Example:

```
/*To check the TIM1 overflow flag*/
FunctionalState OverFlowStatus ;
OverFlowStatus = TIM_GetFlagStatus(TIM1, TIM_FLAG_TO) ;
```

12.2.13 TIM_ClearFlag

Function Name	TIM_ClearFlag
Function Prototype	`void TIM_ClearFlag(TIM_TypeDef* TIMx, u16 TIM_FLAG)`
Behavior Description	Clears the TIMx pending flags.
Input Parameter1	TIMx: where x can be 0,1, 2 or 3 to select the TIM peripheral.
Input Parameter2	TIM_FLAG: specifies the flag to clear. Refer to section "*TIM_FLAG on page 140*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

TIM_FLAG

To clear TIM flags, use a combination of one or more of the following values:

TIM_FLAG	Meaning
TIM_FLAG_IC1	Input Capture Flag channel 1
TIM_FLAG_OC1	Output Compare Flag channel 1
TIM_FLAG_TO	Timer Overflow
TIM_FLAG_IC2	Input Capture Flag channel 2
TIM_FLAG_OC2	Output Compare Flag channel 2

Example:

```
/*To clear the TIM1 overflow flag*/
TIM_ClearFlag(TIM1, TIM_FLAG_TO);
```

12.2.14 TIM_ITConfig

Function Name	TIM_ITConfig
Function Prototype	`void TIM_ITConfig(TIM_TypeDef* TIMx, u16 TIM_IT, FunctionalState TIM_NewState)`
Behavior Description	Enables or disables the specified TIM interrupts.
Input Parameter1	TIMx: where x can be 0,1, 2 or 3 to select the TIM peripheral.
Input Parameter2	TIM_IT: specifies the TIM interrupts sources to be enabled or disabled. Refer to section "*TIM_IT on page 141*" for more details on the allowed values of this parameter.
Input Parameter3	TIM_NewState: new state of the specified TIMx interrupts. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

TIM_IT

To enable or disable TIM interrupts, use a combination of one or more of the following values:

TIM_IT	Meaning
TIM_IT_IC1	Input Capture IT channel 1
TIM_IT_OC1	Output Compare IT channel 1
TIM_IT_TO	Timer Overflow IT
TIM_IT_IC2	Input Capture IT channel 2
TIM_IT_OC2	Output Compare IT channel 2

Example:

```
/*To enable the overflow IT and the input capture 2 of the TIM3*/
TIM_ITConfig(TIM3, TIM_IT_TO|TIM_IT_IC2, ENABLE);
```

12.2.15 TIM_GetCounterValue

Function Name	TIM_GetCounterValue
Function Prototype	`u16 TIM_GetCounterValue(TIM_TypeDef* TIMx)`
Behavior Description	Gets the TIM counter value.
Input Parameter	TIMx: where x can be 0,1, 2 or 3 to select the TIM peripheral.
Output Parameter	None
Return Parameter	The TIM counter value
Required preconditions	...
Called functions	None

Example:

```
/*To get the counter value of the TIM2*/
u16 MyCounter;
MyCounter = TIM_GetCounterValue(TIM2);
```

12.2.16 TIM_DMAConfig

Function Name	TIM_DMAConfig
Function Prototype	`void TIM_DMAConfig (TIM_TypeDef *TIMx, u16 TIM_DMA_Souces);`
Behavior Description	This routine is used to enable the DMA.
Input Parameter1	TIMx: where x can be 0,1, 2 or 3 to select the TIM peripheral.
Input Parameter2	TIM_DMASources: specifies the DMA source to be used. Refer to the section "*TIM_DMASources on page 142*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

TIM_DMASources

The TIM DMA sources that can be selected are listed in the following table.

TIM_DMASources	Meaning
TIM_DMA_IC1	Input Capture DMA channel 1
TIM_DMA_OC1	Output Compare DMA channel 1
TIM_DMA_IC2	Input Capture DMA channel 2
TIM_DMA_OC2	Output Compare DMA channel 2

Example:

```
/*To configure the DMA for the TIM0 peripheral and to choose ICAP1 as a DMA source*/
TIM_DMAConfig(TIM0, TIM_DMA_ICAP1);
```

12.2.17 TIM_DMACmd

Function Name	TIM_DMACmd
Function Prototype	`void TIM_DMACmd(TIM_TypeDef* TIMx, FunctionalState TIM_NewState)`
Behavior Description	Enables or disables the TIMx DMA interface.
Input Parameter1	TIMx: where x can be 0,1, 2 or 3 to select the TIM peripheral.
Input Parameter2	TIM_NewState: new state of the DMA interface. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

Example:

```
/*To enable TIM0 DMA interface*/
TIM_DMACmd(TIM0, ENABLE);
/*To disable TIM0 DMA interface*/
TIM_DMACmd(TIM0, DISABLE);
```

13 DMA Controller (DMA)

The DMA enables memory-to-memory, memory-to-peripheral, peripheral-to-memory, and peripheral-to-peripheral transactions. Each DMA stream provides unidirectional serial DMA transfers for a single source and destination. A bidirectional port requires one stream for transmit and one for receive. The source and destination areas can each be either a memory region or a peripheral.

13.1 DMA Register structure

The DMA register structures *DMA_TypeDef* and *DMA_Channel_TypeDef* are defined in the ***91x_map.h*** file as follows:

```
typedef struct
{
  vu32 DMA_SRC;         /*Channelx Source Address Register     */
  vu32 DMA_DES;         /*Channelx Destination Address Register*/
  vu32 DMA_LLI;         /* Channelx Lincked List Item Register */
  vu32 DMA_CC;         /*Channelx Contol Register              */
  vu32 DMA_CCNF;       /*Channelx Configuration Register       */
} DMA_Channel_TypeDef;

/* x can be ,0,1,2,3,4,5,6 or 7. There are eight Channels AHB BUS Master */
typedef struct
{

  vu32 DMA_ISR;             /*Interrupt Status Register                    */
  vu32 DMA_TCISR;        /*Terminal Count Interrupt Status Register      */
  vu32 DMA_TCICR;       /* Terminal CountInterrupt Clear Register       */
  vu32 DMA_EISR;        /* Error Interrupt Status Register              */
  vu32 DMA_EICR;        /* Error Interrupt Clear Register               */
  vu32 DMA_TCRISR;      /* Terminal Count Raw Interrupt Status Register */
  vu32 DMA_ERISR;       /* Raw Error Interrupt Status Register          */
  vu32 DMA_ENCSR;       /* Enabled Channel Status Register              */
  vu32 DMA_SBRR;        /* Software Burst Request Register              */
  vu32 DMA_SSRR;        /* Software Single Request Register             */
  vu32 DMA_SLBRR;       /* Software Last Burst Request Register         */
  vu32 DMA_SLSRR;       /* Software Last Single Request Register        */
  vu32 DMA_CNFR;        /* Configuration Register                       */
  vu32 DMA_SYNR;        /* Syncronization Register                      */
} DMA_TypeDef;


/* there are forteen commun registers for the eight channels*/
```

The following table presents all the DMA registers:

Register	Description
DMA_SRCx	Channelx Source Address Register
DMA_DESx	Channelx Destination Address Register
DMA_LLIx	Channelx Linked List Item Register
DMA_CCx	Channelx Contol Register
DMA_CCNFx	Channelx Configuration Register
DMA_ISR	Interrupt Status Register
DMA_TCISR	Terminal Count Interrupt Status Register
DMA_TCICR	Terminal Count Interrupt Clear Register
DMA_EISR	Error Interrupt Status Register
DMA_EICR	Error Interrupt Clear Register
DMA_TCRISR	Terminal Count Raw Interrupt Status Register
DMA_ERISR	Error Raw Interrupt Status Register
DMA_ENCSR	Enabled Channel Status Register
DMA_SBRR	Software Burst Request Register
DMA_SSRR	Software Single Request Register
DMA_SLBRR	Software Last Burst Request Register
DMA_SLSRR	Software Last Single Request Register
DMA_CNFR	Configuration Register
DMA_SYNR	Syncronization Register

The DMA interface is declared in the same file:

```
#ifndef EXT
  #define EXT extern
#endif /* EXT */

#define AHB_DMA_U          (0x78000000) /* DMA UnBuffered Space */
#define AHB_DMA_B          (0x68000000) /* DMA Buffered Space   */

#define AHB_DMA_Channel0_OFST    (0x00000100)   /* Offset of Channel 0 */
#define AHB_DMA_Channel1_OFST    (0x00000120)   /* Offset of Channel 1 */
#define AHB_DMA_Channel2_OFST    (0x00000140)   /* Offset of Channel 2 */
#define AHB_DMA_Channel3_OFST    (0x00000160)   /* Offset of Channel 3 */
#define AHB_DMA_Channel4_OFST    (0x00000180)   /* Offset of Channel 4 */
#define AHB_DMA_Channel5_OFST    (0x000001A0)   /* Offset of Channel 5 */
#define AHB_DMA_Channel6_OFST    (0x000001C0)   /* Offset of Channel 6 */
#define AHB_DMA_Channel7_OFST    (0x000001E0)   /* Offset of Channel 7 */



...
#ifndef Buffered
#define DMA_BASE           (AHB_DMA_U)
```

```
...
#else /* Buffered */

#define DMA_BASE           (AHB_DMA_B)
...
#endif /* Buffered */

/* Channel Base Address definition*/
#define DMA_Channel0_BASE  (DMA_BASE + AHB_DMA_Channel0_OFST)
#define DMA_Channel1_BASE  (DMA_BASE + AHB_DMA_Channel1_OFST)
#define DMA_Channel2_BASE  (DMA_BASE + AHB_DMA_Channel2_OFST)
#define DMA_Channel3_BASE  (DMA_BASE + AHB_DMA_Channel3_OFST)
#define DMA_Channel4_BASE  (DMA_BASE + AHB_DMA_Channel4_OFST)
#define DMA_Channel5_BASE  (DMA_BASE + AHB_DMA_Channel5_OFST)
#define DMA_Channel6_BASE  (DMA_BASE + AHB_DMA_Channel6_OFST)
#define DMA_Channel7_BASE  (DMA_BASE + AHB_DMA_Channel7_OFST)
....

#ifndef DEBUG
....

/* DMA peripheral declaration*/

#define DMA                ((DMA_TypeDef *)DMA_BASE)
#define DMA_Channel0       ((DMA_Channel_TypeDef *)DMA_Channel0_BASE)
#define DMA_Channel1       ((DMA_Channel_TypeDef *)DMA_Channel1_BASE)
#define DMA_Channel2       ((DMA_Channel_TypeDef *)DMA_Channel2_BASE)
#define DMA_Channel3       ((DMA_Channel_TypeDef *)DMA_Channel3_BASE)
#define DMA_Channel4       ((DMA_Channel_TypeDef *)DMA_Channel4_BASE)
#define DMA_Channel5       ((DMA_Channel_TypeDef *)DMA_Channel5_BASE)
#define DMA_Channel6       ((DMA_Channel_TypeDef *)DMA_Channel6_BASE)
#define DMA_Channel7       ((DMA_Channel_TypeDef *)DMA_Channel7_BASE)

#else   /* DEBUG */
...
#ifdef _DMA
EXT DMA_TypeDef            *DMA;
#endif /* _DMA */

#ifdef _DMA_Channel0
EXT DMA_Channel_TypeDef    *DMA_Channel0;
#endif /* _DMA_Channel0 */

#ifdef _DMA_Channel1
EXT DMA_Channel_TypeDef    *DMA_Channel1;
#endif /* _DMA_Channel1 */

#ifdef _DMA_Channel2
EXT DMA_Channel_TypeDef    *DMA_Channel2;
#endif /* _DMA_Channel0 */

#ifdef _DMA_Channel3
EXT DMA_Channel_TypeDef    *DMA_Channel3;
#endif /* _DMA_Channel0 */

#ifdef _DMA_Channel4
EXT DMA_Channel_TypeDef    *DMA_Channel4;
#endif /* _DMA_Channel4 */

#ifdef _DMA_Channel5
EXT DMA_Channel_TypeDef    *DMA_Channel5;
#endif /* _DMA_Channel5 */
```

```
#ifdef _DMA_Channel6
EXT DMA_Channel_TypeDef    *DMA_Channel6;
#endif /* _DMA_Channel6 */

#ifdef _DMA_Channel7
EXT DMA_Channel_TypeDef    *DMA_Channel7;
#endif /* _DMA_Channel7 */
```

When debug mode is used, DMA pointers are initialized in the ***91x_lib.c*** file:

```
#ifdef _DMA
  DMA = (DMA_TypeDef *)DMA_BASE;
#endif /* _DMA */

#ifdef _DMA_Channel0
  DMA_Channel0= (DMA_Channel_TypeDef *)DMA_Channel0_BASE;
#endif /* _DMA_Channel0 */

#ifdef _DMA_Channel1
  DMA_Channel1=       (DMA_Channel_TypeDef *)DMA_Channel1_BASE;
#endif /* _DMA_Channel1 */

#ifdef _DMA_Channel2
  DMA_Channel2 =      (DMA_Channel_TypeDef *)DMA_Channel2_BASE;
#endif /* _DMA_Channel2 */

#ifdef _DMA_Channel3
  DMA_Channel3 =       (DMA_Channel_TypeDef *)DMA_Channel3_BASE;
#endif /* _DMA_Channel3 */

#ifdef _DMA_Channel4
 DMA_Channel4 =      (DMA_Channel_TypeDef *)DMA_Channel4_BASE;
#endif /* _DMA_Channel4 */

#ifdef _DMA_Channel5
 DMA_Channel5=       (DMA_Channel_TypeDef *)DMA_Channel5_BASE;
#endif /* _DMA_Channel5*/


#ifdef _DMA_Channel6
 DMA_Channel6 =      (DMA_Channel_TypeDef *)DMA_Channel6_BASE;
#endif /* _DMA_Channel6 */

#ifdef _DMA_Channel7
 DMA_Channel7 =      (DMA_Channel_TypeDef *)DMA_Channel7_BASE;
#endif /* _DMA_Channel7 */
```

_DMA,_DMA_channel0,_DMA_channel1,_DMA_channel2, ..._DMA_channel7 must be defined, in ***91x_conf.h*** file, to access the peripheral registers as follows:

```
#define _DMA
#define _DMA_Channel0
#define _DMA_Channel1
#define _DMA_Channel2
#define _DMA_Channel3
#define _DMA_Channel4
#define _DMA_Channel5
#define _DMA_Channel6
#define _DMA_Channel7

...
```

13.2 Software library functions

The following table enumerates the different functions of the DMA library.

Function Name	Description
DMA_DeInit	Initializes the DMA peripheral registers to their default reset values.
DMA_Init	Initializes the DMA Channelx according to the specified parameters in the DMA_InitStruct.
DMA_StructInit	Fills each DMA_InitStruct member with its reset value.
DMA_Cmd	Enables or disables the DMA peripheral.
DMA_ITMaskConfig	Enables or disables the specified DMA Mask interrupt.
DMA_ITConfig	Enables or disables the Terminal count interrupt for specified DMA channelx.
DMA_SRCIncConfig	Enables or disables the source Address incrementing for the specified DMA channelx.
DMA_DESIncConfig	Enables or disables the destination Address incrementing for the specified DMA channelx.
DMA_GetITStatus	Checks the status of the specified interrupt.
DMA_ClearIT	Clears the pending bit of the specified interrupt.
DMA_SyncConfig	Enables or disables the synchronization logic for the corresponding DMA Request Signal.
DMA_GetSReq	Checks for a specific source if it requests a Single transfer or not.
DMA_GetLSReq	Checks for a specific source if it requests a Last Single transfer or not.
DMA_GetBReq	Checks for a specific source if it requests a Burst transfer or not.
DMA_GetLBReq	Checks for a specific source if it requests a Last Burst transfer or not.
DMA_SetSReq	Sets the DMA to generate a Single transfer request for the corresponding DMA Request Source.
DMA_SetLSReq	Sets the DMA to generate a Last Single transfer request for the corresponding DMA Request Source.
DMA_SetBReq	Sets the DMA to generate a Burst transfer request for the corresponding DMA Request Source.
DMA_SetLBReq	Sets the DMA to generate a Last Burst transfer request for the corresponding DMA Request Source.
DMA_ChannelCmd	Enables or disables the specified DMA Channelx
DMA_ChannelHalt	Enables DMA requests or ignore extra source DMA requests for the specified channelx.
DMA_ChannelBuffering	Enables or disables the access Cache ability for the specified channelx.
DMA_ChannelMode	Enables the access in User or Privileged mode for the specified channelx.

Function Name	Description
DMA_ChannelLockTrsf	Enables or disables locked transfers.
DMA_ChannelCache	Enables or disables the access Cache ability for the specified channelx.
DMA_GetChannelActiveStatus	Checks if the DMA_Channelx FIFO is empty or not. Returns SET while the corresponding channel FIFO is not empty.
DMA_GetChannelStatus	Checks the status of DMA channelx (Enabled or Disabled)

13.2.1 DMA_DeInit

Function Name	DMA_DeInit
Function Prototype	`void DMA_DeInit()`
Behavior Description	Deinitializes the DMA peripheral registers to their default reset values.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	SCU_AHBPeriphReset()

Example:

```
...
DMA_DeInit(); /* set all DMA Peripheral registers to their reset value */
...
```

13.2.2 DMA_Init

Function Name	DMA_Init
Function Prototype	`void DMA_Init(DMA_Channel_TypeDef * DMA_Channelx, DMA_InitTypeDef * DMA_InitStruct);`
Behavior Description	Initializes the DMA_Channelx according to the specified parameters in the DMA_InitStruct.
Input Parameter1	DMA_Channelx: where x can be 0,1,2,3,4,5,6,or 7 to select the DMA Channel.
Input Parameter2	DMA_InitStruct: pointer to a *DMA_InitTypeDef* structure which contains the configuration information for the specified DMA_Channelx. Refer to section "*DMA_InitTypeDef: on page 150*" for more details.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

DMA_InitTypeDef:

The *DMA_InitTypeDef* structure is defined in the ***91x_dma.h*** file:

```
typedef struct
{
u32 DMA_Channel_SrcAdd;
u32 DMA_Channel_DesAdd;
u32 DMA_Channel_LLstItm;
u8 DMA_Channel_DesWidth;
u8 DMA_Channel_SrcWidth;
u8 DMA_Channel_DesBstSize;
u8 DMA_Channel_SrcBstSize;
u16 DMA_Channel_TrsfSize;
u8 DMA_Channel_FlowCntrl;
u8 DMA_Channel_Src;
u8 DMA_Channel_Des;

} DMA_InitTypeDef;
```

DMA_Channel_SrcAdd

The current source address, (byte-aligned), of the data to be transferred.

DMA_Channel_DesAdd

The current destination address, (byte-aligned), of the data to be transferred.

DMA_Channel_LLstItm

The word- aligned address for the next Linked List Item.

DMA_Channel_DesWidth

Destination transfer width.

This member can be one of the following values:

DMA_Channel_DesWidth	Value	Meaning
DMA_DesWidth_Byte	0x00000000	Destination Width is one Byte
DMA_DesWidth_HalfWord	0x00200000	Destination Width is one half word
DMA_DesWidth_Word	0x00400000	Destination Width is one Word

DMA_Channel_SrcWidth

Source transfer width.

This member can be one of the following values:

DMA_Channel_SrcWidth	Value	Meaning
DMA_SrcWidth_Byte	0x00000000	Source width is one Byte
DMA_SrcWidth_HalfWord	0x00040000	Source width is one Half Word
DMA_SrcWidth_Word	0x00080000	Source width is one Word

DMA_Channel_DesBstSize

The destination burst size, which indicates the number of transfers that make up a destination burst transfer request.

This member can be one of the following values:

DMA_Channel_DesBstSize	Value	Meaning
DMA_DesBst_1Data	0x00000000	Destination Burst transfer request is 1 Data (DATA = destination transfer width)
DMA_DesBst_4Data	0x00008000	Destination Burst transfer request is 1 Data
DMA_Bst_8Data	0x00010000	Destination Burst transfer request is 4 Data
DMA_DesBst_16Data	0x00018000	Destination Burst transfer request is 8 Data
DMA_DesBst_32Data	0x00020000	Destination Burst transfer request is 16 Data
DMA_DesBst_64Data	0x00028000	Destination Burst transfer request is 32 Data
DMA_DesBst_128Data	0x00030000	Destination Burst transfer request is 128 Data
DMA_DesBst_256Data	0x00038000	Destination Burst transfer request is 256 Data

DMA_Channel_SrcBstSize

The source burst size indicates the number of transfers that make up a source burst.

This member can be one of the following values:

DMA_Channel_SrcBstSize	Value	Meaning
DMA_SrcBst_1Data	0x00000000	Source Burst transfer request is 1 Data (DATA = Source transfer width)
DMA_SrcBst_4Data	0x00001000	Source Burst transfer request is 4 Data
DMA_SrcBst_8Data	0x00002000	Source Burst transfer request is 8 Data
DMA_SrcBst_16Data	0x00003000	Source Burst transfer request is 16 Data
DMA_SrcBst_32Data	0x00004000	Source Burst transfer request is 32 Data
DMA_SrcBst_64Data	0x00005000	Source Burst transfer request is 64Data
DMA_SrcBst_128Data	0x00006000	Source Burst transfer request is 128 Data
DMA_SrcBst_256Data	0x00007000	Source Burst transfer request is 256 Data

DMA_Channel_TrsfSize

Transfer size indicates the size of the transfer when the DMA controller is the flow controller.

DMA_Channel_FlowCntrl

Flow control and transfer type: This value indicates the flow controller and transfer type. The flow controller can be the DMA, the source peripheral, or the destination peripheral. The transfer type can be memory to-memory, memory-to-peripheral, peripheral-to-memory, or peripheral to-peripheral.

This member can be one of the following values:

DMA_Channel_FlowCntrl	Value	Transfer Type	The flow controller
DMA_FlowCntrlt0_DMA	0x00000000	Memory-to-memory	DMA
DMA_FlowCntrl1_DMA	0x00000800	Memory-to-peripheral	DMA
DMA_FlowCntrl2_DMA	0x00001000	Peripheral-to-memory	DMA
DMA_FlowCntrl3_DMA	0x00001800	Source peripheral-to-destination peripheral	DMA
DMA_FlowCntrl_DestPerip	0x00002000	Source peripheral-to-destination peripheral	Destination peripheral
DMA_FlowCntrl_Perip1	0x00002800	Memory-to-peripheral	peripheral
DMA_FlowCntrl_Perip2	0x00003000	Peripheral-to-memory	peripheral
DMA_FlowCntrl_SrcPerip	0x00003800	Source peripheral-to-destination peripheral	Source peripheral

DMA_Channel_Src

Source peripheral. This value selects the DMA source request peripheral.

This field is ignored if the source of the transfer is from memory.

DMA_Channel_Src	Value	Description
DMA_SRC_USB_RX	0X00	
DMA_SRC_USB_TX	0x02	
DMA_SRC_TIM0	0x04	
DMA_SRC_TIM1	0x06	
DMA_SRC_UART0_RX	0x08	
DMA_SRC_UART0_TX	0x0A	
DMA_SRC_UART1_RX	0x0C	2 out of 3 UARTs have DMA support (UART0 is not supported by DMA)
DMA_SRC_UART1_TX	0x0E	
DMA_SRC_External_Req0	0x10	
DMA_SRC_External_Req1	0x12	
DMA_SRC_I2C0	0x14	
DMA_SRC_I2C1	0x16	
DMA_SRC_SSP0_RX	0x18	
DMA_SRC_SSP0_TX	0x1A	
DMA_SRC_SSP1_RX	0x1C	
DMA_SRC_SSP1_TX	0x1E	

DMA_Channel_Des

Destination peripheral. This value selects the DMA destination request peripheral. This field is ignored if the destination of the transfer is to memory.

DMA_Channel_Des	Value	Description
DMA_DES_USB_RX	0X00	
DMA_DES_USB_TX	0x40	
DMA_DES_TIM0	0x80	
DMA_DES_TIM1	0xC0	
DMA_DES_UART0_RX	0x100	
DMA_DES_UART0_TX	0x140	
DMA_DES_UART1_RX	0x180	2 out of 3 UARTs have DMA support (UART0 is not supported by DMA)
DMA_DES_UART1_TX	0x1C0	
DMA_DES_External_Req0	0x200	
DMA_DES_External_Req1	0x240	
DMA_DES_I2C0	0x280	
DMA_DES_I2C1	0x2C0	
DMA_DES_SSP0_RX	0x300	
DMA_DES_SSP0_TX	0x340	
DMA_DES_SSP1_RX	0x380	
DMA_DES_SSP1_TX	0x3C0	

Example:

```
/* Initialize the DMA Channel0 according to the DMA_InitStruct members */
DMA_InitTypeDef  DMA_InitStruct;
...
DMA_InitStruct.DMA_Channel_SrcAdd= 0x4000000;
DMA_InitStruct. DMA_Channel_DesAdd= 0x4002000;
DMA_InitStruct.DMA_Channel_LLstItm =0;
DMA_InitStruct.DMA_Channel_DesWidth = DMA_Width_HalfWord ;
DMA_InitStruct.DMA_Channel_DestBstSize = DMA_DesBst_4Data ;
DMA_InitStruct.DMA_Channel_SrcBstSize= DMA_SrcBst_1Data;
DMA_InitStruct.DMA_Channel_TrsfSize = 0x14;
DMA_InitStruct.DMA_Channel_FlowCntrl= DMA_FlowCntrl_DMA;
...
DMA_Init(DMA_Channel0,&DMA_InitStruct);
```

13.2.3 DMA_StructInit

Function Name	DMA_StructInit
Function Prototype	`void DMA_StructInit(DMA_InitTypeDef* DMA_InitStruct)`
Behavior Description	Fills each DMA_InitStruct member with its reset value.
Input Parameter	DMA_InitStruct: pointer to a DMA_InitTypeDef structure.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
...
/* define and Initialize a DMA_InitStruct */
DMA_InitTypeDef  DMA_InitStruct;
DMA_StructInit(&DMA_InitStruct);
...
```

13.2.4 DMA_Cmd

Function Name	DMA_Cmd
Function Prototype	`void DMA_Cmd(FunctionalState NewState)`
Behavior Description	Enables or disables the DMA peripheral.
Input Parameter1	NewState: new state of the DMA peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

Example:

```
/* Enable DMA */
...
DMA_Cmd(ENABLE);
...
```

13.2.5 DMA_ITMaskConfig

Function Name	DMA_ITMaskConfig
Function Prototype	`void DMA_ITMaskConfig(DMA_Channel_TypeDef * DMA_Channelx, u16 DMA_ITMask, FunctionalState NewState)`
Behavior Description	Enables or disables the specified DMA Mask interrupt.
Input Parameter1	DMA_Channelx: where x can be 0,1,2,3,4,5,6,or 7 to select the DMA Channel.
Input Parameter2	DMA_ITMask: specifies the DMA interrupt mask source to be enabled or disabled. Refer to section "*DMA_ITMask on page 156*" for more details on the allowed values of this parameter.
Input Parameter3	NewState: new state of the specified DMA_Channelx mask interrupt. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

DMA_ITMask

To enable or disable DMA_Channelx Mask interrupts.

DMA_ITMask	Value	Meaning
DMA_ITMask_IE	0x4000	Interrupt error mask.
DMA_ITMask_ITC	0x8000	Terminal count interrupt mask.
DMA_ITMask_ALL	0xC000	All DMA_Channelx interrupts enable/disable mask

Example:

```
/* Terminal count interrupt Mask Enable for DMA_Channel2 */
...
DMA_ITMaskConfig(DMA_Channel2, DMA_ITMask_ITC, ENABLE);
...
```

13.2.6 DMA_ChannelSRCIncConfig

Function Name	DMA_ChannelSRCIncConfig
Function Prototype	`void DMA_ChannelSRCIncConfig (DMA_Channel_TypeDef * DMA_Channelx, FunctionalState NewState);`
Behavior Description	Enables or disables the source Address incrementing for the specified DMA channelx.
Input Parameter1	DMA_Channelx: where x can be 0,1,2,3,4,5,6,or 7 to select the DMA Channel.
Input Parameter2	NewState: new state of the source incrementing feature. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

Example:

```
/* Enable the source incrementation feature for the channel0 */
...
DMA_ChannelSRCIncConfig (DMA_Channel0, ENABLE);
...
```

13.2.7 DMA_ChannelDESIncConfig

Function Name	DMA_ChannelDESIncConfig
Function Prototype	`void DMA_ChannelDESIncConfig (DMA_Channel_TypeDef * DMA_Channelx, FunctionalState NewState);`
Behavior Description	Enables or disables the source Address incrementing for the specified DMA channelx.
Input Parameter1	DMA_Channelx: where x can be 0,1,2,3,4,5,6,or 7 to select the DMA Channel.
Input Parameter2	NewState: new state of the destination incrementing feature. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

Example:

```
/* Enable the destination incrementation feature for the channel0 */
...
DMA_ChannelDESIncConfig (DMA_Channel0, ENABLE);
...
```

13.2.8 DMA_GetChannelActiveStatus

Function Name	DMA_ITConfig
Function Prototype	`FlagStatus DMA_GetChannelActiveStatus(DMA_Channel_TypeDef * DMA_Channelx)`
Behavior Description	Checks the DMA_Channelx FIFO if it is empty or not. Returns SET while the corresponding channel FIFO is not empty.
Input Parameter1	DMA_Channelx: where x can be 0,1,2,3,4,5,6,or 7 to select the DMA Channel.
Output Parameter	None
Return Parameter	Returned status (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* check the channel0 FIFO */
...
DMA_GetChannelActiveStatus(DMA_Channel0);
...
```

13.2.9 DMA_ITConfig

Function Name	DMA_ITConfig
Function Prototype	`void DMA_ITMaskConfig(DMA_Channel_TypeDef* DMA_Channelx, FunctionalState NewState)`
Behavior Description	Enables or disables the specified DMA_Channelx Terminal count.
Input Parameter1	DMA_Channelx: where x can be 0,1,2,3,4,5,6,or 7 to select the DMA Channel.
Input Parameter2	NewState: new state of the specified DMA_Channelx interrupts. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

13.2.10 DMA_GetChannelStatus

Function Name	DMA_GetChannelStatus
Function Prototype	`FlagStatus DMA_GetChannelStatus(u8 ChannelIndx)`
Behavior Description	Check the status of DMA channelx (Enabled or Disabled)
Input Parameter	ChannelIndx: specifies the DMA Channel to be checked. Refer to section "*ChannelIndx on page 159*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	Returned status (SET or RESET).
Required preconditions	...
Called functions	None

ChannelIndx

This Parameter is used as index for the channel to be checked (there are eight Channels)

ChannelIndx = Channel0=0

ChannelIndx = Channel1 =1

...

ChannelIndx = Channel7 =7

13.2.11 DMA_GetITStatus

Function Name	DMA_GetITStatus
Function Prototype	`ITStatus DMA_GetITStatus(u8 ChannelIndx,u8 DMA_ITReq)`
Behavior Description	Checks the status of Terminal Count and Error interrupts request after and before Masking.
Input Parameter1	ChannelIndx: specifies the DMA Channel to be checked. Refer to section "*ChannelIndx on page 159*" for more details on the allowed values of this parameter.
Input Parameter2	DMA_ITReq: specifies the DMA interrupt request status to be checked. Refer to section "*DMA_ITReq: on page 161*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	Returned status (SET or RESET).
Required preconditions	...
Called functions	None

ChannelIndx

This Parameter is used as index for the channel to be checked (there are eight Channels)

ChannelIndx = Channel0 = 0

ChannelIndx = Channel1 = 1

...

ChannelIndx = Channel7 =7

DMA_ITReq:

Specifies the DMA interrupt request status to be checked and is coded as index for the register

Status containing this interrupt request status.

To check the interrupt status, use a combination of one or more of the following values:

DMA_ITReq	Value (code)	DMA register status	Meaning
DMA_IS	0x01	DMA_ISR	The status of the interrupts after masking.
DMA_TCS	0x02	DMA_TCISR	The status of the terminal count after masking.
DMA_ES	0x03	DMA_EISR	The status of the error request after masking
DMA_TCRS	0x04	DMA_TCRISR	Indicates if the DMA channel is requesting a transfer complete (terminal count Interrupt) prior to masking or Not.
DMA_ERS	0x05	DMA_ERISR	Indicates if the DMA channel is requesting an Error Interrupt prior to masking or Not.

Example:

```
/*Check the status of interrupts after masking on the channel0*/
...
ITStatus DMA_GetITStatus(Channel0,DMA_IS);
...
```

13.2.12 DMA_ClearIT

Function Name	DMA_ClearIT
Function Prototype	`void DMA_ClearIT(u8 ChannelIndx,u8 DMA_ITClr)`
Behavior Description	Clears The Interrupt pending bits for terminal count or Error interrupts for a specified DMA Channel.
Input Parameter1	ChannelIndx: specifies the DMA Channel. Refer to section "*ChannelIndx on page 160*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

DMA_ITClr:

Specifies the DMA interrupt pending to be cleared and is coded as index for the register containing this interrupt pending bit.

To clear interrupts pending bits, use a combination of one or more of the following values:

DMA_ITClr	Value (code)	DMA register status	Meaning
DMA_TCC	0X01	DMA_TCICR	Clear a Terminal Count Interrupt on the corresponding DMA channel
DMA_EC	0X02	DMA_EICR	Clear an Error Interrupt on the corresponding DMA channel

13.2.13 DMA_SyncConfig

Function Name	DMA_SyncConfig
Function Prototype	`void DMA_SyncConfig(u16 DMA_SrcReq, FunctionalState NewState)`
Behavior Description	enable or disable synchronization logic for the corresponding DMA Request Signal.
Input parameter1	`DMA_SrcReq`: specifies the DMA Request Source. Refer to section "*DMA_SrcReq: on page 163*" for more details on the allowed values of this parameter.
Input Parameter2	NewState: new state of the DMA peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

Note: You must use synchronization logic when the peripheral generating the DMA request runs on a different clock to the DMA. For peripherals running on the same clock as the DMA, disabling the synchronization logic improves the DMA request.

DMA_SrcReq:

DMA_SrcReq	Value (code)	Meaning
DMA_USB_RX_Mask	0x0001	
DMA_USB_TX_Mask	0x0002	
DMA_TIM1 _Mask	0x0004	
DMA_TIM2_Mask	0x0008	
DMA_UART0_RX_Mask	0x0010	
DMA_UART0_TX_Mask	0x0020	
DMA_UART1_RX_Mask	0x0040	2 out of 3 UARTs have DMA support (UART0 is not supported by DMA)
DMA_UART1_TX_Mask	0x0080	
DMA_External_Req0_Mask	0x0100	
DMA_External_Req1_Mask	0x0200	
DMA_I2C0_Mask	0x0400	
DMA_I2C1_Mask	0x0800	
DMA_SSP0_RX_Mask	0x1000	
DMA_SSP0_TX_Mask	0x2000	
DMA_SSP1_RX_Mask	0x4000	
DMA_SSP1_TX_Mask	0x8000	

13.2.14 DMA_GetSReq

Function Name	DMA_GetSReq
Function Prototype	`FlagStatus DMA_GetSReq(u16 DMA_SrcReq)`
Behavior Description	Checks for a specific source if it requests a Single transfer or not.
Input Parameter	DMA_SrcReq: specifies the DMA Request Source. Refer to section "*DMA_SrcReq: on page 163*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	Returned status (SET or RESET).
Required preconditions	None
Called functions	None

13.2.15 DMA_GetLSReq

Function Name	DMA_GetLSReq
Function Prototype	`FlagStatus DMA_GetLSReq(u16 DMA_SrcReq)`
Behavior Description	Checks for a specific source if it requests a last Single transfer.
Input Parameter	DMA_SrcReq: specifies the DMA Request Source. Refer to section *DMA_SrcReq: on page 163* for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	Returned status (SET or RESET).
Required preconditions	None
Called functions	None

13.2.16 DMA_GetBReq

Function Name	DMA_GetBReq
Function Prototype	`FlagStatus DMA_GetBReq(u16 DMA_SrcReq)`
Behavior Description	Checks for a specific source if it requests a Burst transfer.
Input Parameter	`DMA_SrcReq`: specifies the DMA Request Source. Refer to section "*DMA_SrcReq: on page 163*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	Returned status (SET or RESET).
Required preconditions	None
Called functions	None

13.2.17 DMA_GetLBReq

Function Name	DMA_GetLBReq
Function Prototype	`FlagStatus DMA_GetLBReq(u16 DMA_SrcReq)`
Behavior Description	Checks for a specific source if it requests a Last Burst transfer.
Input Parameter	DMA_SrcReq: specifies the DMA Request Source. Refer to section" *DMA_SrcReq: on page 163*" for more details on the allowed values of this parameter. (section 1.12.9).
Output Parameter	None
Return Parameter	Returned status (SET or RESET).
Required preconditions	None
Called functions	None

13.2.18 DMA_SetSReq

Function Name	DMA_SetSReq
Function Prototype	`void DMA_SetSReq(u16 DMA_SrcReq)`
Behavior Description	Sets the DMA to generate a Single transfer request for the corresponding DMA Request Source.
Input Parameter	DMA_SrcReq: specifies the DMA Request Source. Refer to section "*DMA_SrcReq: on page 163*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

13.2.19 DMA_SetLSReq

Function Name	DMA_SetLSReq
Function Prototype	`void DMA_SetLSReq(u16 DMA_SrcReq)`
Behavior Description	Sets the DMA to generate a Last Single transfer request for the corresponding DMA Request Source.
Input Parameter	DMA_SrcReq: specifies the DMA Request Source. Refer to section "*DMA_SrcReq: on page 163*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

13.2.20 DMA_SetBReq

Function Name	DMA_SetBReq
Function Prototype	`void DMA_SetBReq(u16 DMA_SrcReq)`
Behavior Description	Sets the DMA to generate a Burst transfer request for the corresponding DMA Request Source.
Input Parameter	DMA_SrcReq: specifies the DMA Request Source. Refer to section "*DMA_SrcReq: on page 163*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

13.2.21 DMA_SetLBReq

Function Name	DMA_SetLBReq
Function Prototype	`void DMA_SetLBReq(u16 DMA_SrcReq)`
Behavior Description	Sets the DMA to generate a Last Burst transfer request for the corresponding DMA Request Source.
Input Parameter	DMA_SrcReq: specifies the DMA Request Source. Refer to section "*DMA_SrcReq: on page 163*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

13.2.22 DMA_ChannelCmd

Function Name	DMA_ChannelCmd
Function Prototype	`void DMA_ChannelCmd (DMA_Channel_TypeDef * DMA_Channelx, FunctionalState NewState)`
Behavior Description	Enables or disables the DMA_Channelx.
Input Parameter1	DMA_Channelx: where x can be 0,1,2,3,4,5,6,or 7 to select the DMA Channel.
Input Parameter2	NewState: new state of the DMA peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable DMA_Channel0 */
DMA_ChannelCmd(DMA_Channel0,ENABLE);
```

13.2.23 DMA_ChannelHalt

Function Name	DMA_ChannelHalt
Function Prototype	`void DMA_ChannelHalt (DMA_Channel_TypeDef * DMA_Channelx,FunctionalState NewState)`
Behavior Description	Enables or disables HALT Mode for the specified DMA_Channelx.
Input Parameter1	DMA_Channelx: where x can be 0,1,2,3,4,5,6,or 7 to select the DMA Channel.
Input Parameter2	NewState: new state of the DMA peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable Halt Mode for DMA_Channel0 (ignore extra source DMA requests)*/
DMA_ChannelHalt(DMA_Channel0,ENABLE);
```

13.2.24 DMA_ChannelBuffering

Function Name	DMA_ChannelBuffering
Function Prototype	`void DMA_ChannelBuffering (DMA_Channel_TypeDef * DMA_Channelx,FunctionalState NewState)`
Behavior Description	Enables or disables the Buffering Feature for the specified DMA_Channelx.
Input Parameter1	DMA_Channelx: where x can be 0,1,2,3,4,5,6,or 7 to select the DMA Channel.
Input Parameter2	NewState: new state of the DMA peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
DMA_ChannelBuffering(DMA_Channel0,ENABLE);
/* Buffering feature enabled for the DMA_Channel0 */
```

13.2.25 DMA_ChannelLockTrsf

Function Name	DMA_ChannelLockTrsf
Function Prototype	`void DMA_ChannelLockTrsf(DMA_Channel_TypeDef * DMA_Channelx,FunctionalState NewState)`
Behavior Description	Enables or disables the Locked Transfers Feature for the specified DMA_Channelx.
Input Parameter1	DMA_Channelx: where x can be 0,1,2,3,4,5,6,or 7 to select the DMA Channel.
Input Parameter2	NewState: new state of the DMA peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the locked transfers feature for the DMA_Channel0 */
DMA_ChannelLockTrsf(DMA_Channel0,ENABLE);
```

13.2.26 DMA_ChannelCache

Function Name	DMA_ChannelCache
Function Prototype	`void DMA_ChannelCache(DMA_Channel_TypeDef * DMA_Channelx,FunctionalState NewState)`
Behavior Description	Enables or disables the cache ability Feature for the specified DMA_Channelx.
Input Parameter1	DMA_Channelx: where x can be 0,1,2,3,4,5,6,or 7 to select the DMA Channel.
Input Parameter2	NewState: new state of the DMA peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the cacheability for the DMA_Channel0 */
DMA_ChannelCache(DMA_Channel0,ENABLE);
```

13.2.27 DMA_ChannelProt0Mode

Function Name	DMA_ChannelProt0Mode
Function Prototype	`void DMA_ChannelProt0Mode(DMA_Channel_TypeDef * DMA_Channelx,u32 Prot0Mode)`
Behavior Description	Sets User or Privileged mode for the specified DMA_Channelx.
Input Parameter1	DMA_Channelx: where x can be 0,1,2,3,4,5,6,or 7 to select the DMA Channel.
Input Parameter2	Prot0Mode: specifies the Privileged mode Or the User mode
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Prot0Mode:

Prot0Mode	**Value**	**Meaning**
DMA_PrivilegedMode	0X10000000	
DMA_UserMode	0XEFFFFFFF	

14 Synchronous Serial Peripheral (SSP)

The SSP is a master or slave interface for synchronous serial communication with peripheral devices that have either Motorola SPI, National Semiconductor or Texas Instruments SSI synchronous serial interfaces.

The first section describes the data structure used in the SSP software library. The second one presents the software library functions.

14.1 SSP Register structure

The SSP register structure *SSP_TypeDef* is defined in the ***91x_map.h*** file as follows:

```
typedef struct
{
vu16 CR0;        /* Control Register 1                   */
  vu16 EMPTY1;
  vu16 CR1;        /* Control Register 2                   */
  vu16 EMPTY2;
  vu16 DR;         /* Data Register                        */
  vu16 EMPTY3;
  vu16 SR;         /* Status Register                      */
  vu16 EMPTY4;
  vu16 PR;       /* Clock Prescale Register              */
  vu16 EMPTY5;
  vu16 IMSCR;       /* Interrupt Mask Set or Clear Register */
  vu16 EMPTY6;
  vu16 RISR;        /* Raw Interrupt Status Register        */
  vu16 EMPTY7;
  vu16 MISR;        /* Masked Interrupt Status Register     */
  vu16 EMPTY8;
  vu16 ICR;        /* Interrupt Clear Register             */
  vu16 EMPTY9;
  vu16 DMACR;      /* DMA Control Register                 */
  vu16 EMPTY10;
} SSP_TypeDef;
```

The following table presents the SSP registers:

Register	Description
CR0	SSP Control Logic register 0
CR1	SSP Control Logic register 1
DR	SSP Data register
SR	SSP Status register
PR	SSP Clock Prescaler register
IMSCR	SSP Interrupt Mask Set and Clear register
RISR	SSP Raw Interrupt Status register
MISR	SSP Masked Interrupt Status register
ICR	SSP Interrupt Status register
DMACR	SSP DMA Control register

The two SSP interfaces are declared in the same file:

```
#ifndef EXT
 #Define EXT extern
#endif
...
#define AHB_APB_BRDG1_U    (0x5C000000) /* AHB/APB Bridge 1 UnBuffered Space */
#define AHB_APB_BRDG1_B    (0x4C000000) /* AHB/APB Bridge 1 Buffered Space   */
...
#define APB_SSP0_OFST       (0x00007000) /* Offset of SSP0 */
#define APB_SSP1_OFST       (0x00008000) /* Offset of SSP1 */
...
#ifndef Buffered
#define AHBAPB1_BASE           (AHB_APB_BRDG1_U)
...
#else /* Buffered */
...
#define AHBAPB1_BASE           (AHB_APB_BRDG1_B)

/* SSP Base Address definition*/
#define SSP0_BASE          (AHBAPB1_BASE + APB_SSP0_OFST)
#define SSP1_BASE          (AHBAPB1_BASE + APB_SSP1_OFST)
...
/* SSP peripheral declaration*/

#ifndef DEBUG
...
#define SSP0    ((SSP_TypeDef *) SSP0_BASE)
#define SSP1    ((SSP_TypeDef *) SSP1_BASE)

...
#else
...
#ifdef _SSP0
EXT SSP_TypeDef            *SSP0;
#endif /* _SSP0 */

#ifdef _SSP1
EXT SSP_TypeDef            *SSP1;
```

```
#endif /* _SSP1 */
...

When debug mode is used, SSP pointers are initialized in 91x_lib.c file:
#ifdef _SSP0
SSP0 = (SSP_TypeDef *)SSP0_BASE;
#endif /*_SSP0 */

#ifdef _SSP1
SSP1 = (SSP_TypeDef *)SSP1_BASE;
#endif /*_SSP1 */
```

_SSP, _SSP0 and _SSP1 must be defined, in the ***91x_conf.h*** file, to access the peripheral registers as follows:

```
#define _SSP
#define _SSP0
#define _SSP1
...
```

14.2 Software library functions

The following table enumerates the functions of the SSP library.

Function Name	Description
SSP_DeInit	Deinitializes the SSPx peripheral registers to their default reset values.
SSP_Init	Initializes the SSPx peripheral according to the specified parameters in the SSP_InitStruct.
SSP_StructInit	Fills each SSP_InitStruct member with its default value.
SSP_Cmd	Enables or disables the specified SSP peripheral.
SSP_ITConfig	Enables or disables the specified SSP interrupts.
SSP_DMACmd	Configures the SSP DMA interface.
SSP_SendData	Transmits a Data through the SSP peripheral.
SSP_ReceiveData	Returns the most recent received data by the SSP peripheral.
SSP_LoopBackConfig	Enables or disables Loop back mode for the selected SSP peripheral.
SSP_GetFlagStatus	Checks whether the specified SSP flag is set or not.
SSP_ClearFlag	Clears the SSPx pending flags.
SSP_GetITStatus	Checks whether the specified SSP interrupt has occurred or not.
SSP_ClearITPendingBit	Clears the SSPx interrupt pending bits.

14.2.1 SSP_DeInit

Function Name	SSP_DeInit
Function Prototype	`void SSP_DeInit(SSP_TypeDef* SSPx)`
Behavior Description	Deinitializes the SSPx peripheral registers to their default reset values.
Input Parameter	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	SCU_APBPeriphReset()

Example:

```
/* Deinitialize the SSP0 peripheral. */
SSP_DeInit(SSP0);
```

14.2.2 SSP_Init

Function Name	SSP_Init
Function Prototype	`void SSP_Init(SSP_TypeDef* SSPx, SSP_InitTypeDef* SSP_InitStruct)`
Behavior Description	Initializes the SSPx peripheral according to the specified parameters in the SSP_InitStruct .
Input Parameter1	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Input Parameter2	SSP_InitStruct: pointer to a SSP_InitTypeDef structure that contains the configuration information for the specified SSP peripheral. Refer to section "*SSP_InitTypeDef on page 174*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

SSP_InitTypeDef

The SSP_InitTypeDef structure is defined in the ***91x_SSP.h*** file:

```
typedef struct
{
u16 SSP_FrameFormat ;
u16 SSP_Mode ;
u16 SSP_CPOL ;
u16 SSP_CPHA ;
u16 SSP_DataSize ;
u16 SSP_SlaveOutput ;
u8  SSP_ClockRate ;
u8  SSP_ClockPrescaler ;
} SSP_InitTypeDef;
```

SSP_FrameFormat

Specifies whether the frame format is Motorola SPI or TI synchronous serial or Microwire. This member can be one of the following values:

SSP_FrameFormat	Meaning
SSP_FrameFormat_Motorola	Motorola frame format is selected
SSP_FrameFormat_TI	TI frame format is selected
SSP_FrameFormat_Microwire	Microwire frame format is selected

SSP_Mode

Specifies the SSP operation mode. This member can be one of the following values:

SSP_Mode	Meaning
SSP_Mode_Master	SSP is configured as a master
SSP_Mode_Slave	SSP is configured as a slave

SSP_CPOL

Specifies the steady state value of the serial clock. This member can be one of the following values:

SSP_CPOL	Meaning
SSP_CPOL_Low	Clock is active low
SSP_CPOL_High	Clock is active high

SSP_CPHA

Specifies on which clock transition the bit capture is made. This member can be one of the following values:

SSP_CPHA	Meaning
SSP_CPHA_2Edge	Data is captured on the second edge
SSP_CPHA_1Edge	Data is captured on the first edge

SSP_DataSize

Specifies the word length operation of the receive and transmit FIFO. This member can be one of the following values:

SSP_DataSize	Meaning
SSP_DataSize_16b	Data Size is 16 bits
SSP_DataSize_15b	Data Size is 15 bits
SSP_DataSize_14b	Data Size is 14 bits
SSP_DataSize_13b	Data Size is 13 bits
SSP_DataSize_12b	Data Size is 12 bits
SSP_DataSize_11b	Data Size is 11 bits
SSP_DataSize_10b	Data Size is 10 bits
SSP_DataSize_9b	Data Size is 9 bits
SSP_DataSize_8b	Data Size is 8 bits
SSP_DataSize_7b	Data Size is 7 bits
SSP_DataSize_6b	Data Size is 6 bits
SSP_DataSize_5b	Data Size is 5 bits
SSP_DataSize_4b	Data Size is 4 bits

SSP_SlaveOutput

Specifies whether the slave output is enabled or disabled. This is used especially in multiple-slave cases. This member can be one of the following values:

SSP_SlaveOutput	Meaning
SSP_SlaveOutput_Enable	Slave output enabled
SSP_SlaveOutput_Disable	Slave output disabled

SSP_ClockRate

Specifies the serial clock rate value used to configure the transmit and receive bit rate of SCK. This member must be a value from 2 to 254.

SSP_ClockPrescaler

Specifies the division factor by which the input PCLK must be divided to be used by the SSP. This member must be an even number between 2 and 254. The least significant bit of the programmed number is hardcoded to zero. If an odd number is written the least significant bit is changed to zero by hardware.

	MOTOROLA Mode		TI Mode		Mirowire Mode	
Members	**Master**	**Slave**	**Master**	**Slave**	**Master**	**Slave**
SSP_FrameFormat	x	x	x	x	x	x
SSP_Mode	x	x	x	x	x	x
SSP_CPOL	x	x				
SSP_CPHA	x	x				
SSP_DataSize	x	x	x	x	x	x
SSP_SlaveOutput		x		x		x
SSP_ClockRate	x	(1)	x	(1)	x	(1)
SSP_ClockPrescaler	x	(1)	x	(1)	x	(1)

(1): the communication clock is derived from the master so no need to set the slave clock

Example:

```
/* Initialize the SSP0 according to the SSP_InitStructure members. */
SSP_InitTypeDef  SSP_InitStructure;

SSP_InitStructure.SSP_FrameFormat = SSP_FrameFormat_TI;
SSP_InitStructure.SSP_Mode = SSP_Mode_Slave;
SSP_InitStructure.SSP_CPOL = SSP_CPOL_High;
SSP_InitStructure.SSP_CPHA = SSP_CPHA_1Edge;
SSP_InitStructure.SSP_DataSize = SSP_DataSize_8b;
SSP_InitStructure.SSP_SlaveOutput = SSP_SlaveOutput_Enable;
SSP_InitStructure.SSP_ClockRate = 0xB;
SSP_InitStructure.SSP_ClockPrescaler = 12;
SSP_Init(SSP0, &SSP_InitStructure);
```

14.2.3 SSP_StructInit

Function Name	SSP_StructInit
Function Prototype	`void SSP_StructInit(SSP_InitTypeDef* SSP_InitStruct)`
Behavior Description	Fills each SSP_InitStruct member with its default value, refer to the following table for more details.
Input Parameter	SSP_InitStruct: pointer to a SSP_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

The SSP_InitStruct members default values are as follows:

Member	Default value
SSP_FrameFormat	SSP_FrameFormat_Motorola
SSP_Mode	SSP_Mode_Master
SSP_CPOL	SSP_CPOL_Low
SSP_CPHA	SSP_CPHA_1Edge
SSP_DataSize	SSP_DataSize_8b
SSP_SlaveOutput	SSP_SlaveOutput_Enable
SSP_ClockRate	0
SSP_ClockPrescaler	0

Example:

```
/* Initialize a SSP_InitTypeDef structure. */
SSP_InitTypeDef  SSP_InitStructure;
SSP_StructInit(&SSP_InitStructure);
```

14.2.4 SSP_Cmd

Function Name	SSP_Cmd
Function Prototype	`void SSP_Cmd(SSP_TypeDef* SSPx, FunctionalState NewState)`
Behavior Description	Enables or disables the specified SSP peripheral.
Input Parameter1	SSPx: where x can be 0 or 1to select the SSP peripheral.
Input Parameter2	NewState: new state of the SSPx peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the SSP0 peripheral */
SSP_Cmd(SSP0, ENABLE);
```

14.2.5 SSP_ITConfig

Function Name	SSP_ITConfig
Function Prototype	`void SSP_ITConfig(SSP_TypeDef* SSPx, u16 SSP_IT, FunctionalState NewState)`
Behavior Description	Enables or disables the specified SSP interrupts.
Input Parameter1	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Input Parameter2	SSP_IT: specifies the SSP interrupts sources to be enabled or disabled. Refer to section "*SSP_IT*" for more details on the allowed values of this parameter. You can select more than one interrupt, by ORing them.
Input Parameter3	NewState: new state of the specified SSP interrupts. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

SSP_IT

To enable or disable SSP interrupts, use a combination of one or more of the following values:

SSP_IT	Meaning
SSP_IT_TxFifo	Transmit FIFO half empty or less condition interrupt
SSP_IT_RxFifo	Receive FIFO half full or less condition interrupt
SSP_IT_RxTimeOut	Receive timeout interrupt
SSP_IT_RxOverrun	Receive overrun interrupt

Example:

```
/* Enable SSP0 Transmit FIFO and Receive FIFO interrupts. */
SSP_ITConfig(SSP0, SSP_IT_TxFifo | SSP_IT_RxFifo, ENABLE);
```

14.2.6 SSP_DMACmd

Function Name	SSP_DMACmd
Function Prototype	`void SSP_DMACmd(SSP_TypeDef* SSPx,u16 SSP_DMAtransfer, FunctionalState NewState)`
Behavior Description	Configures the SSP DMA interface.
Input Parameter1	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Input Parameter2	SSP_DMAtransfer: specifies the SSP DMA transfer to be enabled or disabled. Refer to section "*SSP_DMATransfer*" for more details on the allowed values of this parameter. You can select more than one interrupt, by ORing them.
Input Parameter3	NewState: new state of SSP0 DMA transfer. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None

SSP_DMATransfer

To enable or disable SSP0 DMA transfer, use one of the following values:

SSP0_DMAtransfer	Meaning
SSP0_DMA_Transmit	Transmit FIFO DMA transfer
SSP0_DMA_Receive	Receive FIFO DMA transfer

Example:

```
/* Enable SSP0 transmit FIFO DMA transfer. */
SSP_DMACmd(SSP0, SSP_DMA_Transmit, ENABLE);
```

14.2.7 SSP_SendData

Function Name	SSP_SendData
Function Prototype	`void SSP_SendData(SSP_TypeDef* SSPx, u16 Data)`
Behavior Description	Transmits a Data through the SSP peripheral.
Input Parameter1	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Input Parameter 2	Data: word to be transmitted.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called Functions	None

Example:

```
/* Send 0xA5 through the SSP0 peripheral. */
SSP_SendData(SSP0, 0xA5);
```

14.2.8 SSP_ReceiveData

Function Name	SSP_ReceiveData
Function Prototype	`u16 SSP_ReceiveData(SSP_TypeDef* SSPx)`
Behavior Description	Returns the most recent data received by the SSP peripheral.
Input Parameter	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Output Parameter	None
Return Parameter	The value of the received data.
Required preconditions	None
Called Functions	None

Example:

```
/* Read the most recent data received by the SSP0 peripheral. */
u16 hReceivedData;
hReceivedData = SSP_ReceiveData(SSP0);
```

14.2.9 SSP_LoopBackConfig

Function Name	SSP_LoopBackConfig
Function Prototype	`SSP_LoopBackConfig(SSP_TypeDef* SSPx, FunctionalState NewState)`
Behavior Description	Enables or disables the Loop back mode for the selected SSP peripheral.
Input Parameter1	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Input Parameter2	NewState: new state of the Loop back mode. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called Functions	None

Example:

```
/* Enable the Loop back mode for the SSP0 peripheral. */
SSP_LoopBackConfig(SSP0, ENABLE);
```

14.2.10 SSP_GetFlagStatus

Function Name	SSP_GetFlagStatus
Function Prototype	`FlagStatus SSP_GetFlagStatus(SSP_TypeDef* SSPx, u16 SSP_FLAG)`
Behavior Description	Checks whether the specified SSP flag is set or not.
Input Parameter1	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Input Parameter2	SSP_FLAG: specifies the flag to check. Refer to section " *SSP_FLAG*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of SSP_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

SSP_FLAG

The SSP flags that can be read are listed in the following table:

SSP_FLAG	Meaning
SSP_FLAG_Busy	Busy flag
SSP_FLAG_RxFifoFull	Receive FIFO full flag
SSP_FLAG_RxFifoNotEmpty	Receive FIFO not empty flag
SSP_FLAG_TxFifoNotFull	Transmit FIFO not full flag
SSP_FLAG_TxFifoEmpty	Transmit FIFO empty flag
SSP_FLAG_TxFifo	Transmit FIFO Masked interrupt flag
SSP_FLAG_RxFifo	Receive FIFO Masked interrupt flag
SSP_FLAG_RxTimeOut	Receive timeout Masked interrupt flag
SSP_FLAG_RxOverrun	Receive overrun Masked interrupt flag

Example:

```
/* Get the Receive FIFO full flag status */
FlagStatus Status;
Status = SSP_GetFlagStatus(SSP0, SSP_FLAG_RxFifoFull);
```

14.2.11 SSP_ClearFlag

Function Name	SSP_ClearFlag
Function Prototype	`void SSP_ClearFlag(SSP_TypeDef* SSPx, u16 SSP_FLAG)`
Behavior Description	Clears the SSPx pending flags.
Input Parameter1	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Input Parameter 2	SSP_FLAG: specifies the flag to clear. Refer to the section "*SSP_IT on page 184*" for more details on the allowed values of this parameter.
OutPut Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

SSP_FLAG

To clear the SSP flags, use one of the following values:

SSP_FLAG	Meaning
SSP_FLAG_RxTimeOut	Receive timeout flag
SSP_FLAG_RxOverrun	Receive overrun flag

Example:

```
/* Clear the SSP0 Receive timeout flag */
SSP_ClearFlag(SSP0, SSP_FLAG_RxTimeOut);
```

14.2.12 SSP_GetITStatus

Function Name	SSP_GetITStatus
Function Prototype	`ITStatus SSP_GetITStatus(SSP_TypeDef* SSPx, u16 SSP_IT)`
Behavior Description	Checks whether the specified SSP interrupt has occurred or not.
Input Parameter1	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Input Parameter2	SSP_IT: specifies the interrupt source to check. Refer to section "*SSP_IT on page 184*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of SSP_IT (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Get the SSP0 Receive FIFO interrupt status */
ITStatus Status;
Status = SSP_GetITStatus(SSP0, SSP_IT_RxFifo);
```

14.2.13 SSP_ClearITPendingBit

Function Name	SSP_ClearITPendingBit
Function Prototype	`void SSP_ClearITPendingBit(SSP_TypeDef* SSPx, u16 SSP_IT)`
Behavior Description	Clears the SSPx's interrupt pending bits.
Input Parameter1	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Input Parameter 2	SSP_IT: specifies the interrupt pending bit to clear. More than one interrupt can be cleared using the "|" operator. Refer to the section "*SSP_IT on page 179*" for more details on the allowed values of this parameter.
OutPut Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

SSP_IT

To clear the SSP interrupts pending bits, use a combination of one or more of the following values:

SSP_IT	Meaning
SSP_IT_RxTimeOut	Receive timeout interrupt
SSP_IT_RxOverrun	Receive overrun interrupt

Example:

```
/* Clear the SSP0 Receive timeout interrupt pending bit */
SSP_ClearITPendingBit(SSP0, SSP_IT_RxTimeOut);
```

15 Universal Asynchronous Receiver Transmitter (UART)

The UART interface provides hardware management of the CTS and RTS signals and has Full Modem interface (on UART0 only). It also supports IrDA mode that reduces the signal for IrDA by 3/16.

To optimize the data transfer between the processor and the peripheral, the UART has two FIFOs (receive/transmit) of 16 bytes each. The UART can be served by the DMA controller.

The first section describes the data structures used in the UART software library. The second one presents the software library functions.

15.1 UART register structure

The UART register structure *UART_TypeDef* is defined in the ***91x_map.h*** file as follows:

```
typedef struct
{
vu16 DR;
  vu16 EMPTY1;
  vu16 RSECR;
  vu16 EMPTY2[9];
  vu16 FR;
  vu16 EMPTY3[3];
  vu16 ILPR;
  vu16 EMPTY4;
  vu16 IBRD;
  vu16 EMPTY5;
  vu16 FBRD;
  vu16 EMPTY6;
  vu16 LCR;
  vu16 EMPTY7;
  vu16 CR;
  vu16 EMPTY8;
  vu16 IFLS;
  vu16 EMPTY9;
  vu16 IMSC;
  vu16 EMPTY10;
  vu16 RIS;
  vu16 EMPTY11;
  vu16 MIS;
  vu16 EMPTY12;
  vu16 ICR;
  vu16 EMPTY13;
  vu16 DMACR;
  vu16 EMPTY14
} UART_TypeDef;
```

The following table presents the UART registers:

Register	Description
DR	Data Register
RSECR	Receive Status Register
FR	Flag Register
ILPR	IrDA Low-Power Counter Register
IBRD	Integer Baud Rate Divider Register
FBRD	Fractional Baud Rate Divider Register
LCR	Line Control Register
CR	Control Register
IFLS	Interrupt FIFO Level Select
IMSC	Interrupt Mask Set/Clear Register
RIS	Raw Interrupt Status
MIS	Masked Interrupt Status
ICR	Interrupt Clear Register
DMACR	DMA Control Register

The 3 UART interfaces are declared in the same file:

```
#ifndef EXT
 #Define EXT extern
#endif
...
#define AHB_APB_BRDG1_U    (0x5C000000) /* AHB/APB Bridge 1 UnBuffered Space */
#define AHB_APB_BRDG1_B    (0x4C000000) /* AHB/APB Bridge 1 Buffered Space   */
...
#define APB_UART0_OFST       (0x00004000) /* Offset of UART0 */
#define APB_UART1_OFST       (0x00005000) /* Offset of UART1 */
#define APB_UART2_OFST       (0x00006000) /* Offset of UART2 */
...
#ifndef Buffered
#define AHBAPB1_BASE           (AHB_APB_BRDG1_U)
...
#else /* Buffered */
...
#define AHBAPB1_BASE           (AHB_APB_BRDG1_B)
```

```
/* UART Base Address definition*/
#define UART0_BASE          (AHBAPB1_BASE + APB_UART0_OFST)
#define UART1_BASE          (AHBAPB1_BASE + APB_UART1_OFST)
#define UART2_BASE          (AHBAPB1_BASE + APB_UART2_OFST)
...
/* UART peripheral declaration*/

#ifndef DEBUG
```

```
...
#define UART0    ((UART_TypeDef *) UART0_BASE)
#define UART1    ((UART_TypeDef *) UART1_BASE)
#define UART2    ((UART_TypeDef *) UART2_BASE)

...
#else
...
#ifdef _UART0
EXT UART_TypeDef           *UART0;
#endif /* _UART0 */

#ifdef _UART1
EXT UART_TypeDef           *UART1;
#endif /* _UART1 */

#ifdef _UART2
EXT UART_TypeDef           *UART2;
#endif /* _UART2*/
...
```

- When debug mode is used, UART pointers are initialized in ***91x_lib.c*** file:

```
#ifdef _UART0
UART0 = (UART_TypeDef *)UART0_BASE;
#endif /*_UART0 */

#ifdef _UART1
UART1 = (UART_TypeDef *)UART1_BASE;
#endif /*_UART1 */

#ifdef _UART2
UART2 = (UART_TypeDef *)UART2_BASE;
#endif /*_UART2 */
```

_UART, _UART0, _UART1 and _UART2 must be defined, in ***91x_conf.h*** file, to access the peripheral registers as follows:

```
#define _UART
#define _UART0
#define _UART1
#define _UART2
```

15.2 Software library functions

The following table enumerates the different functions of the UART library.

Function Name	Description
UART_DeInit	Deinitializes the UARTx peripheral registers to their default reset values.
UART_Init	Initializes the UARTx peripheral according to the specified parameters in the UART_InitStruct.
UART_StructInit	Fills each UART_InitStruct member with its default value.
UART_Cmd	Enables or disables the specified UART peripheral.
UART_ITConfig	Enables or disables the specified UART interrupts.
UART_DMAConfig	Configures the UARTx DMA interface.
UART_DMACmd	Enables or disables the UARTx DMA interface.
UART_LoopBackConfig	Enables or disables the UARTx LoopBack mode.
UART_IrDALowPowerConfig	Sets the IrDA low power mode
UART_IrDASetCounter	Sets the IrDA counter divisor value in low power mode.
UART_IrDACmd	Enables or disables the UARTx IrDA interface
UART_SendData	Transmits a Byte of data through the UARTx peripheral.
UART_ReceiveData	Returns the most recent byte received by the UARTx peripheral.
UART_SendBreak	Transmits break characters.
UART_RTSConfig	Sets or resets the RTS signal (for UART0 only).
UART_DTRConfig	Sets or resets the DTR signal (for UART0 only).
UART_GetFlagStatus	Checks whether the specified UART flag is set or not.
UART_ClearFlag	Clears the UARTx pending flags.
UART_GetITStatus	Checks whether the specified UART interrupt has occurred or not.
UART_ClearITPendingBit	Clears the UARTx interrupt pending bits.

15.2.1 UART_DeInit

Function Name	UART_DeInit
Function Prototype	`void UART_DeInit(UART_TypeDef* UARTx)`
Behavior Description	Deinitializes the UARTx peripheral registers to their default reset values.
Input Parameter	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	SCU_APBPeriphReset()

Example:

```
/* Deinitializes the UART0 registers to their default reset value */
UART_DeInit(UART0);
```

15.2.2 UART_Init

Function Name	UART_Init
Function Prototype	`void UART_Init(UART_TypeDef* UARTx, UART_InitTypeDef* UART_InitStruct)`
Behavior Description	Initializes the UARTx peripheral according to the specified parameters in the UART_InitStruct .
Input Parameter1	UARTx: where x can be 0, 1 or 2 to select the UART peripheral.
Input Parameter2	UART_InitStruct: pointer to a UART_InitTypeDef structure that contains the configuration information for the specified UART peripheral. Refer to section "*UART_InitTypeDef*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

UART_InitTypeDef

The UART_InitTypeDef structure is defined in the ***91x_uart.h*** file:

```
typedef struct
{
  u16 UART_WordLength;
  u16 UART_StopBits;
  u16 UART_Parity;
  u32 UART_BaudRate;
  u16 UART_HardwareFlowControl;
  u16 UART_Mode;
  u16 UART_FIFO;
  UART_FIFOLevel UART_TxFIFOLevel;
  UART_FIFOLevel UART_RxFIFOLevel;
}UART_InitTypeDef;
```

UART_WordLength

Indicates the number of data bits transmitted or received in a frame.

This member can be one of the following values:

UART_WordLength	Meaning
UART_WordLength_5D	5 bits Data
UART_WordLength_6D	6 bits Data
UART_WordLength_7D	7 bits Data
UART_WordLength_8D	8 bits Data

UART_StopBits

Specifies the number of transmitted stop bits. This member can be one of the following values:

UART_StopBits	Meaning
UART_StopBits_1	One stop bit is transmitted at the end of frame
UART_StopBits_2	Tow stop bits are transmitted at the end of frame

UART_Parity

Specifies the parity mode. This member can be one of the following values:

UART_Parity	Meaning
UART_Parity_No	Parity Disable
UART_Parity_Even	Even Parity
UART_Parity_Odd	Odd Parity
UART_Parity_OddStick	1 is transmitted as bit parity
UART_Parity_EvenStick	0 is transmitted as bit parity

UART_BaudRate

Specifies the baudrate of the UART communication. The baudrate is computed with this formula :

IntegerDivider = ((APBClock) / (16 * (UART_InitStruct->UART_BaudRate)))

FractionalDivider = ((IntegerDivider - ((u32) IntegerDivider)) * 64 + 0.5)

UART_HardFlowControl

Specifies whether hardware flow control mode is enabled or disabled.

This member can be one of the following values:

UART_HardFlowControl	Meaning
UART_HardwareFlowControl_None	HFC Disable
UART_HardwareFlowControl_RTS	RTS Enable
UART_HardwareFlowControl_CTS	CTS Enable
UART_HardwareFlowControl_RTS_CTS	CTS and RTS Enable

UART_Mode

Specifies whether receive and/or transmit mode is enabled or disabled.

This member can be a combination of one or more of the following values:

UART_Transmit	Meaning
UART_Mode_Rx	Receive Enable
UART_Mode_Tx	Transmit Enable
UART_Mode_Tx_Rx	Transmit and Receive Enable

UART_FIFO

Specifies whether the FIFOs (Rx and Tx FIFOs) are enabled or disabled.

This member can be one of the following values:

UART_Transmit	Meaning
UART_FIFO_Enable	FIFOs Enable
UART_FIFO_Disable	FIFOs Disable

UART_TxFIFOLevel

Specifies the level of the UART TxFIFO (not used when FIFOs are disabled).

This member can be one of the following values:

UART_TxFIFOLevel	Meaning
UART_FIFOLevel_1_8	FIFO becomes >= 1/8 full
UART_FIFOLevel_1_4	FIFO becomes >= 1/4 full
UART_FIFOLevel_1_2	FIFO becomes >= 1/2 full
UART_FIFOLevel_3_4	FIFO becomes >= 3/4 full
UART_FIFOLevel_7_8	FIFO becomes >= 7/8 full

UART_RxFIFOLevel

Specifies the level of the UART RxFIFO (not used when FIFOs are disabled).

This member can be one of the following values:

UART_RxFIFOLevel	Meaning
UART_FIFOLevel_1_8	FIFO becomes >= 1/8 full
UART_FIFOLevel_1_4	FIFO becomes >= 1/4 full
UART_FIFOLevel_1_2	FIFO becomes >= 1/2 full
UART_FIFOLevel_3_4	FIFO becomes >= 3/4 full
UART_FIFOLevel_7_8	FIFO becomes >= 7/8 full

Example:

```
/* The following example illustrates how to configure the UART0 */
  UART_InitTypeDef UART_InitStructure;
```

```
UART_InitStructure.UART_WordLength = UART_WordLength_8D;
UART_InitStructure.UART_StopBits = UART_StopBits_1;
UART_InitStructure.UART_Parity = UART_Parity_Odd;
UART_InitStructure.UART_BaudRate = 9600;
UART_InitStructure.UART_HardwareFlowControl = UART_HardwareFlowControl_RTS_CTS;
UART_InitStructure.UART_Mode = UART_Mode_Tx_Rx;
UART_InitStructure.UART_FIFO = UART_FIFO_Enable;
UART_InitStructure.UART_TxFIFOLevel = UART_FIFOLevel_1_2;
UART_InitStructure.UART_RxFIFOLevel = UART_FIFOLevel_1_2;
UART_Init(UART0, &UART_InitStructure);
```

15.2.3 UART_StructInit

Function Name	UART_StructInit
Function Prototype	`void UART_StructInit(UART_InitTypeDef* UART_InitStruct)`
Behavior Description	Fills each UART_InitStruct member with its default value, refer to the following table for more details.
Input Parameter	UART_InitStruct: pointer to a UART_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

The UART_InitStruct members default values are as follows:

Member	Default value
UART_WordLength	UART_WordLength_8D
UART_StopBits	UART_StopBits_1
UART_Parity	UART_Parity_Odd
UART_BaudRate	9600
UART_HardwareFlowControl	UART_HardwareFlowControl_None
UART_Mode	UART_Mode_Tx_Rx
UART_FIFO	UART_FIFO_Enable
UART_TxFIFOLevel	UART_FIFOLevel_1_2
UART_RxFIFOLevel	UART_FIFOLevel_1_2

Example:

```
/* The following example illustrates how to initialize a UART_InitTypeDef structure
*/
UART_InitTypeDef UART_InitStructure;
UART_StructInit(&UART_InitStructure);
```

15.2.4 UART_Cmd

Function Name	UART_Cmd
Function Prototype	`void UART_Cmd(UART_TypeDef* UARTx, FunctionalState NewState)`
Behavior Description	Enables or disables the specified UART peripheral.
Input Parameter1	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Input Parameter2	NewState: new state of the UARTx peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the UART0 */
UART_Cmd(UART0, ENABLE);
```

15.2.5 UART_ITConfig

Function Name	UART_ITConfig
Function Prototype	`void UART_ITConfig(UART_TypeDef* UARTx, u16 UART_IT, FunctionalState NewState)`
Behavior Description	Enables or disables the specified UART interrupts.
Input Parameter1	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Input Parameter2	UART_IT: specifies the UART interrupts sources to be enabled or disabled. Refer to section "*UART_IT on page 195*" for more details on the allowed values of this parameter.
Input Parameter3	NewState: new state of the specified UARTx interrupts. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

UART_IT

To enable or disable UART interrupts, use a combination of one or more of the following values:

UART_IT	Meaning
UART_IT_OverrunError	Overrun Error interrupt mask
UART_IT_BreakError	Break Error interrupt mask
UART_IT_ParityError	Parity Error interrupt mask
UART_IT_FrameError	Frame Error interrupt mask
UART_IT_ReceiveTimeOut	Receive Time Out interrupt mask
UART_IT_Transmit	Transmit interrupt mask
UART_IT_Receive	Receive interrupt mask
UART_IT_DSR	DSR interrupt mask
UART_IT_DSD	DSD interrupt mask
UART_IT_CTS	CTS interrupt mask
UART_IT_RI	RI interrupt mask

Example:

```
/* Enables the UART0 transmit and receive interrupts */
UART_ITConfig(UART0, UART_IT_Transmit | UART_IT_Receive, ENABLE);
```

15.2.6 UART_DMAConfig

Function Name	UART_DMAConfig
Function Prototype	`void UART_DMAConfig(UART_TypeDef* UARTx, u16 UART_DMAOnError)`
Behavior Description	Configures the UARTx DMA interface.
Input Parameter1	UARTx: where x can be 1 or 2 to select the UART peripheral.
Input Parameter2	UART0_DMAOnError: specifies the DMA on error request. Refer to section "*UART_DMAOnError on page 196*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

UART_DMAOnError

To select whether the DMA is enabled/disabled when an error ocurrs, use one of the following values:

UART_DMAOnError	Meaning
UART_DMAOnError_Enable	DMA receive request enabled when the UART error interrupt is asserted.
UART_DMAOnError_Disable	DMA receive request disabled when the UART error interrupt is asserted.

Example:

```
/* DMA configuration */
UART_DMAConfig(UART1, UART_DMAOnError_Enable);
```

15.2.7 UART_DMACmd

Function Name	UART_DMACmd
Function Prototype	`void UART_DMACmd(UART_TypeDef* UARTx, u8 UART_DMAReq, FunctionalState NewState)`
Behavior Description	Enables or disables the UARTx's DMA interface.
Input Parameter1	UARTx: where x can be 1 or 2 to select the UART peripheral.
Input Parameter2	UART_DMAReq: specifies the DMA request. Refer to section "*UART_DMAReq on page 196*" for more details on the allowed values of this parameter.
Input Parameter3	NewState: new state of the UARTx's DMA request. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

UART_DMAReq

To select the DMA request to be enabled/disabled, use one of the following values:

UART_DMAReq	Meaning
UART_DMAReq_Tx	Transmit DMA request
UART_DMAReq_Rx	Receive DMA request

Example:

```
/* Enable the DMA transfer on Rx action */
  UART_DMACmd(UART1, ENABLE);
```

15.2.8 UART_LoopBackConfig

Function Name	UART_LoopBackConfig
Function Prototype	`void UART_LoopBackConfig(UART_TypeDef* UARTx, FunctionalState NewState)`
Behavior Description	Enables or disables the UARTx's LoopBack mode.
Input Parameter1	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Input Parameter2	NewState: new state of the UARTx's LoopBack mode. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* ENABLE the UART1 LoopBack mode */
UART_LoopBackConfig(UART1, ENABLE);
```

15.2.9 UART_IrDALowPowerConfig

Function Name	UART_IrDALowPowerConfig
Function Prototype	`void UART_IrDALowPowerConfig(u8 IrDAx, FunctionalState NewState)`
Behavior Description	Sets the IrDA low power mode.
Input Parameter1	IrDAx: where x can be 0,1 or 2 to select the IrDA.
Input Parameter2	NewState: new state of the UARTx's IrDA interface. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable IrDA0 in low power*/
UART_IrDALowPowerConfig(IrDA0, ENABLE);
```

15.2.10 UART_IrDACmd

Function Name	UART_IrDACmd
Function Prototype	`void UART_IrDACmd(u8 IrDAx, FunctionalState NewState)`
Behavior Description	Enables or disables the IrDAx interface.
Input Parameter1	IrDAx: where x can be 0, 1 or 2 to select the IrDA..
Input Parameter2	NewState: new state of the IrDAx's interface. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the IrDA0 interface */
UART_IrDACmd(IrDA0, ENABLE);
```

15.2.11 UART_IrDASetCounter

Function Name	UART_IrDASetCounter
Function Prototype	`void UART_IrDASetCounter(u8 IrDAx, u32 IrDA_Counter)`
Behavior Description	Sets the IrDA counter divisor value in low power mode.
Input Parameter1	IrDAx: where x can be 0, 1 or 2 to select the IrDA..
Input Parameter2	IrDA_Counter:sets the IrDA counter divisor value(Hz).
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets IrDA0 counter */
UART_IrDASetCounter(IrDA0, 1420000);
```

15.2.12 UART_SendData

Function Name	UART_SendData
Function Prototype	`void UART_SendData(UART_TypeDef* UARTx, u8 Data)`
Behavior Description	Transmits a byte of data through the UARTx peripheral.
Input Parameter1	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Input Parameter2	Data: the byte to transmit.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Send one byte on UART1 */
UART_SendData(UART1,0x22);
```

15.2.13 UART_ReceiveData

Function Name	UART_ReceiveData
Function Prototype	`vu8 UART_ReceiveData(UART_TypeDef* UARTx)`
Behavior Description	Returns the most recent byte received by the UARTx peripheral.
Input Parameter	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Output Parameter	None
Return Parameter	The received data.
Required preconditions	None
Called functions	None

Example:

```
/* Receive one byte on UART2 */
u8 RxData;
RxData = UART_ReceiveData(UART2);
```

15.2.14 UART_SendBreak

Function Name	UART_SendBreak
Function Prototype	`void UART_SendBreak(UART_TypeDef* UARTx)`
Behavior Description	Transmits break characters.
Input Parameter	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Send break character on UART1 */
UART_SendBreak(UART1);
```

15.2.15 UART_RTSConfig

Function Name	UART_RTSConfig
Function Prototype	`void UART_RTSConfig(UART_LevelTypeDef LevelState)`
Behavior Description	Sets or resets the RTS signal (for UART0 only).
Input Parameter2	LevelState: new state of the RTS signal. This parameter can be: LowLevel or HighLevel.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set the UART0 RTS signal */
UART_RTSConfig(HighLevel);
```

15.2.16 UART_DTRConfig

Function Name	UART_DTRConfig
Function Prototype	`void UART_RTSConfig(UART_LevelTypeDef LevelState)`
Behavior Description	Sets or resets the DTR signal (for UART0 Only).
Input Parameter2	LevelState: new state of the DTR signal. This parameter can be: LowLevel or HighLevel.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set the UART0 DTR signal */
UART_DTRConfig(HighLevel);
```

15.2.17 UART_GetFlagStatus

Function Name	UART_GetFlagStatus
Function Prototype	`FlagStatus UART_GetFlagStatus(UART_TypeDef* UARTx, u16 UART_FLAG)`
Behavior Description	Checks whether the specified UART flag is set or not.
Input Parameter1	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Input Parameter2	UART_FLAG: specifies the flag to check. Refer to section "*UART_FLAG on page 202*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of UART_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

UART_FLAG

The UART flags that can be read are listed in the following table:

UART_FLAG	Meaning
UART_FLAG_OverrunError	Overrun error flag
UART_FLAG_Break	Break error flag
UART_FLAG_ParityError	Parity error flag
UART_FLAG_FrameError	Frame error flag
UART_FLAG_RI	Ring indicator flag
UART_FLAG_TxFIFOEmpty	Transmit FIFO Empty flag
UART_FLAG_RxFIFOFull	Receive FIFO Full flag
UART_FLAG_TxFIFOFull	Transmit FIFO Full flag
UART_FLAG_RxFIFOEmpty	Receive FIFO Empty flag
UART_FLAG_Busy	Busy flag
UART_FLAG_DCD	DCD flag
UART_FLAG_DSR	DSR flag
UART_FLAG_CTS	CTS flag
UART_RawIT_OverrunError	Overrun Error interrupt flag
UART_RawIT_BreakError	Break Error interrupt flag
UART_RawIT_ParityError	Parity Error interrupt flag
UART_RawIT_FrameError	Frame Error interrupt flag
UART_RawIT_ReceiveTimeOut	ReceiveTimeOut interrupt flag
UART_RawIT_Transmit	Transmit interrupt flag
UART_RawIT_Receive	Receive interrupt flag
UART_RawIT_DSR	DSR interrupt flag
UART_RawIT_DCD	DCD interrupt flag
UART_RawIT_CTS	CTS interrupt flag
UART_RawIT_RI	RI interrupt flag

Example:

```
/* Check if the transmit FIFO is full or not */
FlagStatus Status;
Status = UART_GetFlagStatus(UART0, UART_TxFIFOFull);
```

15.2.18 UART_ClearFlag

Function Name	UART_ClearFlag
Function Prototype	`void UART_ClearFlag(UART_TypeDef* UARTx)`

Behavior Description	Clears the UARTx pending flags.
Input Parameter1	UARTx: where x can be 0, 1or 2 to select the UART peripheral.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear flag */
UART_ClearFlag(UART0);
```

15.2.19 UART_GetITStatus

Function Name	UART_GetITStatus
Function Prototype	`ITStatus UART_GetITStatus(UART_TypeDef* UARTx, u16 UART_IT)`
Behavior Description	Checks whether the specified UART interrupt has occurred or not.
Input Parameter1	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Input Parameter2	UART_IT: specifies the interrupt source to check. Refer to section "*UART_IT on page 195*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of UART_IT (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Get the UART0 Overrun Error interrupt status */
ITStatus OverrunITStatus;
OverrunITStatus = UART_GetITStatus(UART0, UART_IT_OverrunError);
```

15.2.20 UART_ClearITPendingBit

Function Name	UART_ClearITPendingBit
Function Prototype	`void UART_ClearITPendingBit(UART_TypeDef* UARTx, u16 UART_IT)`
Behavior Description	Clears the UARTx interrupt pending bits.
Input Parameter1	UARTx: where x can be 0,1or 2 to select the UART peripheral.
Input Parameter2	UART_IT: specifies the interrupt pending bit to clear. More than one interrupt can be cleared using the "|" operator. Refer to section "*UART_IT on page 195*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None

Required preconditions	None
Called functions	None

Example:

```
/* Clear the Overrun error interrupt pending bit */
UART_ClearITPendingBit(UART0,UART_IT_OverrunError);
```

16 I²C Interface Module (I2C)

The I²C Bus interface module serves as interface between the microcontroller and the serial I²C bus. It provides both multi-master and slave functions, and controls all I²C bus-specific sequencing, protocol, arbitration and timing.

16.1 I²C register structure

The I²C register structure *I2C_TypeDef* is defined in the ***91x_map.h*** file as follows:

```
typedef struct
{
vu8  CR;                  /* Control Register                */
u8  EMPTY1[3];
vu8  SR1;                 /* Status Register 1               */
u8  EMPTY2[3];
vu8  SR2;                 /* Status Register 2               */
u8  EMPTY3[3];
vu8  CCR;                 /* Clock Control Register          */
u8  EMPTY4[3];
vu8  OAR1;                /* Own Address Register 1          */
u8  EMPTY5[3];
vu8  OAR2;                /* Own Address Register 2          */
u8  EMPTY6[3];
vu8  DR;                  /* Data Register                   */
u8  EMPTY7[3];
vu8  ECCR;                /* Extended Clock Control Register */
u8  EMPTY8[3];
} I2C_TypeDef;
```

The following table presents the I²C registers:

Register	Description
I2C_CR	Used to configure I²C mode operation
I2C_SR1	Used to check the I²C bus status
I2C_SR2	Used to check the I²C bus status
I2C_CCR	I²C clock control register
I2C_OAR1	Used to define the I²C bus address of the bus
I2C_OAR2	Used to define the I²C bus address of the bus (Bit 8 and 9) , and specify I²C bus setup/hold time
I2C_DR	This register contains the byte to be received or transmitted on the bus
I2C_ECCR	Specify the upper 5 bits of the 11-bit clock divider

The two I²C interfaces are declared in the same file:

```
#ifndef EXT
 #Define EXT extern
#endif
...
#define AHB_APB_BRDG1_U    (0x5C000000) /* AHB/APB Bridge 1 UnBuffered Space */
#define AHB_APB_BRDG1_B    (0x4C000000) /* AHB/APB Bridge 1 Buffered Space   */
...
#define APB_I2C0_OFST       (0x00007000) /* Offset of I2C0 */
#define APB_I2C1_OFST       (0x00008000) /* Offset of I2C1 */
...
#ifndef Buffered
#define AHBAPB1_BASE           (AHB_APB_BRDG1_U)
...
#else /* Buffered */
...
#define AHBAPB1_BASE           (AHB_APB_BRDG1_B)

/* I2C Base Address definition*/
#define I2C0_BASE          (AHBAPB1_BASE + APB_I2C0_OFST)
#define I2C1_BASE          (AHBAPB1_BASE + APB_I2C1_OFST)
...
/* I2C peripheral declaration*/

#ifndef DEBUG
...
#define I2C0    ((I2C_TypeDef *) I2C0_BASE)
#define I2C1    ((I2C_TypeDef *) I2C1_BASE)

...
#else
...
EXT I2C_TypeDef           *I2C0;
EXT I2C_TypeDef           *I2C1;
...

#endif
```

When debug mode is used, I²C pointers are initialized in ***91x_lib.c*** file:

```
#ifdef _I2C0
I2C0 = (I2C_TypeDef *)I2C0_BASE;
#endif /*_I2C0 */

#ifdef _I2C1
I2C1 = (I2C_TypeDef *)I2C1_BASE;
#endif /*_I2C1 */
```

_I2C, _I2C0 and _I2C1 must be defined, in ***91x_conf.h*** file, to access the peripheral registers as follows:

```
#define _I2C
#define _I2C0
#define _I2C1
...
```

16.2 Software library functions

The following table enumerates the different functions of the I²C library.

Function Name	Description
I2C_DeInit	Deinitializes the I2Cx peripheral registers to their default reset values.
I2C_Init	Initializes the I2Cx peripheral according to the specified parameters in the I2C_InitStruct.
I2C_StructInit	Fills each I2C_InitStruct member with its reset value.
I2C_Cmd	Enables or disables the I²C peripheral.
I2C_GenerateSTART	Generates I²C communication START condition.
I2C_GenerateSTOP	Generates I²C communication STOP condition.
I2C_AcknowledgeConfig	Enables or disables I²C acknowledge feature.
I2C_ITConfig	Enables or disables I²C interrupt.
I2C_ReadRegister	Reads any I²C register and return its value.
I2C_GetFlagStatus	Checks whether a I²C flag is set or not. The tested flag is passed as parameter of the function.
I2C_ClearFlag	Clears the specified I²C flag passed as parameter.
I2C_Send7bitAddress	Transmits the address byte to select the slave device.
I2C_SendData	Sends a data byte.
I2C_ReceiveData	Reads the received byte.
I2C_GetLastEvent	Gets the last I²C event that has occurred.
I2C_CheckEvent	Checks if the last occured event is equal to the one passed as parameter.

16.2.1 I2C_DeInit

Function Name	I2C_DeInit
Function Prototype	`void I2C_DeInit(I2C_TypeDef* I2Cx)`
Behavior Description	Resets all I²C registers to their default values.
Input Parameter	I²Cx: where x can be 0,1, to select the I²C peripheral.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	SCU_APBPeriphReset()

Example:

```
/* Deinitialize the I2C0 peripheral */
I2C_DeInit (I2C0);
```

16.2.2 I2C_Init

Function Name	I2C_Init
Function Prototype	`void I2C_Init(I2C_TypeDef* I2Cx, I2C_InitTypeDef* I2C_InitStruct)`
Behavior Description	Configures the selected I^2C according to the choosen mode by writing the corresponding value to the I^2C registers.
Input Parameter1	I2Cx: where x can be 0,1 to select the I^2C peripheral to configure.
Input Parameter2	I2C_InitStruct: pointer to a I2C_InitTypeDef structure that contains the configuration information for the specified I^2C peripheral. Refer to section "*I2C_InitTypeDef on page 208*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

I2C_InitTypeDef

The I2C_InitTypeDef structure is defined in the **91x_i2c.h** file:

```
typedef struct
typedef struct
{
   u32 I2C_CLKSpeed;
   u16 I2C_OwnAddress;
   u8  I2C_GeneralCall;
   u8  I2C_Ack;
}I2C_InitTypeDef;
```

I2C_Ack

Enable/Disable the I^2C acknowledgement feature.

I2C_Ack	Meaning
I2C_Ack_Enable	Enable the Acknowledgement featue
I2C_Ack_Disable	Disable the Acknowledgement feature

I2C_OwnAddress

Select the device own address. It can be a 7-bit or 10-bit address.

I2C_GeneralCall

Enables/disables the General call feature. This member can be one of the following values:

I2C_GeneralCall	Meaning
I2C_GeneralCall_Enable	Enable the General call feature
I2C_GeneralCall_Disable	Disable the General call feature

I2C_ClkSpeed

Select the clock speed frequency, value must be under 400000

Example:

```
/* Initialize the I2C peripheral according to the I2C_InitStructure members */
I2C_InitTypeDef I2C_InitStructure;

I2C_InitStructure.I2C_OwnAddress = 0xA8;
I2C_InitStructure.I2C_CLKSpeed = 100000;
I2C_InitStructure.I2C_GeneralCall = I2C_GeneralCall_Disable ;
I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
I2C_Init(&I2C_InitStructure);
```

16.2.3 I2C_StructInit

Function Name	I2C_StructInit
Function Prototype	`void I2C_StructInit(I2C_InitTypeDef* I2C_InitStruct)`
Behavior Description	Fills each I2C_InitStruct member with its reset value.
Input Parameter	I2C_InitStruct: pointer to a I2C_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

The I2C_InitStruct members have the following default values:

Member	Default value
I2C_CLKSpeed	5000
I2C_OwnAddress	0
I2C_GeneralCall	I2C_GeneralCall_Disable
I2C_Ack	I2C_Ack_Disable

Example:

```
/* Initialize a I2C_InitTypeDef structure */
I2C_InitTypeDef I2C_InitStructure;
I2C_StructInit(&I2C_InitStructure);"
```

16.2.4 I2C_Cmd

Function Name	I2C_Cmd
Function Prototype	`void I2C_Cmd(I2C_TypeDef* I2Cx, FunctionalState NewState)`
Behavior Description	Enables or disables the I^2C peripheral.
Input Parameter1	I2Cx: where x can be 0,1, to select the I^2C peripheral.
Input Parameter2	NewState: new state of the I2Cx peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* To enable I2C0 */
I2C_Cmd (I2C0, ENABLE);
/* To disable I2C1 */
I2C_Cmd (I2C1, DISABLE);
```

16.2.5 I2C_GenerateSTART

Function Name	I2C_GenerateSTART
Function Prototype	`void I2C_GenerateStart(I2C_TypeDef* I2Cx, FunctionalState NewState)`
Behavior Description	Enables or disables the I^2C start generation.
Input Parameter1	I2Cx: where x can be 0,1 to select the PPP peripheral.
Input Parameter2	NewState: specifies whether the start generation is Enabled or Disbaled. This parameter can be ENABLE or DISABLE
Output Parameter	None
Return Parameter	None
Required preconditions	The specified I^2C peripheral must be enabled
Called functions	None

Example:

```
/*To enable the start generation for the I2C0*/
I2C_GenerateSTART (I2C0, ENABLE)
/*To disable the start generation for the I2C1*/
I2C_GenerateSTART (I2C1, DISABLE)
```

16.2.6 I2C_GenerateSTOP

Function Name	I2C_GenerateSTOP
Function Prototype	`void I2C_GenerateSTOP(I2C_TypeDef* I2Cx, FunctionalState NewState)`
Behavior Description	Enables or disables the STOP condition generation
Input Parameter 1	I2Cx : specifies the I^2C to be configured
Input Parameter 2	NewState: specifies whether the stop generation is Enabled or Disbaled. This parameter can be ENABLE or DISABLE
Output Parameter	STOP bit in the I2C_CR is modified according to the NewState parameter
Return Parameter	None
Required preconditions	The specified I^2C peripheral must be enabled
Called functions	None

Example:

```
/*To enable the STOP condition generation for the I2C0*/
I2C_GenerateSTOP (I2C0, ENABLE);
/*To disable the STOP condition generation for the I2C1 /
I2C_GenerateSTOP (I2C1, DISABLE);
```

16.2.7 I2C_AcknowledgeConfig

Function Name	I2C_AcknowledgeConfig
Function Prototype	`void I2C_AcknowledgeConfig(I2c_TypeDef *I2Cx, FunctionalState NewState)`
Behavior Description	Enables or disables the I^2C acknowlegement
Input Parameter 1	I2Cx: specifies the I^2C to be configured
Input Parameter 2	NewState: specifies whether the acknowledgement is enabled or disabled. This parameter can be ENABLE or DISABLE
Output Parameter	None
Return Parameter	None
Required preconditions	The specified I^2C peripheral must be enabled
Called functions	None

Example:

```
/*To enable the acknowledgement feature for the I2C0*/
I2C_AcknowledgeConfig (I2C0, ENABLE);
/*To disable the acknowledgement feature for the I2C1*/
I2C_AcknowledgeConfig (I2C1, DISABLE);
```

16.2.8 I2C_ITConfig

Function Name	I2C_ITConfig
Function Prototype	`void I2C_ITConfig(I2C_TypeDef *I2Cx, FunctionalState NewState)`
Behavior Description	Enables or disables I^2C interrupt feature
Input Parameter1	I2Cx: where x can be 0,1to select the I2C peripheral.
Input Parameter2	NewState: specifies whether the I2C interrupt is enabled or disabled. This parameter can be ENABLE or DISABLE
Output Parameter	ITE bit in the I2C_CR is modified according to condition parameter
Return Parameter	None
Required preconditions	The specified I^2C peripheral must be enabled
Called functions	None

Example:

```
/*To enable the interrupt feature for the I2C0*/
I2C_ITConfig (I2C0, Enable);
/*To disable the interrupt feature for the I2C1*
I2C_ITConfig (I2C1, Enable);
```

16.2.9 I2C_ReadRegister

Function Name	I2C_ReadRegister
Function Prototype	`u8 I2C_ReadRegister(I2C_TypeDef* I2Cx, u8 I2C_Register)`
Behavior Description	Reads any I^2C register and returns its value.
Input Parameter1	I2Cx: where x can be 0,1to select the I^2C peripheral.
Input Parameter2	I2C_Register: specifies the register to be read. Refer to section *I2C Registers: on page 213*" for more details on the allowed values of this parameter.
Output Parameter	Access to register to be read
Return Parameter	None
Required preconditions	...
Called functions	None

Example:

```
/*Read the I2C_CR Register*/
u8 RegisterValue;
RegisterValue = I2C_RegisterRead (I2C0, I2C_CR);
```

I²C Registers:

The I²C registers are listed in the following table:

I²C registers	Meaning
I2C_CR	I2C_CR selected for read
I2C_SR1	I2C_SR1 selected for read
I2C_SR2	I2C_SR2 selected for read
I2C_CCR	I2C_CCR selected for read
I2C_OAR1	I2C_OAR1 selected for read
I2C_OAR2	I2C_OAR2 selected for read
I2C_DR	I2C_DR selected for read
I2C_ECCR	I2C_ECCR selected for read

16.2.10 I2C_GetFlagStatus

Function Name	I2C_GetFlagStatus
Function Prototype	`Flagstatus I2C_GetFlagStatus(I2C_TypeDef* I2Cx, u16 I2C_FLAG)`
Behavior Description	Checks whether the I2C flag is set or not.
Input Parameter 1	I2Cx: where x can be 0,1 to select the I^2C peripheral.
Input Parameter 2	I2C_Flag: specifies the flag to check Refer to section *I2C Flags on page 214* for more details about the allowed values of the I2C_Flag parameter.
Output Parameter	Access to register containing the flag to be checked.
Return Parameter	FlagStatus: the status of the specified I2C_Flag. It can be either: SET: if the tested flag is set RESET: if the tested flag is reset
Required preconditions	None
Called functions	None

Example:

```
/*Get the I2C_FLAG_AF status */
FlagStatus Status;
Status = I2C_GetFlagStatus (I2C0, I2C_FLAG_AF);
```

I2C Flags

The I2C flags are listed in the following table:

I2C_FLAG	Meaning
I2C_FLAG_SB	Start (Master mode) flag
I2C_FLAG_M_SL	Master/Slave flag
I2C_FLAG_ADSL	Address matched (Slave mode) flag
I2C_FLAG_BTF	Byte transfer finished flag
I2C_FLAG_BUSY	Bus busy flag
I2C_FLAG_TRA	Transmitter/Receiver flag
I2C_FLAG_ADD10	10-bit addressing in master mode flag
I2C_FLAG_EVF	Event flag
I2C_FLAG_GCAL	General call (slave mode) flag
I2C_FLAG_BERR	Bus error flag
I2C_FLAG_ARLO	Arbitration lost flag
I2C_FLAG_STOPF	Stop detection (slave mode) flag
I2C_FLAG_AF	Acknowledge failure flag
I2C_FLAG_ENDAD	End of address transmission flag
I2C_FLAG_ACK	Acknowledge enabled flag

16.2.11 I2C_ClearFlag

Function Name	I2C_ClearFlag
Function Prototype	`void I2C_ClearFlag(I2C_TypeDef* I2Cx, u16 I2C_FLAG, ...)`
Behavior Description	Clears the I^2C pending flags.
Input Parameter1	I2Cx: where x can be 0,1 to select the I^2C peripheral.
Input Parameter2	I2C_Flag: specifies the flag to be cleared. Refer to section '*I2C Flags on page 214*' values for more details on allowed values for this parameter
Input Parameter3	Parameter needed in cases where a write to a register is needed to clear the flag.
Output Parameter	Some flags may be cleared when the register is read.
Return Parameter	None
Required preconditions	None
Called functions	I2C_Cmd

Example:

```
/*Clears I2C_FLAG_STOPF flag */
I2C_ClearFlag(I2C0, I2C_FLAG_STOPF);
```

16.2.12 I2C_Send7bitAddress

Function Name	I2C_Send7bitAddress
Function Prototype	`void I2C_Send7bitAddress(I2C_TypeDef* I2Cx, u8 Address, u8 Direction)`
Behavior Description	Transmits the address byte to select the slave device.
Input Parameter1	I2Cx: where x can be 0,1 to select the I^2C peripheral.
Input Parameter2	Address: specifies the slave address which will be transmitted
Input Parameter3	Direction: specifies whether the I^2C device will be a Transmitter or a Receiver. This parameter could be: I2C_MODE_TRANSMITTER: Transmitter mode. I2C_MODE_RECEIVER: Receiver mode.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Send from I2C0, as transmitter, the Slave device address 0xA8 in 7-bit addressing
mode */
I2C_Send7bitAddress(I2C0, 0xA8, I2C_MODE_TRANSMITTER);
```

16.2.13 I2C_SendData

Function Name	I2C_SendData
Function Prototype	`void I2C_SendData(I2C_TypeDef* I2Cx, u8 bData)`
Behavior Description	Transmits a single byte
Input Parameter1	I2Cx: where x can be 0,1 to select the I^2C peripheral.
Input Parameter2	bData: indicates the data to be transmitted (1 byte)
Output Parameter	I2C_DR register value will be modified
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/*Transmits the byte 0xAF through the I2C1*/
I2C_SendData (I2C1, 0xAF);
```

16.2.14 I2C_ReceiveData

Function Name	I2C_ReceiveData
Function Prototype	`u8 I2C_ReceiveData(I2C_TypeDef* I2Cx)`
Behavior Description	Returns the most recent received byte
Input Parameter1	I2Cx: where x can be 0,1 to select the I^2C peripheral.
Output Parameter	None
Return Parameter	I2C_DR register Value
Required preconditions	None
Called functions	None

Example:

```
/* I2C0 Receives a single byte */
u8 bByteReceived;
bByteReceived = I2C_ReceiveData(I2C0);
```

16.2.15 I2C_GetLastEvent

Function Name	I2C_GetLastEvent
Function Prototype	`u16 I2C_GetLastEvent(I2C_TypeDef* I2Cx)`
Behavior Description	Gets the last occurred I^2C event.
Input Parameter1	I2Cx: where x can be 0,1 to select the I^2C peripheral.
Output Parameter	None
Return Parameter	The last occurred event.
Required preconditions	None
Called functions	None

Example:

```
/*Get the I2C0 peripheral status*/
u16 LastEvent;
LastEvent=I2C_GetLastEvent (I2C0);
```

16.2.16 I2C_CheckEvent

Function Name	I2C_CheckEvent
Function Prototype	`ErrorStatus I2C_CheckEvent(I2C_TypeDef* I2Cx, u16 I2C_Event`
Behavior Description	Checks whether the last I²C event is equal to the one passed as parameter
Input Parameter1	I2Cx: where x can be 0,1 to select the I²C peripheral
Input Parameter2	I2C_EVENT: specifies the event to be checked. Refer to section *I2C Events* for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	An ErrorStatus enumeration value: SUCCESS: Last event is equal to the I2C_Event ERROR: Last event is different from the I2C_Event
Required preconditions	None
Called functions	I2C_GetLastEvent()

Example:

```
/*Checks if the last event is equal to I2C_EVENT_MASTER_BYTE_RECEIVED*/
I2C_CheckEvent (I2C0, I2C_EVENT_MASTER_BYTE_RECEIVED)
```

I2C Events

The I²C events are listed in the following table:

I2C_EVENT	Meaning
I2C_EVENT_SLAVE_ADDRESS_MATCHED	EV1
I2C_EVENT_SLAVE_BYTE_RECEIVED	EV2
I2C_EVENT__SLAVE_BYTE_TRANSMITTED	EV3
I2C_EVENT_SLAVE_ACK_FAILURE	EV4
I2C_EVENT_MASTER_MODE_SELECT	EV5
I2C_EVENT_MASTER_MODE_SELECTED	EV6
I2C_EVENT_MASTER_BYTE_RECEIVED	EV7
I2C_EVENT_MASTER_BYTE_TRANSMITTED	EV8
I2C_EVENT_MASTER_MODE_ADDRESS10	EV9
I2C_EVENT_SLAVE_STOP_DETECTED	EV3-1

17 3-phase induction motor controller (MC)

The MC controller is designed for variable speed motor control applications. Three PWM outputs are available for controlling a three-phase motor drive. Rotor speed feedback is provided by capturing a tachogenerator input signal.

The first section describes the data structures used in the MC software library. The second one presents the software library functions.

17.1 MC register structure

The MC register structure *MC_TypeDef* is defined in the ***91x_map.h*** file as follows:

```
typedef struct
{
vu16 TCPT;
u16 EMPTY1;
vu16 TCMP;
u16 EMPTY2;
vu16 IPR;
u16 EMPTY3;
vu16 TPRS;
u16 EMPTY4;
vu16 CPRS;
u16 EMPTY5;
vu16 REP;
u16 EMPTY6;
vu16 CMPW;
u16 EMPTY7;
vu16 CMPV;
u16 EMPTY8;
vu16 CPMU;
u16 EMPTY9;
vu16 CMP0;
u16 EMPTY10;
vu16 PCR0;
u16 EMPTY11;
vu16 PCR1;
u16 EMPTY12;
vu16 PCR2;
u16 EMPTY13;
vu16 PSR;
u16 EMPTY14;
vu16 OPR;
u16 EMPTY15;
vu16 IMR;
u16 EMPTY16;
vu16 DTG;
u16 EMPTY16;
vu16 ESC;
u16 EMPTY16;
} MC_TypeDef;
```

The following table presents the MC registers:

Register	Description
TCPT	Tacho capture register
TCMP	Tacho compare register
IPR	Interrupt pending register
TPRS	Tacho prescaler register
CPRS	PWM counter prescaler register
REP	Repetition counter register
CMPW	Compare phase W preload register
CMPV	Compare phase V preload register
CMPU	Compare phase U preload register
CMP0	Compare 0 preload register
PCR0	Peripheral control register 0
PCR1	Peripheral control register 1
PCR2	Peripheral control register 2
PSR	Polarity selection register
OPR	Output peripheral register
IMR	Interrupt mask register
DTG	Dead time generator register
ESC	Emergency stop clear register

The MC is declared in the file below

```
#ifndef EXT
 #Define EXT extern
#endif
...
#define AHB_APB_BRDG1_U    (0x5C000000) /* AHB/APB Bridge 1 UnBuffered Space */
#define AHB_APB_BRDG1_B    (0x4C000000) /* AHB/APB Bridge 1 Buffered Space   */
...
#define APB_MC_OFST       (0x00003000) /* Offset of MC */
...
#ifndef Buffered
#define AHBAPB1_BASE           (AHB_APB_BRDG1_U)
...
#else /* Buffered */
...
#define AHBAPB1_BASE           (AHB_APB_BRDG1_B)

/* MC Base Address definition*/
#define MC_BASE          (AHBAPB1_BASE + APB_MC_OFST)


...
/* MC peripheral declaration*/

#ifndef DEBUG
...
```

```
#define MC          ((MC_TypeDef *) MC_BASE)
...
#else
...
EXT  MC_TypeDef            *MC;
...

#endif
```

When debug mode is used, the MC pointer is initialized in the ***91x_lib.c*** file:

```
#ifdef _MC
  MC = (MC_TypeDef *)MC_BASE
#endif /* _MC */
```

_MC must be defined, in ***91x_conf.h*** file, to access the peripheral registers as follows:

```
#define _MC
```

17.2 Software library functions

The following table enumerates the different functions of the MC library.

Function Name	Description
MC_DeInit	Deinitializes the MC peripheral registers to their default reset values.
MC_Init	Initializes the MC peripheral according to the specified parameters in the MC_InitStruct.
MC_StructInit	Fills each MC_InitStruct member with its default value.
MC_Cmd	Enables or disables the MC peripheral.
MC_ClearPWMCounter	Clears the MC PWM Counter.
MC_ClearTachoCounter	Clears the MC Tacho Counter.
MC_CtrlPWMOutputs	Enables or disables MC peripheral Main Outputs.
MC_ITConfig	Enables or disables the MC interrupts.
MC_SetPrescaler	Sets the MC PWM prescaler value.
MC_SetPeriod	Sets the MC period value.
MC_SetPulseU	Sets the PWM pulse U value.
MC_SetPulseV	Sets the PWM pulse V value.
MC_SetPulseW	Sets the PWM pulse W value.
MC_SetTachoCompare	Sets the MC Tacho compare value.
MC_PWMModeConfig	Selects the MC PWM Counter Mode.
MC_SetDeadTime	Sets the MC dead time value.
MC_EmergencyCmd	Enables or disables the MC emergency feature.
MC_EmergencyClear	Clears the MC emergency register.
MC_GetPeriod	Gets the MC period value.
MC_GetPulseU	Gets the MC pulse U value.
MC_GetPulseV	Gets the MC pulse V value.
MC_GetPulseW	Gets the MC pulse W value.
MC_GetTachoCapture	Gets the MC Tacho capture value.
MC_ClearOnTachoCapture	Enables or disables the Clear on Capture of Tacho counter.
MC_ForceDataTransfer	Sets the MC outputs default states.
MC_SoftwarePreloadConfig	Enables the software data transfer.
MC_SoftwareTachoCapture	Enables the software Tacho capture.
MC_GetCountingStatus	Checks whether the PWM counter is counting UP or DOWN.
MC_GetFlagStatus	Checks whether the specified MC flag is set or not.
MC_ClearFlag	Clears the MC pending flags.
MC_GetITStatus	Checks whether the specified MC interrupt is occurred or not.
MC_ClearITPendingBit	Clears the MC interrupt pending bits.

17.2.1 MC_DeInit

Function Name	MC_DeInit
Function Prototype	`void MC_DeInit(void)`
Behavior Description	Deinitializes the MC peripheral registers to their default reset values.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	SCU_APBPeriphReset()

Example:

```
/* Deinitializes the MC registers to their default reset value */
MC_DeInit ();
```

17.2.2 MC_Init

Function Name	MC_Init
Function Prototype	`void MC_Init(MC_InitTypeDef* MC_InitStruct)`
Behavior Description	Initializes the MC peripheral according to the specified parameters in the MC_InitStruct.
Input Parameter	MC_InitStruct: pointer to a MC_InitTypeDef structure that contains the configuration information for the MC peripheral. Refer to section "*: MC_InitTypeDef on page 223*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

MC_InitTypeDef

The MC_InitTypeDef structure is defined in the ***91x_mc.h*** file:

```
typedef struct
{
u16 MC_OperatingMode;
u16 MC_TachoMode;
u16 MC_TachoEventMode;
u8  MC_Prescaler;
u16 MC_TachoPrescaler;
u16 MC_PWMMode;
u16 MC_Complementary;
u8 MC_ForcedPWMState;
u16 MC_Emergency;
u16 MC_Period;
u8 MC_TachoPeriod;
u16 MC_Channel;
u16 MC_PulseU;
u16 MC_PulseV;
u16 MC_PulseW;
```

```
u16 MC_PolarityUL;
u16 MC_PolarityUH;
u16 MC_PolarityVL;
u16 MC_PolarityVH;
u16 MC_PolarityWL;
u16 MC_PolarityWH;
u16 MC_TachoPolarity;
u16 MC_DeadTime;
u8 MC_RepetitionCounter;
} MC_InitTypeDef;
```

MC_OperatingMode

Specifies the MC operating mode. This member can be one of the following values:

MC_Mode	Meaning
MC_HardwareOperating_Mode	Hardware operating Mode
MC_SoftwareOperating_Mode	Software operating Mode

MC_TachoMode

Specifies the MC Tacho mode. This member can be one of the following values:

MC_TachoMode	Meaning
MC_TachoOneShot_Mode	One Shot Tacho Mode
MC_TachoContinuous_Mode	Continuous Tacho Mode

MC_TachoEventMode

Specifies the MC Tacho mode. This member can be one of the following values:

MC_TachoEventMode	Meaning
MC_TachoEvent_Hardware_Mode	Tacho Hardware event Mode
MC_TachoEvent_Software_Mode	Tacho Software event Mode

MC_Prescaler

Specifies the Prescaler value to divide the MC Input clock. The clock of the PWM Counter is divided by MC_Prescaler + 1.

This member must be a number between 0x00 and 0xFF.

MC_TachoPrescaler

Specifies the Prescaler value to divide the Tacho Input clock. The clock of the Tacho Counter is divided by MC_TachoPrescaler + 1.

This member must be a number between 0x000 and 0xFFF.

MC_PWMMode

Specifies the MC PWM mode. This member can be one of the following values:

MC_PWMMode	Meaning
MC_PWMClassical_Mode	Classical PWM Mode
MC_PWMZeroCenterd_Mode	Zero Centered Mode

MC_Complementary

Enables or disables the complementary MC feature. This member can be one of the following values:

MC_Complementary	Meaning
MC_Complementary_Enable	MC Complementary Mode Enable.
MC_Complementary_Disable	MC Complementary Mode Disable.

MC_ForcedPWMState

Specifies the default PWM signal states. This member can be one of the following values:

MC_ForcedPWMState	Meaning
MC_Polarity_Inverted	PWM signal polarity inverted
MC_Polarity_NonInverted	PWM signal polarity non-inverted

MC_Emergency

Enables or disables the Emergency MC feature. This member can be one of the following values:

MC_Emergency	Meaning
MC_Emergency_Enable	MC Emergency Enable.
MC_Emergency_Disable	MC Emergency Disable.

MC_Period

Specifies the period value to be loaded in the active Auto-Reload Register at the next update event. This member must be a number between 0x0000 and 0xFFFF.

MC_TachoPeriod

Specifies the Tacho Compare period. This member must be a number between 0x00 and 0xFF.

MC_Channel

Specifies the MC Channel to be used. This member can be one of the following values:

MC_Channel	Meaning
MC_Channel_U	MC Channel U is used.
MC_Channel_V	MC Channel V is used.
MC_Channel_W	MC Channel W is used.
MC_Channel_ALL	MC Channel U, V and W are used.

MC_PulseU

Specifies the Pulse U value to be loaded in the CMPU Register.

The MC_PulseU presents the DutyCycle value. This member must be a number between 0x000 and 0x7FF.

MC_PulseV

Specifies the Pulse V value to be loaded in the CMPV Register.

The MC_PulseV presents the DutyCycle value. This member must be a number between 0x000 and 0x7FF.

MC_PulseW

Specifies the Pulse W value to be loaded in the CMPW Register.

The MC_PulseW presents the DutyCycle value. This member must be a number between 0x000 and 0x7FF.

MC_PolarityUL

Specifies the Channel UL signal Polarity. This member can be one of the following values:

MC_PolarityUL	Meaning
MC_Polarity_Inverted	Channel UL signal polarity inverted
MC_Polarity_NonInverted	Channel UL signal polarity non-inverted

MC_PolarityUH

Specifies the Channel UH signal Polarity. This member can be one of the following values:

MC_PolarityUH	Meaning
MC_Polarity_Inverted	Channel UH signal polarity inverted
MC_Polarity_NonInverted	Channel UH signal polarity non-inverted

MC_PolarityVL

Specifies the Channel VL signal Polarity. This member can be one of the following values:

MC_PolarityVL	Meaning
MC_Polarity_Inverted	Channel VL signal polarity inverted
MC_Polarity_NonInverted	Channel VL signal polarity non-inverted

MC_PolarityVH

Specifies the Channel VH signal Polarity. This member can be one of the following values:

MC_PolarityVH	Meaning
MC_Polarity_Inverted	Channel VH signal polarity inverted
MC_Polarity_NonInverted	Channel VH signal polarity non-inverted

MC_PolarityWL

Specifies the Channel WL signal Polarity. This member can be one of the following values:

MC_PolarityWL	Meaning
MC_Polarity_Inverted	Channel WL signal polarity inverted
MC_Polarity_NonInverted	Channel WL signal polarity non-inverted

MC_PolarityWH

Specifies the Channel WH signal Polarity. This member can be one of the following values:

MC_PolarityWH	Meaning
MC_Polarity_Inverted	Channel WH signal polarity inverted
MC_Polarity_NonInverted	Channel WH signal polarity non-inverted

MC_TachoPolarity

Specifies the Tacho Input signal Polarity. This member can be one of the following values:

MC_TachoPolarity	Meaning
MC_Polarity_Inverted	Tacho Input signal polarity inverted
MC_Polarity_NonInverted	Tacho Input signal polarity non-inverted

MC_DeadTime

Specifies the dead time for managing the time between the switching-off and the switching-on instants of the outputs.

MC_RepetitionCounter

Specifies the repetition counter value. Each time the RER down-counter reaches zero, an update event is generated and it restarts counting from RER value.

Example:

```
/* The following example illustrates how to configure the MC hardware operating
complementary Mode */
MC_InitStructure.MC_OperatingMode = MC_HardwareOperating_Mode;
MC_InitStructure.MC_TachoMode = MC_TachoContinuous_Mode;
MC_InitStructure.MC_Prescaler = 0x00;
MC_InitStructure.MC_TachoPrescaler = 0x000;
MC_InitStructure.MC_PWMMode = MC_PWMZeroCentered_Mode;
MC_InitStructure.MC_Complementary = MC_Complementary_Enable;
MC_InitStructure.MC_Emergency = MC_Emergency_Enable;
MC_InitStructure.MC_Period = 0x3FF;
MC_InitStructure.MC_TachoCompare = 0xFF;
MC_InitStructure.MC_Channel = MC_Channel_ALL;
MC_InitStructure.MC_PulseU = 0x1FF;
MC_InitStructure.MC_PulseV = 0xFF;
MC_InitStructure.MC_PulseW = 0x7F;
MC_InitStructure.MC_PolarityUL = MC_Polarity_Inverted;
MC_InitStructure.MC_PolarityUH = MC_Polarity_Inverted;
MC_InitStructure.MC_PolarityVL = MC_Polarity_NonInverted;
MC_InitStructure.MC_PolarityVH = MC_Polarity_NonInverted;
MC_InitStructure.MC_PolarityWL = MC_Polarity_Inverted;
MC_InitStructure.MC_PolarityWH = MC_Polarity_Inverted;
MC_InitStructure.MC_TachoPolarity = MC_TachoEventEdge_Falling;
MC_InitStructure.MC_DeadTime = 0x0F;
MC_InitStructure.MC_RepetitionCounter = 0x0;

MC_Init(&MC_InitStructure);
```

17.2.3 MC_StructInit

Function Name	MC_StructInit
Function Prototype	`void MC_StructInit(MC_InitTypeDef* MC_InitStruct)`
Behavior Description	Fills each MC_InitStruct member with its default value.
Input Parameter	MC_InitStruct: pointer to an MC_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* The following example illustrates how to initialize a MC_InitTypeDef structure */
MC_InitTypeDef MC_InitStructure;
MC_StructInit(&MC_InitStructure);
```

17.2.4 MC_Cmd

Function Name	MC_Cmd
Function Prototype	`void MC_Cmd(FunctionalState NewState)`
Behavior Description	Enables or disables the MC peripheral.
Input Parameter	NewState: new state of the MC peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the MC */
MC_Cmd(ENABLE);
```

17.2.5 MC_ClearPWMCounter

Function Name	MC_ClearPWMCounter
Function Prototype	`void MC_ClearPWMCounter(void)`
Behavior Description	Clears the MC PWM counter.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clears the PWM Counter*/
MC_ClearPWMCounter();
```

17.2.6 MC_ClearTachoCounter

Function Name	MC_ClearTachoCounter
Function Prototype	`void MC_ClearTachoCounter(void)`
Behavior Description	Clears the MC Tacho counter.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clears the Tacho Counter*/
MC_ClearTachoCounter();
```

17.2.7 MC_CtrlPWMOutputs

Function Name	MC_CtrlPWMOutputs
Function Prototype	`void MC_CtrlPWMOutputs(FunctionalState Newstate)`
Behavior Description	Enables or disables MC peripheral Main Outputs.
Input Parameter	NewState: new state of the MC peripheral outputs. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the MC Outputs */
MC_CtrlPWMOutputs(ENABLE);
```

17.2.8 MC_ITConfig

Function Name	MC_ITConfig
Function Prototype	`void MC_ITConfig(u16 MC_IT, FunctionalState NewState)`
Behavior Description	Enables or disables the MC interrupts.
Input Parameter1	MC_IT: specifies the MC interrupts sources to be enabled or disabled. Refer to section *: MC_IT on page 231* for more details on the allowed values of this parameter.
Input Parameter2	NewState: new state of the MC interrupts. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

MC_IT

To enable or disable MC interrupts, use a combination of one or more of the following values:

MC_IT	Meaning
MC_IT_CMPW	Compare W interrupt
MC_IT_CMPV	Compare V interrupt
MC_IT_CMPU	Compare U interrupt
MC_IT_ZPC	Zero of PWM counter interrupt
MC_IT_ADT	Automatic data transfer interrupt
MC_IT_OTC	Overflow of tacho counter interrupt
MC_IT_CPT	Capture of tacho counter interrupt
MC_IT_CM0	Compare 0 interrupt

Example:

```
/* Enables the MC Output Compare W Interrupt */
MC_ITConfig(MC_IT_CMPW, ENABLE);
```

17.2.9 MC_SetPrescaler

Function Name	MC_SetPrescaler
Function Prototype	`void MC_SetPrescaler(u8 MC_Prescaler)`
Behavior Description	Sets the MC prescaler value.
Input Parameter	MC_Prescaler: PWM prescaler new value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the MC new Prescaler value */

u8 MCPrescaler = 0xFF;
MC_SetPrescaler( MCPrescaler);
```

17.2.10 MC_SetPeriod

Function Name	MC_SetPeriod
Function Prototype	`void MC_SetPeriod(u16 MC_Period)`
Behavior Description	Sets the MC period value.
Input Parameter	MC_Period: MC period new value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the MC new Period value */

u16 MCPeriod = 0x3FF;
MC_SetPrescaler(MCPeriod);
```

17.2.11 MC_SetPulseU

Function Name	MC_SetPulseU
Function Prototype	`void MC_SetPulseU(u16 MC_PulseU)`
Behavior Description	Sets the MC pulse U value.
Input Parameter	MC_PulseU: MC pulse U new value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the MC new Channel U Pulse value */

u16 MCPulse = 0x00F0;
MC_SetPulseU(MCPulse);
```

17.2.12 MC_SetPulseV

Function Name	MC_SetPulseV
Function Prototype	`void MC_SetPulseV(u16 MC_PulseV)`
Behavior Description	Sets the MC pulse V value.
Input Parameter	MC_PulseV: MC pulse V new value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the MC new Channel V Pulse value */

u16 MCPulse = 0x00F0;
MC_SetPulseV(MCPulse);
```

17.2.13 MC_SetPulseW

Function Name	MC_SetPulseW
Function Prototype	`void MC_SetPulseW(u16 MC_PulseW)`
Behavior Description	Sets the MC pulse W value.
Input Parameter	MC_PulseW: MC pulse W new value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the MC new Channel W Pulse value */

u16 MCPulse = 0x00F0;
MC_SetPulseW(MCPulse);
```

17.2.14 MC_SetTachoCompare

Function Name	MC_SetTachoCompare
Function Prototype	`void MC_SetTachoCompare(u8 MC_Compare)`
Behavior Description	Sets the MC Tacho compare value.
Input Parameter	MC_Compare: MC Tacho compare value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the MC Tacho Compare value */

u8 MCTachoCompare = 0x0F;
MC_SetTachoCompare(MCTachoCompare);
```

17.2.15 MC_PWMModeConfig

Function Name	MC_PWMModeConfig
Function Prototype	`void MC_PWMModeConfig(u16 MC_PWMMode)`
Behavior Description	Configures the MC PWM Mode.
Input Parameter	MC_PWMMode: MC PWM Mode. Refer to section *: MC_PWMMode on page 225* for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Configures the MC to generate a Zero Centred PWM Mode */

MC_PWMModeConfig(MC_PWMZeroCenterd_Mode);
```

17.2.16 MC_SetDeadTime

Function Name	MC_SetDeadTime
Function Prototype	`void MC_SetDeadTime(u16 DeadTime)`
Behavior Description	Sets the MC Dead Time value.
Input Parameter	DeadTime: MC Dead Time value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the MC new Dead Time value */

u16 MCDeadTime = 0x00F;
MC_SetDeadTime(MCDeadTime);
```

17.2.17 MC_EmergencyCmd

Function Name	MC_EmergencyCmd
Function Prototype	`void MC_EmergencyCmd(FunctionalState NewState)`
Behavior Description	Enables or disables the MC emergency feature.
Input Parameter	NewState: new state of the MC peripheral Emergency Input. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the MC Emergency Input */

MC_EmergencyCmd( ENABLE);
```

17.2.18 MC_EmergencyClear

Function Name	MC_EmergencyClear
Function Prototype	`void MC_EmergencyClear(void)`
Behavior Description	Clears the MC emergency register.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clears the MC Emergency Register*/

MC_EmergencyClear();
```

17.2.19 MC_GetPeriod

Function Name	MC_GetPeriod
Function Prototype	`u16 MC_GetPeriod(void);`
Behavior Description	Gets the MC period value.
Input Parameter	None
Output Parameter	None
Return Parameter	MC period value.
Required preconditions	None
Called functions	None

Example:

```
/* Gets the MC Period value */

u16 MCPeriod = 0x000;
MCPeriod = MC_GetPeriod();
```

17.2.20 MC_GetPulseU

Function Name	MC_GetPulseU
Function Prototype	`u16 MC_GetPulseU(void);`
Behavior Description	Gets the MC pulse U value.
Input Parameter	None
Output Parameter	None
Return Parameter	MC pulse U value.
Required preconditions	None
Called functions	None

Example:

```
/* Gets the MC Channel U Pulse value */

u16 MCPulse = 0x000;
MCPulse = MC_GetPulseU();
```

17.2.21 MC_GetPulseV

Function Name	MC_GetPulseV
Function Prototype	`u16 MC_GetPulseV(void);`
Behavior Description	Gets the MC pulse V value.
Input Parameter	None
Output Parameter	None
Return Parameter	MC pulse V value.
Required preconditions	None
Called functions	None

Example:

```
/* Gets the MC Channel V Pulse value */

u16 MCPulse = 0x000;

MCPulse = MC_GetPulseV();
```

17.2.22 MC_GetPulseW

Function Name	MC_GetPulseW
Function Prototype	`u16 MC_GetPulseW(void);`
Behavior Description	Gets the MC pulse W value.
Input Parameter	None
Output Parameter	None
Return Parameter	MC pulse W value.
Required preconditions	None
Called functions	None

Example:

```
/* Gets the MC Channel W Pulse value */

u16 MCPulse = 0x000;
MCPulse = MC_GetPulseW();
```

17.2.23 MC_GetTachoCapture

Function Name	MC_GetTachoCapture
Function Prototype	`u16 MC_GetTachoCapture(void)`
Behavior Description	Gets the MC Tacho capture value.
Input Parameter	None
Output Parameter	None
Return Parameter	MC Tacho capture value.
Required preconditions	None
Called functions	None

Example:

```
/* Gets the MC Tacho Capture value */

u16 MCTachoCapture = 0x0000;
MCTachoCapture = MC_GetTachoCapture();
```

17.2.24 MC_ClearOnTachoCapture

Function Name	MC_ClearOnTachoCapture
Function Prototype	`void MC_ClearOnTachoCapture(void)`
Behavior Description	Enables or disables the clear on capture of Tacho counter.
Input Parameter	NewState: new state of the CCPT Bit. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the Clear on capture of Tacho Counter */
MC_ClearOnTachoCapture(ENABLE);
```

17.2.25 MC_ForceDataTransfer

Function Name	MC_ForceDataTransfer
Function Prototype	`void MC_ForceDataTransfer(u8 MC_ForcedData)`
Behavior Description	Sets the MC Outputs default states.
Input Parameter	MC_ForcedData: MC outputs new states.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the MC PWM forced state: all outputs on high level*/

MC_ForceDataTransfer(0x2F);
```

17.2.26 MC_SoftwarePreloadConfig

Function Name	MC_SoftwarePreloadConfig
Function Prototype	`void MC_SoftwarePreloadConfig(void)`
Behavior Description	Enables the software data transfer.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the software Data Transfer */
MC_SoftwarePreloadConfig();
```

17.2.27 MC_SoftwareTachoCapture

Function Name	MC_SoftwareTachoCapture
Function Prototype	`void MC_SoftwareTachoCapture(void)`
Behavior Description	Enables the software Tacho Capture.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the software Tacho Capture */
MC_SoftwareTachoCapture();
```

17.2.28 MC_GetCountingStatus

Function Name	MC_GetCountingStatus
Function Prototype	`CountingStatus MC_GetCountingStatus(void)`
Behavior Description	Checks whether the PWM Counter is counting Up or Down.
Input Parameter	None
Output Parameter	None
Return Parameter	The new state of the PWM Counter (DOWN or UP).
Required preconditions	None
Called functions	None

Example:

```
/* Gets the MC counting state */
CountingStatus MC_CounterStatus = DOWN;
MC_CounterStatus = MC_GetCountingStatus();
```

17.2.29 MC_GetFlagStatus

Function Name	MC_GetFlagStatus
Function Prototype	`FlagStatus MC_GetFlagStatus(u16 MC_FLAG)`
Behavior Description	Checks whether the MC flag is set or not.
Input Parameter	MC_FLAG: specifies the flag to check. Refer to section "*MC_FLAG on page 242*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of MC_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

MC_FLAG

The MC flags that can be read are listed in the following table:

MC_FLAG	Meaning
MC_FLAG_CMPW	Compare W pending bit
MC_FLAG_CMPV	Compare V pending bit
MC_FLAG_CMPU	Compare U pending bit
MC_FLAG_ZPC	Zero of PWM counter pending bit
MC_FLAG_ADT	Automatic data transfer pending bit
MC_FLAG_OTC	Overflow of tacho counter pending bit
MC_FLAG_CPT	Capture of tacho counter pending bit
MC_FLAG_CM0	Compare 0 of PWM pending bit
MC_FLAG_EST	Emergency stop pending bit

Example:

```
/* Check if the MC Overflow of Tacho counter (OTC)flag is set or reset */
if (MC_GetFlagStatus(MC_FLAG_OTC) == SET)
  {
  }
```

17.2.30 MC_ClearFlag

Function Name	MC_ClearFlag
Function Prototype	`void MC_ClearFlag(u16 MC_FLAG)`
Behavior Description	Clears the MC pending flags.
Input Parameter	MC_FLAG: specifies the flags to clear. Refer to section "*MC_FLAG on page 242*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the MC Output Compare W flag */
  MC_ClearFlag(MC_IT_CMPW);
```

17.2.31 MC_GetITStatus

Function Name	MC_GetITStatus
Function Prototype	`ITStatus MC_GetITStatus(u16 MC_IT)`
Behavior Description	Checks whether the specific MC interrupt has occurred or not.
Input Parameter	MC_IT: specifies the interrupt to check. Refer to section "*MC_IT on page 231*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of MC_IT (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Gets the MC output compare W interrupt state */

ITStatus MC_ITStatus = RESET;
MC_ITStatus = MC_GetITStatus(MC_IT_CMPW);
```

17.2.32 MC_ClearITPendingBit

Function Name	MC_ClearITPending Bit
Function Prototype	`void MC_ClearITPendingBit(u16 MC_IT)`
Behavior Description	Clears the MC's interrupt pending bits.
Input Parameter	MC_IT: specifies the pending bit to clear. Refer to section "*MC_IT on page 231*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the MC Output Compare W interrupt pending bit */
  MC_ClearITPendingBit (MC_IT_CMPW);
```

18 Controller area network (CAN)

This peripheral performs communication according to the CAN protocol version 2.0 part A and B. The bitrate can be programmed to values up to 1Mbit/s. Another main feature of this cell is that 32 message objects are implemented which can be fully configured with 2 interfaces.

The first section describes the data structures used in the CAN software library. The second one presents the software library functions.

Note: Before using any CAN function, the I/O ports linked to the CAN RX and CAN TX pins must be set up as follows: GPIO 1.5 (CAN RX pin) must be input Tri-state CMOS, and GPIO 1.6 (CAN TX pin) must be output alternate push-pull.

18.1 CAN Register structure

The structures of the *CAN_TypeDef* and *CAN_MsgObj_TypeDef* registers are defined in the ***91x_map.h*** file as follows:

```
typedef volatile struct
{
  vu16 CRR;
  u16  EMPTY1;
  vu16 CMR;
  u16  EMPTY2;
  vu16 M1R;
  u16  EMPTY3;
  vu16 M2R;
  u16  EMPTY4;
  vu16 A1R;
  u16  EMPTY5;
  vu16 A2R;
  u16  EMPTY6;
  vu16 MCR;
  u16  EMPTY7;
  vu16 DA1R;
  u16  EMPTY8;
  vu16 DA2R;
  u16  EMPTY9;
  vu16 DB1R;
  u16  EMPTY10;
  vu16 DB2R;
  u16  EMPTY11[27];
} CAN_MsgObj_TypeDef;

typedef volatile struct
{
  vu16 CR;
  u16  EMPTY1;
  vu16 SR;
  u16  EMPTY2;
  vu16 ERR;
  u16  EMPTY3;
  vu16 BTR;
  u16  EMPTY4;
  vu16 IDR;
  u16  EMPTY5;
  vu16 TESTR;
  u16  EMPTY6;
```

```
  vu16 BRPR;
  u16  EMPTY7[3];
  CAN_MsgObj_TypeDef sMsgObj[2];
  u16  EMPTY8[16];
  vu16 TXR1R;
  u16  EMPTY9;
  vu16 TXR2R;
  u16  EMPTY10[13];
  vu16 ND1R;
  u16  EMPTY11;
  vu16 ND2R;
  u16  EMPTY12[13];
  vu16 IP1R;
  u16  EMPTY13;
  vu16 IP2R;
  u16  EMPTY14[13];
  vu16 MV1R;
  u16  EMPTY15;
  vu16 MV2R;
  u16  EMPTY16;
} CAN_TypeDef;
```

The following table presents the CAN registers:

Register	Description
CR	CAN Control Register
SR	CAN Status Register
ERR	CAN Error counter Register
BTR	CAN Bit Timing Register
IDR	CAN Interrupt Identifier Register
TESTR	CAN Test Register
BRPR	CAN BRP Extension Register
CRR	CAN IF1 Command Request Register
CMR	CAN IF1 Command Mask Register
M1R	CAN IF1 Message Mask 1 Register
M2R	CAN IF1 Message Mask 2 Register
A1R	CAN IF1 Message Arbitration 1 Register
A2R	CAN IF1 Message Arbitration 2 Register
MCR	CAN IF1 Message Control Register
DA1R	CAN IF1 DATA A 1 Register
DA2R	CAN IF1 DATA A 2 Register
DB1R	CAN IF1 DATA B 1 Register
DB2R	CAN IF1 DATA B 2 Register
CRR	CAN IF2 Command request Register
CMR	CAN IF2 Command Mask Register
M1R	CAN IF2 Message Mask 1 Register
M2R	CAN IF2 Message Mask 2 Register

Register	Description
A1R	CAN IF2 Message Arbitration 1 Register
A2R	CAN IF2 Message Arbitration 2 Register
MCR	CAN IF2 Message Control Register
DA1R	CAN IF2 DATA A 1 Register
DA2R	CAN IF2 DATA A 2 Register
DB1R	CAN IF2 DATA B 1 Register
DB2R	CAN IF2 DATA B 2 Register
TXR1R	CAN Transmission Request 1 Register
TXR2R	CAN Transmission Request 2 Register
ND1R	CAN New Data 1 Register
ND2R	CAN New Data 2 Register
IP1R	CAN Interrupt Pending 1 Register
IP2R	CAN Interrupt Pending 2 Register
MV1R	CAN Message Valid 1 Register
MV2R	CAN Message Valid 2 Register

The CAN is declared in the file below

```
#ifndef EXT
 #Define EXT extern
#endif
...
#define AHB_APB_BRDG1_U    (0x5C000000) /* AHB/APB Bridge 1 UnBuffered Space */
#define AHB_APB_BRDG1_B    (0x4C000000) /* AHB/APB Bridge 1 Buffered Space   */
...
#define APB_CAN_OFST       (0x00009000) /* Offset of CAN */
...
#ifndef Buffered
#define AHBAPB1_BASE           (AHB_APB_BRDG1_U)
...
#else /* Buffered */
...
#define AHBAPB1_BASE           (AHB_APB_BRDG1_B)

/* CAN Base Address definition*/
#define CAN_BASE          (AHBAPB1_BASE + APB_CAN_OFST)


...
/* CAN peripheral declaration*/

#ifndef DEBUG
...
#define CAN         ((CAN_TypeDef *) CAN_BASE)
...
#else
...
EXT  CAN_TypeDef           *CAN;
...

#endif
```

When debug mode is used, CAN pointer is initialized in ***91x_lib.c*** file:

```
#ifdef _CAN
  CAN = (CAN_TypeDef *)CAN_BASE
#endif /* _CAN */
```

_CAN must be defined, in ***91x_conf.h*** file, to access the peripheral registers as follows:

```
#define _CAN
...
```

18.2 Software library functions

The following table lists the various functions of the CAN library.

Function Name	Description
CAN_DeInit	Deinitializes the CAN peripheral registers to their default reset values.
CAN_Init	Initializes the CAN cell and sets the bitrate.
CAN_StructInit	Fills each CAN_InitStruct member with its default value.
CAN_EnterInitMode	Switches the CAN to initialization mode.
CAN_LeaveInitMode	Leaves initialization mode (switches to normal mode).
CAN_EnterTestMode	Switches the CAN to test mode.
CAN_LeaveTestMode	Leaves the current test mode (switches to normal mode).
CAN_SetBitrate	Sets up a standard CAN bitrate.
CAN_SetTiming	Sets up the CAN timing with specific parameters.
CAN_SetUnusedMsgObj	Configures the message object as unused.
CAN_SetTxMsgObj	Configures the message object as TX.
CAN_SetRxMsgObj	Configures the message object as RX.
CAN_InvalidateAllMsgObj	Configures all the message objects as unused.
CAN_ReleaseMessage	Releases the message object.
CAN_ReleaseTxMessage	Releases the transmit message object.
CAN_ReleaseRxMessage	Releases the receive message object.
CAN_SendMessage	Starts transmission of a message.
CAN_ReceiveMessage	Gets the message, if received.
CAN_WaitEndOfTx	Waits until current transmission is finished.
CAN_BasicSendMessage	Starts transmission of a message in BASIC mode.
CAN_BasicReceiveMessage	Gets the message in BASIC mode, if received.
CAN_IsMessageWaiting	Tests the waiting status of a received message.
CAN_IsTransmitRequested	Tests the request status of a transmitted message.

Function Name	Description
CAN_IsInterruptPending	Tests the interrupt status of a message object.
CAN_IsObjectValid	Tests the validity of a message object (ready to use).

18.2.1 CAN_DeInit

Function Name	CAN_DeInit
Function Prototype	`void CAN_DeInit(void)`
Behavior Description	Deinitializes the CAN peripheral registers to their default reset values.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	SCU_APBPeriphReset()

Example:

This example illustrates how to initialize the CAN registers.

```
{
/* Initialize CAN registers*/
    CAN_DeInit ();
}
```

18.2.2 CAN_Init

Function Name	CAN_Init
Function Prototype	`void CAN_Init(CAN_InitTypeDef* CAN_InitStruct)`
Behavior Description	Initializes the CAN peripheral according to the specified parameters in the CAN_InitStruct.
Input Parameter	CAN_InitStruct: pointer to a CAN_InitTypeDef structure that contains the configuration information for the CAN peripheral. Refer to section "*CAN_InitTypeDef on page 249*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called Functions	CAN_EnterInitMode() CAN_SetBitrate() CAN_LeaveInitMode() CAN_LeaveTestMode()

CAN_InitTypeDef

The CAN_InitTypeDef structure is defined in the ***91x_can.h*** file:

```
typedef struct
{
```

```
   u8  CAN_ConfigParameters;
   u32 CAN_Bitrate;
}CAN_InitTypeDef;
```

CAN_ConfigParameters

Specifies the CAN configuration parameters. This member can be any combination of the following values:

CAN_ConfigParameters	Meaning
CAN_CR_TEST	Test mode enable
CAN_CR_CCE	Configuration change enable
CAN_CR_DAR	Disable automatic retransmission
CAN_CR_EIE	Error interrupt enable
CAN_CR_SIE	Status change interrupt enable
CAN_CR_IE	Module interrupt enable
CAN_CR_INIT	Initialization

CAN_Bitrate

Specifies the CAN bit rate. This member can be one of the following values:

CAN_Bitrate	Meaning
CAN_BITRATE_100K	100 kbit/s bit rate
CAN_BITRATE_125K	125 kbit/s bit rate
CAN_BITRATE_250K	250 kbit/s bit rate
CAN_BITRATE_500K	500 kbit/s bit rate
CAN_BITRATE_1M	1 Mbit/s bit rate

Example:

This example illustrates how to initialize the CAN at 100 kbit/s and enable the interrupts.

```
{
  /* Init Structure declarations for CAN*/
  CAN_InitTypeDef CAN_InitStructure;

  /*Configure CAN registers*/
  CAN_InitStructure.CAN_ConfigParameters = CAN_CR_IE;
  CAN_InitStructure.CAN_Bitrate = CAN_BITRATE_100K;
  CAN_Init(&CAN_InitStructure);
}
```

18.2.3 CAN_StructInit

Function Name	CAN_StructInit
Function Prototype	`void CAN_StructInit(CAN_InitTypeDef* CAN_InitStruct)`
Behavior Description	Fills each CAN_InitStruct member with its default value.

Function Name	CAN_StructInit
Input Parameter	CAN_InitStruct: pointer to a CAN_InitTypeDef structure which will be initialized.
OutPut Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

Example:

This example illustrates how to initialize CAN init structure.

```
{
  /* Init Structures declarations for CAN */
  CAN_InitTypeDef CAN_InitStructure;

  /*Initialize CAN structure*/
  CAN_StructInit (&CAN_InitStructure);
}
```

18.2.4 CAN_EnterInitMode

Function Name	CAN_EnterInitMode
Prototype	`void CAN_EnterInitMode(u8 InitMask)`
Behavior Description	Switches the CAN into initialization mode. This function must be used in conjunction with CAN_LeaveInitMode().
Input Parameter	InitMask: specifies the CAN configuration in normal mode. Refer to section "*InitMask on page 252*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Value	None
Required preconditions	None
Called Functions	None

Note: This function sets the INIT bit in the Control register, ORed with the mask and resets the Status register.

InitMask

Specifies the CAN configuration which will be set after the initialisation in normal mode.

This member can be any combination of the following values:

InitMask	Meaning
CAN_CR_CCE	Configuration change enable
CAN_CR_DAR	Disable automatic retransmission
CAN_CR_EIE	Error interrupt enable
CAN_CR_SIE	Status change interrupt enable
CAN_CR_IE	Module interrupt enable

Example:

This example illustrates how to initialize the CAN enable interrupts.

```
{
  CAN_EnterInitMode(CAN_CR_IE);
}
```

18.2.5 CAN_LeaveInitMode

Function Name	CAN_LeaveInitMode
Prototype	`void CAN_LeaveInitMode()`
Behavior Description	Leaves initialization mode (switch into normal mode). This function must be used in conjunction with CAN_EnterInitMode().
Input Parameter	None
Output Parameter	None
Return Value	None
Required preconditions	None
Called Functions	None

Note: This function clears the INIT and CCE bits in the Control register.

Example:

This example illustrates how to leave *CAN* init mode.

```
{
  CAN_LeaveInitMode();
}
```

18.2.6 CAN_EnterTestMode

Function Name	CAN_EnterTestMode
Prototype	`void CAN_EnterTestMode(u8 TestMask);`
Behavior Description	Switches the CAN into test mode. This function must be used in conjunction with CAN_LeaveTestMode().
Input Parameter	TestMask: specifies the configuration in test modes. Refer to section "*TestMask on page 253*"for more details on the allowed values of this parameter.
Output Parameter	None
Return Value	None
Required preconditions	None
Called Functions	None

Note: This function sets the TEST bit in the Control register to enable test mode and updates the Test register by ORing its value with the mask.

TestMask

Specifies the CAN configuration in test modes. This member can be any combination of the following values:

TestMask	Meaning
CAN_TESTR_LBACK	Loopback mode enabled
CAN_TESTR_SILENT	Silent mode enabled
CAN_TESTR_BASIC	Basic mode enabled

Example:

This example illustrates how to switch the CAN into Loopback mode, i.e. RX is disconnected from the bus, and TX is internally linked to RX.

```
{
CAN_EnterTestMode(CAN_TESTR_LBACK);
}
```

18.2.7 CAN_LeaveTestMode

Function Name	CAN_LeaveTestMode
Prototype	`void CAN_LeaveTestMode()`
Behavior Description	Leaves the current test mode (switches into normal mode). This function must be used in conjunction with CAN_EnterTestMode().
Input Parameter	None
Output Parameter	None
Return Value	None
Required preconditions	None
Called Functions	None

Note: This function sets the TEST bit in the Control register to enable write access to the Test register, clears the LBACK, SILENT and BASIC bits in the Test register and clears the TEST bit in the Control register to disable write access to the Test register.

Example:

```
{
CAN_LeaveTestMode();
}
```

18.2.8 CAN_SetBitrate

Function Name	CAN_SetBitrate
Prototype	`void CAN_SetBitrate(u32 bitrate);`
Behavior Description	Sets up a standard CAN bitrate.
Input Parameter	BitRate: specifies the bit rate. Refer to section "*BitRate on page 255*"for more details on the allowed values of this parameter.
Output Parameter	None
Return Value	None
Required preconditions	CAN_EnterInitMode() must have been called before. The APB clock must be 8 MHz
Called Functions	None

Note: This function writes the predefined timing value in the Bit Timing register and clears the BRPR register.

BitRate

Specifies the bit rate. This parameter can be one of the following values:

BitRate	Meaning
CAN_BITRATE_100K	100 kbit/s
CAN_BITRATE_125K	125 kbit/s
CAN_BITRATE_250K	250 kbit/s
CAN_BITRATE_500K	500 kbit/s
CAN_BITRATE_1M	1 Mbit/s

Example:

This example illustrates how to enable the configuration change bit, to be able to set the bitrate

```
{
CAN_EnterInitMode(CAN_CR_CCE);
CAN_SetBitrate(CAN_BITRATE_100K);
CAN_LeaveInitMode();
}
```

18.2.9 CAN_SetTiming

Function Name	CAN_SetTiming
Prototype	`void CAN_SetTiming(u32 tseg1, u32 tseg2, u32 sjw, u32 brp);`
Behavior Description	Sets up the CAN timing with specific parameters.
Input Parameter 1	tseg1: Time Segment before the sample point position. It can take values from 1 to 16.
Input Parameter 2	tseg2: Time Segment after the sample point position. It can take values from 1 to 8.
Input Parameter 3	sjw: Synchronization Jump Width. It can take values from 1 to 4.
Input Parameter 4	brp: Baud Rate Prescaler. It can take values from 1 to 1024.
Output Parameter	None
Return Value	None
Required preconditions	CAN_EnterInitMode() must have been called before.
Called Functions	None

Note: This function writes the timing value in the Bit Timing register, from the tseg1, tseg2, sjw parameters and bits 5..0 of brp parameter and writes the BRPR register with bits 9..0 of brp parameter; and all written values are the real values decremented by one unit.

Example:

This example illustrates how to enable the configuration change bit, to be able to set the specific timing parameters: TSEG1=11, TSEG2=4, SJW=4, BRP=5

```
{
CAN_EnterInitMode(CAN_CR_CCE);
CAN_SetTiming(11, 4, 4, 5);
CAN_LeaveInitMode();
}
```

18.2.10 CAN_SetUnusedMsgObj

Function Name	CAN_SetUnusedMsgObj
Prototype	`ErrorStatus CAN_SetUnusedMsgObj(u32 msgobj);`
Behavior Description	Configure the message object as unused.
Input Parameter	msgobj The message object number, from 0 to 31.
Output Parameter	None
Return Value	None
Required preconditions	An ErrorStatus enumeration value: - SUCCESS: Found Interface to treat the message - ERROR: No interface found to treat the message
Called Functions	CAN_GetFreeIF()

Note: This function searches for a free message interface from IF0 and IF1, sets the WR/RD, Mask, Arb, Control, DataA and DataB bits in the Command Mask register, clears the Mask1 and Mask2 registers, clears the Arb1 and Arb2 register, clears the Message Control register, clears the DataA1, DataA2, DataB1, DataB2 registers and writes the value 1+msgobj in the Command Request register.

Example:

This example illustrates how to invalidate the message objects from 16 to 31: these objects will not be used by the hardware

```
{
for (i=16; i<=31; i++) CAN_SetUnusedMsgObj(i);
}
```

18.2.11 CAN_SetTxMsgObj

Function Name	CAN_SetTxMsgObj
Prototype	`ErrorStatus CAN_SetTxMsgObj(u32 msgobj, u32 idType)`
Behavior Description	Configures the message object as TX.
Input Parameter 1	msgobj: The message object number, from 0 to 31.
Input Parameter 2	idType: The identifier type of the frames that will be transmitted using this message object. The value is one of the following: CAN_STD_ID (standard ID, 11-bit) CAN_EXT_ID (extended ID, 29-bit)
Output Parameter	None
Return Value	An ErrorStatus enumeration value: - SUCCESS: Interface to treat the message - ERROR: No interface to treat the message
Required preconditions	None
Called Functions	CAN_GetFreeIF()

Note: 1 This function: Search for a free message interface from IF0 and IF1. Set the WR/RD, Mask, Arb, Control, DataA and DataB bits in the Command Mask register. Clear the Mask1 and Arb1 registers. Set the MDir bit in the Mask2 register, also the MXtd bit if extended ID is used. Set the MsgVal and Dir bits in the Arb2 register, also the Xtd bit if extended ID is used. Set the TxIE and EoB bits in the Message Control register. Clear the DataA1, DataA2, DataB1, DataB2 registers. Write the value 1+msgobj to the Command Request register to copy the registers into the message RAM.

2 When defining which message object number to use for TX or RX, you must take into account the priority levels when processing the objects. The lower number (0) has the highest priority and the higher number (31) has the lowest priority, whatever their type. Also, for optimum performance, it is not recommended to have "holes" in the object list.

Example:

This example illustrates how to define transmit message object 0 with standard identifiers

```
{
CAN_SetTxMsgObj(0, CAN_STD_ID);
}
```

18.2.12 CAN_SetRxMsgObj

Function Name	CAN_SetRxMsgObj
Prototype	`ErrorStatus CAN_SetRxMsgObj(u32 msgobj, u32 idType, u32 idLow, u32 idHigh, bool singleOrFifoLast);`
Behavior Description	Configures the message object as RX.
Input Parameter 1	msgobj: The message object number, from 0 to 31.
Input Parameter 2	idType: The identifier type of the frames that will be transmitted using this message object. The value is one of the following: CAN_STD_ID (standard ID, 11-bit) CAN_EXT_ID (extended ID, 29-bit)
Input Parameter 3	idLow: The low part of the identifier range used for acceptance filtering. It can take values from 0 to 0x7FF for standard ID, and values from 0 to 0x1FFFFFFF for extended ID.
Input Parameter 4	idHigh: The high part of the identifier range used for acceptance filtering. It can take values from 0 to 0x7FF for standard ID, and values from 0 to 0x1FFFFFFF for extended ID. idHigh must be above idLow. For convenience, use one of the following values to set the maximum ID: CAN_LAST_STD_ID or CAN_LAST_EXT_ID
Input Parameter 5	singleOrFifoLast: End-of-buffer indicator, it can take the following values: -TRUE for a single receive object or a FIFO receive object that is the last one in the FIFO -FALSE for a FIFO receive object that is not the last one
Output Parameter	None
Return Value	An ErrorStatus enumeration value: - SUCCESS: Interface to treat the message - ERROR: No interface to treat the message
Required preconditions	None
Called Functions	CAN_GetFreeIF()

Note: 1 This function: Search for a free message interface from IF0 and IF1. Set the WR/RD, Mask, Arb, Control, DataA and DataB bits in the Command Mask register. Write the ID mask value formed from idLow and idHigh in the Mask1 and Mask2 registers, and also set the MXtd bit in the Mask2 register if extended ID is used. Write the ID arbitration value formed from idLow and idHigh in the Arb1 and Arb2 registers, set the MsgVal bit, and also Xtd bit in the Arb2 register if extended ID is used. Set the RxIE and UMask bits in the Message Control register, and also the EoB bit if the parameter singleOrFifoLast is TRUE. Clear the DataA1, DataA2, DataB1, DataB2 registers. Write the value 1+msgobj to the Command Request register to copy the selected registers into the message RAM.

2 Care must be taken when defining an ID range: all combinations of idLow and idHigh will not always produce the expected result, because of the way identifiers are filtered by the hardware. The criteria applied to keep a received frame is as follows: (received ID) AND (ID mask) = (ID arbitration), where AND is a bitwise operator. Consequently, for idLow, it is generally better to choose a value with some LSBs cleared, and for idHigh a value that

"logically contains" idLow and with the same LSBs set. Example: the range 0x100-0x3FF will work, but the range 0x100-0x2FF will not because 0x100 is not logically contained in 0x2FF (i.e. 0x100 & 0x2FF = 0).

Example:

This example illustrates how to define FIFOs and acceptance filtering

```
{
/*Define a receive FIFO of depth 2 (objects 0 and 1) for standard identifiers, in
which IDs are filtered in the range 0x400-0x5FF*/
CAN_SetRxMsgObj(0, CAN_STD_ID, 0x400, 0x5FF, FALSE);
CAN_SetRxMsgObj(1, CAN_STD_ID, 0x400, 0x5FF, TRUE);
/*Define a single receive object for extended identifiers, in which all IDs are
filtered in*/
CAN_SetRxMsgObj(2, CAN_EXT_ID, 0, CAN_LAST_EXT_ID, TRUE);
}
```

18.2.13 CAN_InvalidateAllMsgObj

Function Name	CAN_InvalidateAllMsgObj
Prototype	`void CAN_InvalidateAllMsgObj();`
Behavior Description	Configures all the message objects as unused.
Input Parameter	None
Output Parameter	None
Return Value	None
Required preconditions	None
Called Functions	CAN_SetUnusedMsgObj()

Example:

```
{
CAN_InvalidateAllMsgObj();
}
```

18.2.14 CAN_ReleaseMessage

Function Name	CAN_ReleaseMessage
Prototype	`void CAN_ReleaseMessage(u32 msgobj);`
Behavior Description	Releases the message object.
Input Parameter	msgobj The message object number, from 0 to 31.
Output Parameter	None
Return Value	None
Required preconditions	None
Called Functions	CAN_GetFreeIF()

Note: This function:

Search for a free message interface from IF0 and IF1.

Set the bits ClrIntPnd and TxRqst/NewDat in the Command Mask register.

Write the value 1+msgobj to the Command Request register to copy the selected registers into the message RAM.

Example:

This example illustrates how to release the message object 0.

```
{
CAN_ReleaseMessage(0);
}
```

18.2.15 CAN_ReleaseTxMessage

Function Name	CAN_ReleaseTxMessage
Prototype	`void CAN_ReleaseTxMessage(u32 msgobj);`
Behavior Description	Releases the transmit message object.
Input Parameter	msgobj: The message object number, from 0 to 31.
Output Parameter	None
Return Value	None
Required preconditions	The message interface 0 must not be busy.
Called Functions	None

Note: This function:

Sets the ClrIntPnd and TxRqst/NewDat bits in the Command Mask register of message interface 0.

Writes the value 1+msgobj to the Command Request register to copy the selected registers into the message RAM.

Example:

This example illustrates how to release transmit message object 0.

```
{
/*It is assumed that message interface 0 is always used for transmission*/
/*Release the transmit message object 0*/
CAN_ReleaseTxMessage(0);
}
```

18.2.16 CAN_ReleaseRxMessage

Function Name	CAN_ReleaseRxMessage
Prototype	`void CAN_ReleaseRxMessage(u32 msgobj);`
Behavior Description	Releases the receive message object.
Input Parameter	msgobj: The message object number, from 0 to 31.
Output Parameter	None
Return Value	None
Required preconditions	The message interface 1 must not be busy.
Called Functions	None

Note: This function:

Sets the bits ClrIntPnd and TxRqst/NewDat in the Command Mask register of message interface 1.

Writes the value 1+msgobj to the Command Request register to copy the selected registers into the message RAM.

Example:

This example illustrates how to release the receive message object 0.

```
{
/*It is assumed that message interface 1 is always used for reception*/
/*Release the receive message object 0*/
CAN_ReleaseRxMessage(0);
}
```

18.2.17 CAN_SendMessage

Function Name	CAN_SendMessage
Prototype	`ErrorStatus CAN_SendMessage(u32 msgobj, canmsg* pCanMsg);`
Behavior Description	Starts transmission of a message.
Input Parameter 1	msgobj: The message object number, from 0 to 31.
Input Parameter 2	pCanMsg: Pointer to the canmsg structure that contains the data to transmit: ID type, ID value, data length, data values.
Output Parameter	None
Return Value	An ErrorStatus enumeration value: - SUCCESS: Transmission OK - ERROR: No transmission
Required preconditions	The message object must have been set up properly.
Called Functions	None

Note: This function:

Waits for message interface 0 to be free.

Read the Arbitration and Message Control registers.

Waits for message interface 0 to be free.

Updates the Arbitration, Message Control, DataA and DataB registers with the message contents.

Writes the value 1+msgobj to the Command Request register to copy the selected registers into the message RAM and to start the transmission.

Example:

This example illustrates how to send a standard ID data frame containing 4 data value.

```
{
canmsg CanMsg = { CAN_STD_ID, 0x111, 4, {0x10, 0x20, 0x40, 0x80} };
/*Send a standard ID data frame containing 4 data values*/
CAN_SendMessage(0, &CanMsg);
}
```

18.2.18 CAN_ReceiveMessage

Function Name	CAN_ReceiveMessage
Prototype	`ErrorStatus CAN_ReceiveMessage(u32 msgobj, bool release, canmsg* pCanMsg);`
Behavior Description	Gets the message, if received.
Input Parameter 1	msgobj: The message object number, from 0 to 31.
Input Parameter 2	Release: The message release indicator, it can take the following values: -TRUE: the message object is released at the same time as it is copied from message RAM, then it is free for next reception -FALSE: the message object is not released, it is to the caller to do it
Input Parameter 3	pCanMsg: Pointer to the canmsg structure where the received message is copied.
Output Parameter	None
Return Value	An ErrorStatus enumeration value: - SUCCESS: Transmission OK - ERROR: No transmission
Required preconditions	The message object must have been set up properly.
Called Functions	macro CAN_IsMessageWaiting()

Note: This function:

Test the bit corresponding to the message object number in the NewData registers.

Clear the bit RxOk in the Status register.

Copy the message contents from the message RAM to the registers and to the structure, and release the message object if asked.

Example:

This example illustrates how to receive a message.

```
{
  canmsg CanMsg;
/*Receive a message in the object 0 and ask for release*/
  if (CAN_ReceiveMessage(CAN0, 0, TRUE, &CanMsg))
  {
    /*Check or copy the message contents*/
  }
  else
  {
    /* Error handling*/
  }
}
```

18.2.19 CAN_WaitEndOfTx

Function Name	CAN_WaitEndOfTx
Prototype	`ErrorStatus CAN_WaitEndOfTx(void);`
Behavior	Tests the TxOk bit in the Status register, and loops until it is set. Clears this bit to prepare the next transmission.
Input Parameter	None
Output Parameter	None
Return Value	An ErrorStatus enumeration value: - SUCCESS: Transmission ended - ERROR: Transmission did not occur yet
Required preconditions	A message must have been sent before.
Called Functions	None

Example:

This example illustrates how to send frames.

```
{
/*Send consecutive data frames using message object 0*/
for (i = 0; i < 10; i++)
{
  CAN_SendMessage(0, CanMsgTable[i]);
  CAN_WaitEndOfTx();
}
}
```

18.2.20 CAN_BasicSendMessage

Function Name	CAN_BasicSendMessage
Prototype	`ErrorStatus CAN_BasicSendMessage(canmsg* pCanMsg);`
Behavior Description	Starts transmission of a message in BASIC mode. This mode does not use the message RAM.
Input Parameter	pCanMsg: Pointer to the canmsg structure that contains the data to transmit: ID type, ID value, data length, data values.
Output Parameter	None
Return Value	An ErrorStatus enumeration value: - SUCCESS: Transmission OK - ERROR: No transmission
Required preconditions	The CAN must have been switched into BASIC mode.
Called Functions	None

Note: This function:

Clears the bit NewDat in message interface 1.

Writes the Arbitration, Message Control, DataA and DataB registers of message interface 0, with the message contents.

Writes the value 1+msgobj to the Command Request register to start the transmission.

Example:

This example illustrates how to send frames.

```
{
/*Send consecutive data frames using message object 0*/
for (i = 0; i < 10; i++)
{
  CAN_SendMessage(0, CanMsgTable[i]);
  CAN_WaitEndOfTx();
}
}
```

18.2.21 CAN_BasicReceiveMessage

Function Name	CAN_BasicReceiveMessage
Prototype	`ErrorStatus CAN_BasicReceiveMessage(canmsg* pCanMsg);`
Behavior Description	Gets the message in BASIC mode, if received. This mode does not use the message RAM.
Input Parameter	pCanMsg: Pointer to the canmsg structure where the received message is copied.
Output Parameter	None
Return Value	An ErrorStatus enumeration value: - SUCCESS: Reception OK - ERROR: No message pending
Required preconditions	The CAN must have been switched into BASIC mode.
Called Functions	None

Note: This function:

Tests the bit NewDat in the Message Control register of message interface 1.

Clears the bit RxOk in the Status register.

Copies the message contents from the message interface 1 registers to the structure.

Example:

This example illustrates how to receive frame in basic mode.

```
{
 canmsg CanMsg;
/*Receive a message in BASIC mode*/
if (CAN_BasicReceiveMessage(&CanMsg))
{
  /* Check or copy the message contents*/
}
else
{
  /* Error handling*/
}
}
```

18.2.22 CAN_IsMessageWaiting

Function Name	CAN_IsMessageWaiting
Prototype	`u32 CAN_IsMessageWaiting(u32 msgobj);`
Behavior Description	Tests the waiting status of a received message.
Input Parameter	msgobj: The message object number, from 0 to 31.
Output Parameter	None
Return Value	A non-zero value if the corresponding message object has received a message waiting to be copied, else 0.
Required preconditions	The corresponding message object must have been set as RX.
Called Functions	None

Note: This function:

Tests the corresponding bit in the NewData 1 or 2 registers.

Example:

This example illustrates how to test the new data registers for the message object 0.

```
{
/*Test if a message is pending in the receive message object 0*/
if (CAN_IsMessageWaiting(0))
{
  /* Receive the message from this message object (i.e. get its data from message
RAM)*/
}
}
```

18.2.23 CAN_IsTransmitRequested

Function Name	CAN_IsTransmitRequested
Prototype	`u32 CAN_IsTransmitRequested(u32 msgobj);`
Behavior Description	Tests the request status of a transmitted message.
Input Parameter	msgobj: The message object number, from 0 to 31.
Output Parameter	None
Return Value	A non-zero value if the corresponding message is requested to transmit, else 0.
Required preconditions	A message must have been sent before.
Called Functions	None

Note: This function:

Tests the corresponding bit in the Transmission Request 1 or 2 registers.

Example:

This example illustrates how to test the transmit request.

```
{
/*Send a message using object 0*/
CAN_SendMessage(0, &CanMsg);
/*Wait for the end of transmit request*/
while (CAN_IsTransmitRequested(0));
/*Now, the message is being processed by the priority handler of the CAN cell, and
ready to be emitted on the bus*/
}
```

18.2.24 CAN_IsInterruptPending

Function Name	CAN_IsInterruptPending
Prototype	`u32 CAN_IsInterruptPending(u32 msgobj);`
Behavior Description	Tests the interrupt status of a message object.
Input Parameter	msgobj: The message object number, from 0 to 31.
Output Parameter	None
Return Value	A non-zero value if the corresponding message has an interrupt pending, else 0.
Required preconditions	The interrupts must have been enabled.
Called Functions	None

This function Tests the corresponding bit in the Interrupt Pending 1 or 2 registers.

Example:

This example illustrates how to test interrupt pending.

```
{
/*Send a message using object 0*/
CAN_SendMessage(0, &CanMsg)
/* Wait for the TX interrupt*/
while (!CAN_IsInterruptPending(0));
}
```

18.2.25 CAN_IsObjectValid

Function Name	CAN_IsObjectValid
Prototype	`u32 CAN_IsObjectValid(u32 msgobj);`
Behavior Description	Tests the validity of a message object (ready to use). A valid object means that it has been set up either as TX or as RX, and so is used by the hardware.
Input Parameter	msgobj: The message object number, from 0 to 31.
Output Parameter	None
Return Value	A non-zero value if the corresponding message object is valid, else 0.
Required preconditions	None
Called Functions	None

Note: This function tests the corresponding bit in the Message Valid 1 or 2 registers.

Example:

This example illustrates how to test the validity of message object 10.

```
{
if (CAN_IsObjectValid(10))
{
  /* Do something with message object 10*/
}
}
```

19 Analog-to-Digital Converter (ADC)

The 10-bit ADC is a simple successive approximation based ADC that has 8 analog inputs. The analog inputs are just a simple analog switch (mux). Only one channel can be active at a time.

19.1 ADC Register structure

The ADC register structure *ADC_TypeDef* is defined in the ***91x_map.h*** file as follows:

```
typedef struct
{
  vu16 CR;
  vu16 EMPTY1;
  vu16 CCR;
  vu16 EMPTY2;
  vu16 HTR;
  vu16 EMPTY3;
  vu16 LTR;
  vu16 EMPTY4;
  vu16 CRR;
  vu16 EMPTY5;
  vu16 DR0;
  vu16 EMPTY6;
  vu16 DR1;
  vu16 EMPTY7;
  vu16 DR2;
  vu16 EMPTY8;
  vu16 DR3;
  vu16 EMPTY9;
  vu16 DR4;
  vu16 EMPTY10;
  vu16 DR5;
  vu16 EMPTY11;
  vu16 DR6;
  vu16 EMPTY12;
  vu16 DR7;
  vu16 EMPTY13;
  vu16 PRS;
  vu16 EMPTY14;
} ADC_TypeDef;
```

The following table presents the ADC registers:

Register	Description
CR	Control Register
CCR	Channel Configuration Register
HTR	Higher Threshold Register
LTR	Lower Threshold Register
CRR	Compare Result Register
DR0	Data Register for Channel 0
DR1	Data Register for Channel 1
DR2	Data Register for Channel 2
DR3	Data Register for Channel 3
DR4	Data Register for Channel 4
DR5	Data Register for Channel 5
DR6	Data Register for Channel 6
DR7	Data Register for Channel 7
PRS	Prescaler Value Register

The ADC is declared in the file below

```
#ifndef EXT
 #Define EXT extern
#endif
...
#define AHB_APB_BRDG1_U    (0x5C000000) /* AHB/APB Bridge 1 UnBuffered Space */
#define AHB_APB_BRDG1_B    (0x4C000000) /* AHB/APB Bridge 1 Buffered Space   */
...
#define APB_ADC_OFST       (0x0000A000) /* Offset of ADC */
...
#ifndef Buffered
#define AHBAPB1_BASE           (AHB_APB_BRDG1_U)
...
#else /* Buffered */
...
#define AHBAPB1_BASE           (AHB_APB_BRDG1_B)

/* ADC Base Address definition*/
#define ADC_BASE          (AHBAPB1_BASE + APB_ADC_OFST)




...
/* ADC peripheral declaration*/

#ifndef DEBUG
...
#define ADC         ((ADC_TypeDef *) ADC_BASE)
...
#else
...
#ifdef _ADC
```

```
EXT ADC_TypeDef            *ADC;
#endif /* _ADC */
...

#endif
```

When debug mode is used, ADC pointer is initialized in ***91x_lib.c*** file:

```
#ifdef _ADC
  ADC = (ADC_TypeDef *)ADC_BASE
#endif /* _ADC */
```

_ADC must be defined, in the ***91x_conf.h*** file, to access the peripheral registers as follows:

```
#define _ADC
...
```

19.2 Software library functions

The following table enumerates the different functions of the ADC library.

Function Name	Description
ADC_DeInit	Reset all the ADC registers to their default values.
ADC_StructInit	Reset all the Init struct parameters to their default values.
ADC_Init	Configure the ADC according to the input passed parameters.
ADC_PrescalerConfig	Configure the ADC prescaler value.
ADC_GetPrescalerValue	Get the ADC prescaler value.
ADC_GetFlagStatus	Check whether the specified ADC flag is set or not.
ADC_ClearFlag	Clear the ADC pending flags.
ADC_GetConversionValue	Read the result of conversion from the appropriate data register.
ADC_GetAnalogWatchdogResult	Return the result of the comparison on the selected Analog Watchdog.
ADC_ClearAnalogWatchdogResult	Clear result of the comparison on the selected Analog Watchdog.
ADC_GetWatchdogThreshold	Get the Higher or the Lower thresholds values of the watchdog.
ADC_ITConfig	Enable /Disable the specified ADC interrupts.
ADC_StandbyModeConfig	Enable/disable the standby mode.
ADC_Cmd	Power on or put in the Reset mode the ADC peripheral.
ADC_ConversionCmd	Start or stop the ADC conversion in the selected mode.

19.2.1 ADC_DeInit

Function Name	ADC_DeInit
Function Prototype	`void ADC_DeInit(void)`
Behavior Description	Deinitializes the ADC peripheral registers to their default reset values.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	`SCU_APBPeriphReset()`

Example:

```
/* To initialize the ADC registers to their default values */
ADC_DeInit();
```

19.2.2 ADC_StructInit

Function Name	ADC_StructInit
Function Prototype	`void ADC_StructInit(ADC_InitTypeDef* ADC_InitStruct)`
Behavior Description	Fills each ADC_InitStruct member with its reset value.
Input Parameter	ADC_InitStruct: pointer to a ADC_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

Example:

```
/* Initialize the ADC structure */
ADC_StructInit(&ADC_InitStruct);
```

19.2.3 ADC_Init

Function Name	ADC_Init
Function Prototype	`void ADC_Init(ADC_InitTypeDef* ADC_InitStruct)`
Behavior Description	Initializes the ADC peripheral according to the specified parameters in the ADC_InitStruct.
Input Parameter	ADC_InitStruct: pointer to a ADC_InitTypeDef structure that contains the configuration information for the specified ADC peripheral. Refer to section "*ADC_InitTypeDef on page 275*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None

Required preconditions	...
Called functions	None

ADC_InitTypeDef

The *ADC_InitTypeDef* structure is defined in the ***91x_ADC.h*** file:

```
typedef struct
{
  u16 ADC_WDG_High_Threshold;
  u16 ADC_WDG_Low_Threshold;
  u16 ADC_Channel_0_Mode;
  u16 ADC_Channel_1_Mode;
  u16 ADC_Channel_2_Mode;
  u16 ADC_Channel_3_Mode;
  u16 ADC_Channel_4_Mode;
  u16 ADC_Channel_5_Mode;
  u16 ADC_Channel_6_Mode;
  u16 ADC_Channel_7_Mode;
  u16 ADC_Select_Channel;
  FunctionalState ADC_Scan_Mode;
  u16 ADC_Conversion_Mode;
}ADC_InitTypeDef;
```

ADC_WDG_High_Threshold

The high threshold value of the watchdog. It can be a value between 0 and 0x3FF.

ADC_WDG_Low_Threshold

The low threshold value of the watchdog. It can be a value between 0 and ADC_WDG_High_Threshold.

ADC_Channel_i_Mode

The channel i conversion mode. It can take one of the following values.

ADC_Channel_i_Mode	**Meaning**
ADC_NoThreshold_Conversion	Channel i is converted without watchdog
ADC_HighThreshold_Conversion	Channel i is converted with watchdog on the higher threshold
ADC_LowThreshold_Conversion	Channel i is converted with watchdog on the lower threshold
ADC_NoConversion	Channel i is never converted

ADC_Select_Channel

The channel to be converted when scan mode is disabled. It can be one of the following values.

ADC_Select_Channel	Meaning
ADC_Channel_0	ADC Channel 0
ADC_Channel_1	ADC Channel 1
ADC_Channel_2	ADC Channel 2
ADC_Channel_3	ADC Channel 3
ADC_Channel_4	ADC Channel 4
ADC_Channel_5	ADC Channel 5
ADC_Channel_6	ADC Channel 6
ADC_Channel_7	ADC Channel 7

ADC_Scan_Mode

The scan mode status. It can be one of the following values.

ADC_Scan_Mode	Meaning
ENABLE	Scan mode is enabled, all the ADC inputs are converted.
DISABLE	Scan mode is disabled, only the selected channel is converted.

ADC_Conversion_Mode

The type of the conversion. It can be one of the following values.

ADC_Conversion_Mode	Meaning
ADC_Continuous_Mode	The conversion is restarted continuously
ADC_Single_Mode	One single conversion

Example:

```
/* Configure the ADC to convert the channel 2 on the high threshold (0xFF) and in
continuous mode */
ADC_InitStruct      ADC_Init_Structure;
ADC_StructInit(&ADC_Init_Structure);
ADC_Init_Structure->ADC_WDG_High_Threshold = 0xFF;
ADC_Init_Structure->ADC_Channel_2_Mode = ADC_HighThreshold_Conversion;
ADC_Init_Structure->ADC_Conversion_Mode = ADC_Continuous_Mode;
ADC_Init(&ADC_Init_Structure);
```

19.2.4 ADC_PrescalerConfig

Function Name	ADC_PrescalerConfig
Function Prototype	`void ADC_PrescalerConfig(u8 ADC_Prescaler);`
Behavior Description	Configures the ADC prescaler value.
Input Parameter	ADC_Prescaler: specifies the prescaler value. This parameter can be a value from 0x0 to 0xFF.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

Example:

```
/* Configure the ADC prescaler to 0xF */
ADC_PrescalerConfig(0xF);
```

19.2.5 ADC_GetPrescalerValue

Function Name	ADC_GetPrescalerValue
Function Prototype	`u8 ADC_GetPrescalerValue(void);`
Behavior Description	Gets the ADC prescaler value.
Input Parameter	None
Output Parameter	None
Return Parameter	The prescaler value.
Required preconditions	...
Called functions	None

Example:

```
u8 ADC_Prescaler;
/* Get the ADC prescaler value */
ADC_Prescaler = ADC_GetPrescalerValue();
```

19.2.6 ADC_GetFlagStatus

Function Name	ADC_GetFlagStatus
Function Prototype	`FlagStatus ADC_GetFlagStatus(u16 ADC_Flag)`
Behavior Description	Checks whether the specified ADC flag is set or not.
Input Parameter	ADC_Flag: specifies the flag to check. Refer to section "*ADC_Flag on page 278*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of ADC_FLAG (SET or RESET).
Required preconditions	...
Called functions	None

ADC_Flag

The ADC flags that can be read are listed in the following table:

ADC_Flag	Meaning
ADC_FLAG_OV_CH_0	Conversion overflow status for channel 0
ADC_FLAG_OV_CH_1	Conversion overflow status for channel 1
ADC_FLAG_OV_CH_2	Conversion overflow status for channel 2
ADC_FLAG_OV_CH_3	Conversion overflow status for channel 3
ADC_FLAG_OV_CH_4	Conversion overflow status for channel 4
ADC_FLAG_OV_CH_5	Conversion overflow status for channel 5
ADC_FLAG_OV_CH_6	Conversion overflow status for channel 6
ADC_FLAG_OV_CH_7	Conversion overflow status for channel 7
ADC_FLAG_ECV	End of conversion status
ADC_FLAG_AWD	Analog watchdog status

Example:

```
/* To get the end of conversion flag */
FlagStatus ADC_ECV_Status;
ADC_ECV_Status = ADC_GetFlagStatus(ADC_FLAG_ECV);
```

19.2.7 ADC_ClearFlag

Function Name	ADC_ClearFlag
Function Prototype	`void ADC_ClearFlag(16 ADC_Flag)`
Behavior Description	Clears the ADC pending flags.
Input Parameter	ADC_Flag: specifies the flag to clear. Refer to section “ADC_FLAG” for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

ADC_Flag

To clear ADC flags, use a combination of one or more of the following values:

ADC_Flag	Meaning
ADC_FLAG_ECV	End of conversion status
ADC_FLAG_AWD	Analog watchdog status

Example:

```
/* To clear the end of conversion flag */
ADC_ClearFlag(ADC_FLAG_ECV);
```

19.2.8 ADC_GetConversionValue

Function Name	ADC_GetConversionValue
Function Prototype	`u16 ADC_GetConversionValue(u16 ADC_Channel)`
Behavior Description	Reads the result of conversion from the appropriate data register.
Input Parameter	ADC_Channel: the corresponding channel of the ADC peripheral. Refer to section "*ADC_Channel on page 280*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	Returns the result of the conversion for the specific channel.
Required preconditions	...
Called functions	None

ADC_Channel

The ADC channels that be selected are listed in the following table:

ADC_Channel	**Meaning**
ADC_Channel_0	ADC Channel 0
ADC_Channel_1	ADC Channel 1
ADC_Channel_2	ADC Channel 2
ADC_Channel_3	ADC Channel 3
ADC_Channel_4	ADC Channel 4
ADC_Channel_5	ADC Channel 5
ADC_Channel_6	ADC Channel 6
ADC_Channel_7	ADC Channel 7

Example:

```
/* To get the conversion value of channel 1 */
u16 ADC_Conversion_Value;
ADC_Conversion_Value = ADC_GetConversionValue(ADC_Channel_1);
```

19.2.9 ADC_GetAnalogWatchdogResult

Function Name	ADC_GetAnalogWatchdogReslt
Function Prototype	`FlagStatus ADC_GetAnalogWatchdogResult(u16 ADC_Channel)`
Behavior Description	Returns the result of the comparison on the selected Analog Watchdog
Input Parameter	ADC_Channel: the corresponding channel of the ADC peripheral. Refer to section "*ADC_Channel on page 282*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The state of the comparison (SET or RESET)
Required preconditions	...
Called functions	None

ADC_Channel

The ADC channels that can be selected are listed in the following table:

ADC_Channel	**Meaning**
ADC_Channel_0	ADC Channel 0
ADC_Channel_1	ADC Channel 1
ADC_Channel_2	ADC Channel 2
ADC_Channel_3	ADC Channel 3
ADC_Channel_4	ADC Channel 4
ADC_Channel_5	ADC Channel 5
ADC_Channel_6	ADC Channel 6
ADC_Channel_7	ADC Channel 7

Example:

```
/* To get the watchdog result of channel 1 */
FlagStatus ADC_Analog_WDG_Status;
ADC_Analog_WDG_Status = ADC_GetAnalogWatchdogResult(ADC_Channel_1);
```

19.2.10 ADC_ClearAnalogWatchdogResult

Function Name	ADC_ClearAnalogWatchdogResult
Function Prototype	`void ADC_ClearAnalogWatchdogResult(u16 ADC_Channel)`
Behavior Description	Clear the result of the comparison on the selected Analog Watchdog
Input Parameter	ADC_Channel: the corresponding channel of the ADC peripheral. Refer to section "*ADC_Channel on page 282*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

ADC_Channel

The ADC channels that can be selected are listed in the following table:

ADC_Channel	**Meaning**
ADC_Channel_0	ADC Channel 0
ADC_Channel_1	ADC Channel 1
ADC_Channel_2	ADC Channel 2
ADC_Channel_3	ADC Channel 3
ADC_Channel_4	ADC Channel 4
ADC_Channel_5	ADC Channel 5
ADC_Channel_6	ADC Channel 6
ADC_Channel_7	ADC Channel 7

Example:

```
/* To clear the watchdog result of the channel 2 */
ADC_ClearAnalogWatchdogResult(ADC_Channel_2);
```

19.2.11 ADC_GetWatchdogThreshold

Function Name	ADC_GetWatchdogThreshold
Function Prototype	`u16 ADC_GetWatchdogThreshold(ADC_ThresholdType ADC_Threshold)`
Behavior Description	Gets the higher or the lower threshold values of the watchdog.
Input Parameter	ADC_Threshold: The lower or the higher threshold. Refer to section “ADC_ThresholdType” for more details on the allowed values of this parameter
Output Parameter	None
Return Parameter	The selected threshold.
Required preconditions	...
Called functions	None

ADC_ThresholdType

The ADC threshold that can be read are listed in the following table:

ADC_ThresholdType	**Meaning**
ADC_HighThreshold	The high threshold of the watchdog
ADC_LowThreshold	The low threshold of the watchdog

Example:

```
/* To get the high threshold of the watchdog*/
u16 ADC_High_Threshold;
ADC_High_Threshold = ADC_GetWatchdogThreshold(ADC_HighTreshold);
```

19.2.12 ADC_ITConfig

Function Name	ADC_ITConfig
Function Prototype	`void ADC_ITConfig(u16 ADC_IT, FunctionalState ADC_NewState)`
Behavior Description	Enables or disables the specified ADC interrupts.
Input Parameter1	ADC_IT: specifies the ADC interrupt sources to be enabled or disabled. Refer to section “ADC_IT” for more details on the allowed values of this parameter.
Input Parameter2	ADC_NewState: new state of the specified ADC interrupts. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

ADC_IT

To enable or disable ADC interrupts, use a combination of one or more of the following values:

ADC_IT	Meaning
ADC_IT_ECV	End of conversion interrupt
ADC_IT_AWD	Analog watchdog interrupt

Example:

```
/* To enable the end of conversion interrupt */
ADC_ITConfig(ADC_IT_ECV,ENABLE);
```

19.2.13 ADC_StandbyModeCmd

Function Name	ADC_StandbyModeCmd
Function Prototype	`void ADC_StandbyModeCmd(FunctionalState ADC_NewState)`
Behavior Description	Enables or disables ADC standby mode.
Input Parameter	ADC_NewState: new state of the ADC standby mode. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

Example:

```
/* To enable the standby mode */
ADC_StandbyModeCmd(ENABLE);
```

19.2.14 ADC_Cmd

Function Name	ADC_Cmd
Function Prototype	`void ADC_Cmd(FunctionalState ADC_NewState)`
Behavior Description	Powers on or puts the ADC peripheral in reset mode.
Input Parameter	ADC_NewState: new state of the ADC peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

Example:

```
/* To power on the ADC */
ADC_Cmd(ENABLE);
```

19.2.15 ADC_ConversionCmd

Function Name	ADC_ConversionCmd
Function Prototype	`void ADC_ConversionCmd(u16 ADC_Conversion)`
Behavior Description	Start or stop the ADC conversion in the selected mode.
Input Parameter	ADC_Conversion: the conversion command of the ADC peripheral. Refer to section "*ADC_Conversion on page 285*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	...
Called functions	None

ADC_Conversion

The ADC conversion that can check for are listed in the following table:

ADC_Conversion	**Meaning**
ADC_Conversion_Start	Start the conversion
ADC_Conversion_Stop	Stop the conversion

Example:

```
/* To start the conversion */
ADC_ConversionCmd(ADC_Conversion_Start);
```

20 AHB/APB Bridges (AHBAPB)

The AHB/APB bridge provides a completely asynchronous connection between the AHB with the APB buses. The AHB/APB clock ratio could be typically around 1/10. As a consequence, an APB read access will need more than 20 AHB cycle to be done.

The first section below describes the data structures used in the AHBAPB software library. The second one presents the software library functions.

20.1 AHBAPB register structure

The AHBAPB register structure *AHBAPB_TypeDef* is defined in the ***91x_map.h*** file as follows:

```
typedef struct
{
vu32 BSR;        /* Bridge Status Register            */
vu32 BCR;        /* Bridge Configuration Register     */
vu32 PAER;       /* Peripheral Address Error register */
} AHBAPB_TypeDef;
```

The following table presents the AHBAPB registers:

Register	Description
AHBAPB_BSR	Bridge Status Register
AHBAPB_BCR	Bridge Configuration Register
AHBAPB_PAER	Peripheral Address Error Register

The 2 AHBAPB interfaces are declared in the same file:

```
...
#define AHB_APB_BRDG0_B    (0x48000000)
#define AHB_APB_BRDG1_B    (0x4C000000)
...
#define AHB_APB_BRDG0_U    (0x58000000)
#define AHB_APB_BRDG1_U    (0x5C000000)
...
#ifndef Buffered
...
#define AHBAPB0_BASE           (AHB_APB_BRDG0_U)
#define AHBAPB1_BASE           (AHB_APB_BRDG1_U)
...
#else /* Buffered */
...
#define AHBAPB0_BASE           (AHB_APB_BRDG0_B)
#define AHBAPB1_BASE           (AHB_APB_BRDG1_B)
...
#endif
...
#ifndef DEBUG
...
#define AHBAPB0               ((AHBAPB_TypeDef *)AHBAPB0_BASE)
#define AHBAPB1               ((AHBAPB_TypeDef *)AHBAPB1_BASE)
...
#else
...
#ifdef _AHBAPB0
```

```
EXT AHBAPB_TypeDef          *AHBAPB0;
#endif /* _AHBAPB0 */

#ifdef _AHBAPB1
EXT AHBAPB_TypeDef          *AHBAPB1;
#endif /*_AHBAPB1 */
```

When debug mode is used, the AHBAPB pointer is initialized in the ***91x_lib.c*** file:

```
#ifdef _AHBAPB0
AHBAPB0 = (AHBAPB_TypeDef *)AHBAPB0_BASE;
#endif /*_AHBAPB0 */
#ifdef _AHBAPB1
AHBAPB1 = (AHBAPB_TypeDef *)AHBAPB1_BASE;
#endif /*_AHBAPB1 */
```

_AHBAPB, _AHBAPB0, _AHBAPB1 must be defined, in the ***91x_conf.h*** file, to access the peripheral registers as follows:

```
#define _AHBAPB
#define _AHBAPB0
#define _AHBAPB1
```

20.2 Software library functions

The following table enumerates the different functions of the AHBAPB library.

Function Name	Description
AHBAPB_DeInit	Deinitializes the AHBAPBx peripheral registers to their default reset values.
AHBAPB_Init	Initializes the AHBAPBx peripheral according to the specified parameters in the AHBAPB_InitStruct.
AHBAPB_StructInit	Fills each AHBAPB_InitStruct member with its reset value.
AHBAPB_GetFlagStatus	Checks whether the specified AHBAPB flag is set or not.
AHBAPB_ClearFlag	Clears the AHBAPBx's pending flags.
AHBAPB_GetPeriphAddrError	Gets the AHBAPB error address peripherals.

20.2.1 AHBAPB_DeInit

Function Name	AHBAPB_DeInit
Function Prototype	`void AHBAPB_DeInit(AHBAPB_TypeDef* AHBAPBx)`
Behavior Description	Deinitializes the AHBAPBx peripheral registers to their default reset values.
Input Parameter	AHBAPBx: where x can be 0 or1 to select the AHBAPB peripheral.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/*Deinitializes the AHPABP0 peripheral registers to their default reset values*/
AHBAPB_DeInit(AHBAPB0);
```

20.2.2 AHBAPB_Init

Function Name	AHBAPB_Init
Function Prototype	`void AHBAPB_Init(AHBAPB_TypeDef* AHBAPBx, AHBAPB_InitTypeDef* AHBAPB_InitStruct)`
Behavior Description	Initializes the AHBAPBx peripheral according to the specified parameters in the AHBAPB_InitStruct.
Input Parameter1	AHBAPBx: where x can be 0 or1 to select the AHBAPB peripheral.
Input Parameter2	AHBAPB_InitStruct: pointer to an AHBAPB_InitTypeDef structure that contains the configuration information for the specified AHBAPB peripheral. Refer to section "*AHBAPB_InitTypeDef on page 288*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

AHBAPB_InitTypeDef

The AHBAPB_InitTypeDef structure is defined in the *91x_AHBAPB.h* file:

```
typedef struct
{
u32 AHBAPB_Error;
  u32 AHBAPB_SetTimeOut;
u32 AHBAPB_Split;
  u8 AHBAPB_SplitCounter;
} AHBAPB_InitTypeDef;
```

AHBAPB_SetTimeOut

Sets the Time-out, in terms of APB clock periods, that the bridge can wait for a target completion, before asserting the time-out error, allowed values are from 0 to 31. When the time-out counter = 0, is disabled.

AHBAPB_Error

Enables or disables error generation.

This member can be one of the following values:

AHBAPB_Error	Meaning
AHBAPB_Error_Enable	Enables error generation.
AHBAPB_Error_Disable	Disable error generation.

Note: If AHBAPB_Error_Disable is used, AHBAPB_SetTimeOut struct member has no effect.

AHBAPB_Split

Enables or disables accesses to be split after the number of AHB cycles.

This member can be one of the following values:

AHBAPB_Split	Meaning
AHBAPB_Split_Enable	Enable accesses to be split after the number of AHB cycles.
AHBAPB_Split_Disable	Disable accesses to be split after the number of AHB cycles. The bridge will provide the bus with HREADY or a timeout condition

Note: If AHBAPB_Split_Disable is used, AHBAPB_SplitCounter struct member has no effect.

AHBAPB_SplitCounter

Sets the number of AHB cycle to be performed before returning a split to the arbiter.

Allowed values are from 0 to 31.

Example:

```
/* Configure AHBAPB0 bridge */
AHBAPB_InitTypeDef AHBAPB_InitStructure;
AHBAPB_InitStructure.AHBAPB_SetTimeOut = 0x0F;
AHBAPB_InitStructure.AHBAPB_Error= AHBAPB_Error_Enable;
AHBAPB_InitStructure.AHBAPB_Split = AHBAPB_Split_Enable;
AHBAPB_InitStructure.AHBAPB_SplitCounter = 0x0E;
AHBAPB_Init(AHBAPB0, &AHBAPB_InitStructure);
```

20.2.3 AHBAPB_StructInit

Function Name	AHBAPB_StructInit
Function Prototype	`void AHBAPB_StructInit(AHBAPB_InitTypeDef* AHBAPB_InitStruct)`
Behavior Description	Fills each AHBAPB_InitStruct member with its reset value.
Input Parameter	AHBAPB_InitStruct: pointer to a AHBAPB_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

The AHBAPB_InitStruct members default values are as follows:

Member	Default value
AHBAPB_Split	AHBAPB_Split_Enable
AHBAPB_SplitCounter	0xFF
AHBAPB_Error	AHBAPB_Error_Enable
AHBAPB_SetTimeOut	0xFF
AHBAPB_IsoFrequency	AHBAPB_IsoFrequency_Enable

Example:

```
/*Initialize the AHBAPB0 Init Structure parameters*/
AHBAPB_InitTypeDef AHBAPB_InitStruct;
AHBAPB_StructInit(AHBAPB_InitStructure);
```

20.2.4 AHBAPB_GetFlagStatus

Function Name	AHBAPB_GetFlagStatus
Function Prototype	`GetFlagStatus AHBAPB_FlagStatus(AHBAPB_TypeDef* AHBAPBx, u8 AHBAPB_FLAG)`
Behavior Description	Checks whether the specified AHBAPB flag is set or not.
Input Parameter1	AHBAPBx: where x can be 0 or 1 to select the AHBAPB peripheral.
Input Parameter2	AHBAPB_FLAG: specifies the flag to check. Refer to section "*AHBAPB_FLAG on page 291*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of AHBAPB_FLAG (SET or RESET).
Required preconditions	None.
Called functions	None.

AHBAPB_FLAG

The AHBAPB flags that can be read are listed in the following table:

AHBAPB_FLAG	Meaning
AHBAPB_FLAG_ERROR	A previous access has been aborted because it generates an error.
AHBAPB_FLAG_OUTM	An access out of memory has been attempted.
AHBAPB_FLAG_APBT	A peripheral did not answer before the time out.
AHBAPB_FLAG_RW	The type of access that generate the error conditions: read/write.

Example:

```
/* Get the error flag status */
FlagStatus Status;
Status = AHBAPB_GetFlagStatus(AHBAPB0, AHBAPB_FLAG_ERROR);
```

20.2.5 AHBAPB_ClearFlag

Function Name	AHBAPB_ClearFlag
Function Prototype	`void AHBAPB_ClearFlag(AHBAPB_TypeDef* AHBAPBx, u8 AHBAPB_FLAG)`
Behavior Description	Clears the AHBAPBx's pending flags.
Input Parameter1	AHBAPBx: where x can be 0,1 to select the AHBAPB peripheral.
Input Parameter2	AHBAPB_FLAG: specifies the flag to clear. Refer to section "*AHBAPB_FLAG on page 291*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

AHBAPB_FLAG

To clear AHBAPB flags, use a combination of one or more of the following values:

AHBAPB_FLAG	Meaning
AHBAPB_FLAG_ERROR	A previous access has been aborted because it generates an error.
AHBAPB_FLAG_OUTM	An access out of memory has been attempted.
AHBAPB_FLAG_APBT	A peripheral did not answer before the time out.

Example:

```
/* Clear the AHBAPB0 error flag*/
AHBAPB_ClearFlag(AHBAPB0, AHBAPB_FLAG_ERROR);
```

20.2.6 AHBAPB_GetPeriphAddrError

Function Name	AHBAPB_GetPeriphAddrError
Function Prototype	`u32 AHBAPB_GetPeriphAddr(AHBAPB_TypeDef* AHBAPBx)`
Behavior Description	Gets the AHBAPB peripheral address error.
Input Parameter	AHBAPBx: where x can be 0,1 to select the AHBAPB peripheral.
Output Parameter	None
Return Parameter	AHBAPB peripheral address error.
Required preconditions	None
Called functions	None

Example:

```
/* return AHBAPB0 pheripheral address error */
AHBAPB_GetPeriphAddrError(AHBAPB0);
```

21 Revision history

Date	Revision	Changes
15-May-2006	1	Initial release.
28-May-2007	2	*Section 8: Vectored Interrupt Controller (VIC)* updated. *Section 9: Wake-Up Interrupt Unit (WIU)* updated. *Section 12: 16-bit Timer (TIM)* updated. *Section 13: DMA Controller (DMA)* updated. *Section 14: Synchronous Serial Peripheral (SSP)* updated. *Section 15: Universal Asynchronous Receiver Transmitter (UART)* updated.

I

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZE REPRESENTATIVE OF ST, ST PRODUCTS ARE NOT DESIGNED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS, WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com