gpio.h

#ifndef \_\_GPIO\_H\_\_  
#define \_\_GPIO\_H\_\_  
#ifdef \_\_cplusplus  
extern "C" {  
#endif  
#include "main.h"  
typedef struct  
{  
 int encoderA;  
 int encoderB;  
}encoder\_InitTypedef;  
extern encoder\_InitTypedef speed;  
void MX\_GPIO\_Init(void);  
#ifdef \_\_cplusplus  
}  
#endif  
#endif /\*\_\_ GPIO\_H\_\_ \*/

i2c.h

#ifndef \_\_I2C\_H\_\_  
#define \_\_I2C\_H\_\_  
#ifdef \_\_cplusplus  
extern "C" {  
#endif  
#include "main.h"  
extern I2C\_HandleTypeDef hi2c1;  
extern I2C\_HandleTypeDef hi2c2;  
void MX\_I2C1\_Init(void);  
void MX\_I2C2\_Init(void);  
#ifdef \_\_cplusplus  
}  
#endif  
#endif /\* \_\_I2C\_H\_\_ \*/

main.h

#ifndef \_\_MAIN\_H  
#define \_\_MAIN\_H  
#ifdef \_\_cplusplus  
extern "C" {  
#endif  
#include "stm32f1xx\_hal.h"  
#include "i2c.h"  
#include "spi.h"  
#include "tim.h"  
#include "usart.h"  
#include "gpio.h"  
#include <stdio.h>  
#include <stdbool.h>  
#include <string.h>  
   
#include "OLED.h"  
#include "usart1.h"  
#include "exit.h"  
#include "PS2.h"  
#include "Servo.h"  
#include "em\_alg.h"  
#include "Motor.h"  
#include "encoder.h"  
#include "pid.h"  
#include "PCA9685.h"  
void Error\_Handler(void);  
#define KEY1\_Pin GPIO\_PIN\_0  
#define KEY1\_GPIO\_Port GPIOC  
#define KEY2\_Pin GPIO\_PIN\_1  
#define KEY2\_GPIO\_Port GPIOC  
#define KEY3\_Pin GPIO\_PIN\_2  
#define KEY3\_GPIO\_Port GPIOC  
#define KEY4\_Pin GPIO\_PIN\_3  
#define KEY4\_GPIO\_Port GPIOC  
#define MOTORA\_1\_Pin GPIO\_PIN\_0  
#define MOTORA\_1\_GPIO\_Port GPIOA  
#define MOTORA\_2\_Pin GPIO\_PIN\_1  
#define MOTORA\_2\_GPIO\_Port GPIOA  
#define USART2\_TX\_Pin GPIO\_PIN\_2  
#define USART2\_TX\_GPIO\_Port GPIOA  
#define USART2\_RX\_Pin GPIO\_PIN\_3  
#define USART2\_RX\_GPIO\_Port GPIOA  
#define CS\_Pin GPIO\_PIN\_4  
#define CS\_GPIO\_Port GPIOA  
#define CLK\_Pin GPIO\_PIN\_5  
#define CLK\_GPIO\_Port GPIOA  
#define DAT\_Pin GPIO\_PIN\_6  
#define DAT\_GPIO\_Port GPIOA  
#define CMD\_Pin GPIO\_PIN\_7  
#define CMD\_GPIO\_Port GPIOA  
#define SERVO3\_Pin GPIO\_PIN\_0  
#define SERVO3\_GPIO\_Port GPIOB  
#define SERVO4\_Pin GPIO\_PIN\_1  
#define SERVO4\_GPIO\_Port GPIOB  
#define BIN2\_Pin GPIO\_PIN\_12  
#define BIN2\_GPIO\_Port GPIOB  
#define BIN1\_Pin GPIO\_PIN\_13  
#define BIN1\_GPIO\_Port GPIOB  
#define AIN1\_Pin GPIO\_PIN\_14  
#define AIN1\_GPIO\_Port GPIOB  
#define AIN2\_Pin GPIO\_PIN\_15  
#define AIN2\_GPIO\_Port GPIOB  
#define PWMA\_Pin GPIO\_PIN\_6  
#define PWMA\_GPIO\_Port GPIOC  
#define PWMB\_Pin GPIO\_PIN\_7  
#define PWMB\_GPIO\_Port GPIOC  
#define SERVO5\_Pin GPIO\_PIN\_9  
#define SERVO5\_GPIO\_Port GPIOC  
#define LED\_RED\_Pin GPIO\_PIN\_8  
#define LED\_RED\_GPIO\_Port GPIOA  
#define USART1\_TX\_Pin GPIO\_PIN\_9  
#define USART1\_TX\_GPIO\_Port GPIOA  
#define USART1\_RX\_Pin GPIO\_PIN\_10  
#define USART1\_RX\_GPIO\_Port GPIOA  
#define UART4\_TX\_Pin GPIO\_PIN\_10  
#define UART4\_TX\_GPIO\_Port GPIOC  
#define UART4\_RX\_Pin GPIO\_PIN\_11  
#define UART4\_RX\_GPIO\_Port GPIOC  
#define UART5\_TX\_Pin GPIO\_PIN\_12  
#define UART5\_TX\_GPIO\_Port GPIOC  
#define UART5\_RX\_Pin GPIO\_PIN\_2  
#define UART5\_RX\_GPIO\_Port GPIOD  
#define SERVO1\_Pin GPIO\_PIN\_4  
#define SERVO1\_GPIO\_Port GPIOB  
#define SERVO2\_Pin GPIO\_PIN\_5  
#define SERVO2\_GPIO\_Port GPIOB  
#define MOTORB\_1\_Pin GPIO\_PIN\_6  
#define MOTORB\_1\_GPIO\_Port GPIOB  
#define MOTORB\_2\_Pin GPIO\_PIN\_7  
#define MOTORB\_2\_GPIO\_Port GPIOB  
#define OLED\_SCL\_Pin GPIO\_PIN\_8  
#define OLED\_SCL\_GPIO\_Port GPIOB  
#define OLED\_SDA\_Pin GPIO\_PIN\_9  
#define OLED\_SDA\_GPIO\_Port GPIOB  
#ifdef \_\_cplusplus  
}  
#endif  
#endif /\* \_\_MAIN\_H \*/

spi.h

#ifndef \_\_SPI\_H\_\_  
#define \_\_SPI\_H\_\_  
#ifdef \_\_cplusplus  
extern "C" {  
#endif  
#include "main.h"  
extern SPI\_HandleTypeDef hspi1;  
void MX\_SPI1\_Init(void);  
#ifdef \_\_cplusplus  
}  
#endif  
#endif /\* \_\_SPI\_H\_\_ \*/

stm32f1xx\_hal\_conf.h

#ifndef \_\_STM32F1xx\_HAL\_CONF\_H  
#define \_\_STM32F1xx\_HAL\_CONF\_H  
#ifdef \_\_cplusplus  
 extern "C" {  
#endif  
#define HAL\_MODULE\_ENABLED  
#define HAL\_GPIO\_MODULE\_ENABLED  
#define HAL\_I2C\_MODULE\_ENABLED  
#define HAL\_SPI\_MODULE\_ENABLED  
#define HAL\_TIM\_MODULE\_ENABLED  
#define HAL\_UART\_MODULE\_ENABLED  
#define HAL\_CORTEX\_MODULE\_ENABLED  
#define HAL\_DMA\_MODULE\_ENABLED  
#define HAL\_FLASH\_MODULE\_ENABLED  
#define HAL\_EXTI\_MODULE\_ENABLED  
#define HAL\_GPIO\_MODULE\_ENABLED  
#define HAL\_PWR\_MODULE\_ENABLED  
#define HAL\_RCC\_MODULE\_ENABLED  
#if !defined (HSE\_VALUE)  
 #define HSE\_VALUE 8000000U /\*!< Value of the External oscillator in Hz \*/  
#endif /\* HSE\_VALUE \*/  
#if !defined (HSE\_STARTUP\_TIMEOUT)  
 #define HSE\_STARTUP\_TIMEOUT 100U /\*!< Time out for HSE start up, in ms \*/  
#endif /\* HSE\_STARTUP\_TIMEOUT \*/  
#if !defined (HSI\_VALUE)  
 #define HSI\_VALUE 8000000U /\*!< Value of the Internal oscillator in Hz\*/  
#endif /\* HSI\_VALUE \*/  
#if !defined (LSI\_VALUE)  
 #define LSI\_VALUE 40000U /\*!< LSI Typical Value in Hz \*/  
#endif /\* LSI\_VALUE \*/ /\*!< Value of the Internal Low Speed oscillator in Hz  
 The real value may vary depending on the variations  
 in voltage and temperature. \*/  
#if !defined (LSE\_VALUE)  
 #define LSE\_VALUE 32768U /\*!< Value of the External oscillator in Hz\*/  
#endif /\* LSE\_VALUE \*/  
#if !defined (LSE\_STARTUP\_TIMEOUT)  
 #define LSE\_STARTUP\_TIMEOUT 5000U /\*!< Time out for LSE start up, in ms \*/  
#endif /\* LSE\_STARTUP\_TIMEOUT \*/  
 === you can define the HSE value in your toolchain compiler preprocessor. \*/  
#define VDD\_VALUE 3300U /\*!< Value of VDD in mv \*/  
#define TICK\_INT\_PRIORITY 15U /\*!< tick interrupt priority (lowest by default) \*/  
#define USE\_RTOS 0U  
#define PREFETCH\_ENABLE 1U  
#define USE\_HAL\_ADC\_REGISTER\_CALLBACKS 0U /\* ADC register callback disabled \*/  
#define USE\_HAL\_CAN\_REGISTER\_CALLBACKS 0U /\* CAN register callback disabled \*/  
#define USE\_HAL\_CEC\_REGISTER\_CALLBACKS 0U /\* CEC register callback disabled \*/  
#define USE\_HAL\_DAC\_REGISTER\_CALLBACKS 0U /\* DAC register callback disabled \*/  
#define USE\_HAL\_ETH\_REGISTER\_CALLBACKS 0U /\* ETH register callback disabled \*/  
#define USE\_HAL\_HCD\_REGISTER\_CALLBACKS 0U /\* HCD register callback disabled \*/  
#define USE\_HAL\_I2C\_REGISTER\_CALLBACKS 0U /\* I2C register callback disabled \*/  
#define USE\_HAL\_I2S\_REGISTER\_CALLBACKS 0U /\* I2S register callback disabled \*/  
#define USE\_HAL\_MMC\_REGISTER\_CALLBACKS 0U /\* MMC register callback disabled \*/  
#define USE\_HAL\_NAND\_REGISTER\_CALLBACKS 0U /\* NAND register callback disabled \*/  
#define USE\_HAL\_NOR\_REGISTER\_CALLBACKS 0U /\* NOR register callback disabled \*/  
#define USE\_HAL\_PCCARD\_REGISTER\_CALLBACKS 0U /\* PCCARD register callback disabled \*/  
#define USE\_HAL\_PCD\_REGISTER\_CALLBACKS 0U /\* PCD register callback disabled \*/  
#define USE\_HAL\_RTC\_REGISTER\_CALLBACKS 0U /\* RTC register callback disabled \*/  
#define USE\_HAL\_SD\_REGISTER\_CALLBACKS 0U /\* SD register callback disabled \*/  
#define USE\_HAL\_SMARTCARD\_REGISTER\_CALLBACKS 0U /\* SMARTCARD register callback disabled \*/  
#define USE\_HAL\_IRDA\_REGISTER\_CALLBACKS 0U /\* IRDA register callback disabled \*/  
#define USE\_HAL\_SRAM\_REGISTER\_CALLBACKS 0U /\* SRAM register callback disabled \*/  
#define USE\_HAL\_SPI\_REGISTER\_CALLBACKS 0U /\* SPI register callback disabled \*/  
#define USE\_HAL\_TIM\_REGISTER\_CALLBACKS 0U /\* TIM register callback disabled \*/  
#define USE\_HAL\_UART\_REGISTER\_CALLBACKS 0U /\* UART register callback disabled \*/  
#define USE\_HAL\_USART\_REGISTER\_CALLBACKS 0U /\* USART register callback disabled \*/  
#define USE\_HAL\_WWDG\_REGISTER\_CALLBACKS 0U /\* WWDG register callback disabled \*/  
#define MAC\_ADDR0 2U  
#define MAC\_ADDR1 0U  
#define MAC\_ADDR2 0U  
#define MAC\_ADDR3 0U  
#define MAC\_ADDR4 0U  
#define MAC\_ADDR5 0U  
#define ETH\_RX\_BUF\_SIZE ETH\_MAX\_PACKET\_SIZE /\* buffer size for receive \*/  
#define ETH\_TX\_BUF\_SIZE ETH\_MAX\_PACKET\_SIZE /\* buffer size for transmit \*/  
#define ETH\_RXBUFNB 8U /\* 4 Rx buffers of size ETH\_RX\_BUF\_SIZE \*/  
#define ETH\_TXBUFNB 4U /\* 4 Tx buffers of size ETH\_TX\_BUF\_SIZE \*/  
#define DP83848\_PHY\_ADDRESS 0x01U  
#define PHY\_RESET\_DELAY 0x000000FFU  
#define PHY\_CONFIG\_DELAY 0x00000FFFU  
#define PHY\_READ\_TO 0x0000FFFFU  
#define PHY\_WRITE\_TO 0x0000FFFFU  
#define PHY\_BCR ((uint16\_t)0x00) /\*!< Transceiver Basic Control Register \*/  
#define PHY\_BSR ((uint16\_t)0x01) /\*!< Transceiver Basic Status Register \*/  
#define PHY\_RESET ((uint16\_t)0x8000) /\*!< PHY Reset \*/  
#define PHY\_LOOPBACK ((uint16\_t)0x4000) /\*!< Select loop-back mode \*/  
#define PHY\_FULLDUPLEX\_100M ((uint16\_t)0x2100) /\*!< Set the full-duplex mode at 100 Mb/s \*/  
#define PHY\_HALFDUPLEX\_100M ((uint16\_t)0x2000) /\*!< Set the half-duplex mode at 100 Mb/s \*/  
#define PHY\_FULLDUPLEX\_10M ((uint16\_t)0x0100) /\*!< Set the full-duplex mode at 10 Mb/s \*/  
#define PHY\_HALFDUPLEX\_10M ((uint16\_t)0x0000) /\*!< Set the half-duplex mode at 10 Mb/s \*/  
#define PHY\_AUTONEGOTIATION ((uint16\_t)0x1000) /\*!< Enable auto-negotiation function \*/  
#define PHY\_RESTART\_AUTONEGOTIATION ((uint16\_t)0x0200) /\*!< Restart auto-negotiation function \*/  
#define PHY\_POWERDOWN ((uint16\_t)0x0800) /\*!< Select the power down mode \*/  
#define PHY\_ISOLATE ((uint16\_t)0x0400) /\*!< Isolate PHY from MII \*/  
#define PHY\_AUTONEGO\_COMPLETE ((uint16\_t)0x0020) /\*!< Auto-Negotiation process completed \*/  
#define PHY\_LINKED\_STATUS ((uint16\_t)0x0004) /\*!< Valid link established \*/  
#define PHY\_JABBER\_DETECTION ((uint16\_t)0x0002) /\*!< Jabber condition detected \*/  
#define PHY\_SR ((uint16\_t)0x10U) /\*!< PHY status register Offset \*/  
#define PHY\_SPEED\_STATUS ((uint16\_t)0x0002U) /\*!< PHY Speed mask \*/  
#define PHY\_DUPLEX\_STATUS ((uint16\_t)0x0004U) /\*!< PHY Duplex mask \*/  
#define USE\_SPI\_CRC 0U  
#ifdef HAL\_RCC\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_rcc.h"  
#endif /\* HAL\_RCC\_MODULE\_ENABLED \*/  
#ifdef HAL\_GPIO\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_gpio.h"  
#endif /\* HAL\_GPIO\_MODULE\_ENABLED \*/  
#ifdef HAL\_EXTI\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_exti.h"  
#endif /\* HAL\_EXTI\_MODULE\_ENABLED \*/  
#ifdef HAL\_DMA\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_dma.h"  
#endif /\* HAL\_DMA\_MODULE\_ENABLED \*/  
#ifdef HAL\_ETH\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_eth.h"  
#endif /\* HAL\_ETH\_MODULE\_ENABLED \*/  
#ifdef HAL\_CAN\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_can.h"  
#endif /\* HAL\_CAN\_MODULE\_ENABLED \*/  
#ifdef HAL\_CAN\_LEGACY\_MODULE\_ENABLED  
 #include "Legacy/stm32f1xx\_hal\_can\_legacy.h"  
#endif /\* HAL\_CAN\_LEGACY\_MODULE\_ENABLED \*/  
#ifdef HAL\_CEC\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_cec.h"  
#endif /\* HAL\_CEC\_MODULE\_ENABLED \*/  
#ifdef HAL\_CORTEX\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_cortex.h"  
#endif /\* HAL\_CORTEX\_MODULE\_ENABLED \*/  
#ifdef HAL\_ADC\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_adc.h"  
#endif /\* HAL\_ADC\_MODULE\_ENABLED \*/  
#ifdef HAL\_CRC\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_crc.h"  
#endif /\* HAL\_CRC\_MODULE\_ENABLED \*/  
#ifdef HAL\_DAC\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_dac.h"  
#endif /\* HAL\_DAC\_MODULE\_ENABLED \*/  
#ifdef HAL\_FLASH\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_flash.h"  
#endif /\* HAL\_FLASH\_MODULE\_ENABLED \*/  
#ifdef HAL\_SRAM\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_sram.h"  
#endif /\* HAL\_SRAM\_MODULE\_ENABLED \*/  
#ifdef HAL\_NOR\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_nor.h"  
#endif /\* HAL\_NOR\_MODULE\_ENABLED \*/  
#ifdef HAL\_I2C\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_i2c.h"  
#endif /\* HAL\_I2C\_MODULE\_ENABLED \*/  
#ifdef HAL\_I2S\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_i2s.h"  
#endif /\* HAL\_I2S\_MODULE\_ENABLED \*/  
#ifdef HAL\_IWDG\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_iwdg.h"  
#endif /\* HAL\_IWDG\_MODULE\_ENABLED \*/  
#ifdef HAL\_PWR\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_pwr.h"  
#endif /\* HAL\_PWR\_MODULE\_ENABLED \*/  
#ifdef HAL\_RTC\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_rtc.h"  
#endif /\* HAL\_RTC\_MODULE\_ENABLED \*/  
#ifdef HAL\_PCCARD\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_pccard.h"  
#endif /\* HAL\_PCCARD\_MODULE\_ENABLED \*/  
#ifdef HAL\_SD\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_sd.h"  
#endif /\* HAL\_SD\_MODULE\_ENABLED \*/  
#ifdef HAL\_NAND\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_nand.h"  
#endif /\* HAL\_NAND\_MODULE\_ENABLED \*/  
#ifdef HAL\_SPI\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_spi.h"  
#endif /\* HAL\_SPI\_MODULE\_ENABLED \*/  
#ifdef HAL\_TIM\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_tim.h"  
#endif /\* HAL\_TIM\_MODULE\_ENABLED \*/  
#ifdef HAL\_UART\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_uart.h"  
#endif /\* HAL\_UART\_MODULE\_ENABLED \*/  
#ifdef HAL\_USART\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_usart.h"  
#endif /\* HAL\_USART\_MODULE\_ENABLED \*/  
#ifdef HAL\_IRDA\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_irda.h"  
#endif /\* HAL\_IRDA\_MODULE\_ENABLED \*/  
#ifdef HAL\_SMARTCARD\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_smartcard.h"  
#endif /\* HAL\_SMARTCARD\_MODULE\_ENABLED \*/  
#ifdef HAL\_WWDG\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_wwdg.h"  
#endif /\* HAL\_WWDG\_MODULE\_ENABLED \*/  
#ifdef HAL\_PCD\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_pcd.h"  
#endif /\* HAL\_PCD\_MODULE\_ENABLED \*/  
#ifdef HAL\_HCD\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_hcd.h"  
#endif /\* HAL\_HCD\_MODULE\_ENABLED \*/  
#ifdef HAL\_MMC\_MODULE\_ENABLED  
#include "stm32f1xx\_hal\_mmc.h"  
#endif /\* HAL\_MMC\_MODULE\_ENABLED \*/  
#ifdef USE\_FULL\_ASSERT  
#define assert\_param(expr) ((expr) ? (void)0U : assert\_failed((uint8\_t \*)\_\_FILE\_\_, \_\_LINE\_\_))  
void assert\_failed(uint8\_t\* file, uint32\_t line);  
#else  
#define assert\_param(expr) ((void)0U)  
#endif /\* USE\_FULL\_ASSERT \*/  
#ifdef \_\_cplusplus  
}  
#endif  
#endif /\* \_\_STM32F1xx\_HAL\_CONF\_H \*/

stm32f1xx\_it.h

#ifndef \_\_STM32F1xx\_IT\_H  
#define \_\_STM32F1xx\_IT\_H  
#ifdef \_\_cplusplus  
 extern "C" {  
#endif  
void NMI\_Handler(void);  
void HardFault\_Handler(void);  
void MemManage\_Handler(void);  
void BusFault\_Handler(void);  
void UsageFault\_Handler(void);  
void SVC\_Handler(void);  
void DebugMon\_Handler(void);  
void PendSV\_Handler(void);  
void SysTick\_Handler(void);  
void TIM1\_UP\_IRQHandler(void);  
void TIM1\_TRG\_COM\_IRQHandler(void);  
void TIM2\_IRQHandler(void);  
void TIM3\_IRQHandler(void);  
void TIM4\_IRQHandler(void);  
void USART1\_IRQHandler(void);  
void TIM8\_UP\_IRQHandler(void);  
void TIM8\_TRG\_COM\_IRQHandler(void);  
#ifdef \_\_cplusplus  
}  
#endif  
#endif /\* \_\_STM32F1xx\_IT\_H \*/

tim.h

#ifndef \_\_TIM\_H\_\_  
#define \_\_TIM\_H\_\_  
#ifdef \_\_cplusplus  
extern "C" {  
#endif  
#include "main.h"  
extern TIM\_HandleTypeDef htim1;  
extern TIM\_HandleTypeDef htim2;  
extern TIM\_HandleTypeDef htim3;  
extern TIM\_HandleTypeDef htim4;  
extern TIM\_HandleTypeDef htim8;  
void MX\_TIM1\_Init(void);  
void MX\_TIM2\_Init(void);  
void MX\_TIM3\_Init(void);  
void MX\_TIM4\_Init(void);  
void MX\_TIM8\_Init(void);  
void HAL\_TIM\_MspPostInit(TIM\_HandleTypeDef \*htim);  
#ifdef \_\_cplusplus  
}  
#endif  
#endif /\* \_\_TIM\_H\_\_ \*/

usart.h

#ifndef \_\_USART\_H\_\_  
#define \_\_USART\_H\_\_  
#ifdef \_\_cplusplus  
extern "C" {  
#endif  
#include "main.h"  
extern UART\_HandleTypeDef huart4;  
extern UART\_HandleTypeDef huart5;  
extern UART\_HandleTypeDef huart1;  
extern UART\_HandleTypeDef huart2;  
void MX\_UART4\_Init(void);  
void MX\_UART5\_Init(void);  
void MX\_USART1\_UART\_Init(void);  
void MX\_USART2\_UART\_Init(void);  
#ifdef \_\_cplusplus  
}  
#endif  
#endif /\* \_\_USART\_H\_\_ \*/

em\_alg.h

#ifndef \_\_EM\_ALG\_H\_\_  
#define \_\_EM\_ALG\_H\_\_  
#include <math.h>  
#include "main.h"  
#define M\_PI 3.14  
void alg\_Init(void);  
void alg\_positive\_operation(float angleA,float angleB,float angleC);  
void inverse\_operation(float x,float y,float z);  
void alg\_set\_move\_action(int \*data);  
void alg\_move\_run(void);  
void alg\_Yolo(int YoloX,int YoloY,int Percent);  
#endif

encoder.h

#ifndef \_\_ENCODER\_H\_\_  
#define \_\_ENCODER\_H\_\_  
#include "main.h"  
float Read\_EncoderA(void);  
float Read\_EncoderB(void);  
float Read\_SPEEDA(void);  
float Read\_SPEEDB(void);  
#endif

exit.h

#ifndef \_\_EXIT\_H\_\_  
#define \_\_EXIT\_H\_\_  
#include "main.h"  
#define KEY1\_\_4\_CLK\_ENABLE() \_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE()  
#define KEY1\_Read HAL\_GPIO\_ReadPin(KEY1\_GPIO\_Port,KEY1\_Pin)  
#define KEY2\_Read HAL\_GPIO\_ReadPin(KEY2\_GPIO\_Port,KEY2\_Pin)  
#define KEY3\_Read HAL\_GPIO\_ReadPin(KEY3\_GPIO\_Port,KEY3\_Pin)  
#define KEY4\_Read HAL\_GPIO\_ReadPin(KEY4\_GPIO\_Port,KEY4\_Pin)  
typedef struct  
{  
 uint32\_t KEY1\_val;  
 uint32\_t KEY2\_val;  
 uint32\_t KEY3\_val;  
 uint32\_t KEY4\_val;  
} KEY;  
extern KEY KEY\_t;  
void KEY\_EXIT\_Init(void);  
uint8\_t GetKEY1val(void);  
uint8\_t GetKEY2val(void);  
uint8\_t GetKEY3val(void);  
uint8\_t GetKEY4val(void);  
#endif

IIC.h

#ifndef \_\_IIC\_H  
#define \_\_IIC\_H  
void PCA\_Init(void);  
#endif

Motor.h

#ifndef \_\_MOTOR\_H  
#define \_\_MOTOR\_H  
#include "main.h"  
#define AIN1HIGH HAL\_GPIO\_WritePin(AIN1\_GPIO\_Port,AIN1\_Pin,GPIO\_PIN\_SET)  
#define AIN1LOW HAL\_GPIO\_WritePin(AIN1\_GPIO\_Port,AIN1\_Pin,GPIO\_PIN\_RESET)  
#define AIN2HIGH HAL\_GPIO\_WritePin(AIN2\_GPIO\_Port,AIN2\_Pin,GPIO\_PIN\_SET)  
#define AIN2LOW HAL\_GPIO\_WritePin(AIN2\_GPIO\_Port,AIN2\_Pin,GPIO\_PIN\_RESET)  
#define BIN1HIGH HAL\_GPIO\_WritePin(BIN1\_GPIO\_Port,BIN1\_Pin,GPIO\_PIN\_SET)  
#define BIN1LOW HAL\_GPIO\_WritePin(BIN1\_GPIO\_Port,BIN1\_Pin,GPIO\_PIN\_RESET)  
#define BIN2HIGH HAL\_GPIO\_WritePin(BIN2\_GPIO\_Port,BIN2\_Pin,GPIO\_PIN\_SET)  
#define BIN2LOW HAL\_GPIO\_WritePin(BIN2\_GPIO\_Port,BIN2\_Pin,GPIO\_PIN\_RESET)  
#define Limit\_Data 500  
void Limit(int \*motorA,int \*motorB );  
int abs(int p);  
void motor\_speed(int motorA,int motorB);  
void stop(void);  
void step\_run(void);  
#endif

OLED.h

#ifndef \_\_OLED\_H  
#define \_\_OLED\_H  
void OLED\_Init(void);  
void OLED\_Clear(void);  
void OLED\_ShowChar(uint8\_t Line, uint8\_t Column, char Char);  
void OLED\_ShowString(uint8\_t Line, uint8\_t Column, char \*String);  
void OLED\_ShowNum(uint8\_t Line, uint8\_t Column, uint32\_t Number, uint8\_t Length);  
void OLED\_ShowSignedNum(uint8\_t Line, uint8\_t Column, int32\_t Number, uint8\_t Length);  
void OLED\_ShowHexNum(uint8\_t Line, uint8\_t Column, uint32\_t Number, uint8\_t Length);  
void OLED\_ShowBinNum(uint8\_t Line, uint8\_t Column, uint32\_t Number, uint8\_t Length);  
void OLED\_Clear\_Part(uint8\_t Line, uint8\_t start, uint8\_t end);  
void OLED\_ShowWord(uint8\_t Line, uint8\_t Column, uint8\_t Chinese);  
void OLED\_ShowChinese(uint8\_t Line, uint8\_t Column, uint8\_t \*Chinese,uint8\_t Length);  
#endif

OLED\_Font.h

#ifndef \_\_OLED\_FONT\_H  
#define \_\_OLED\_FONT\_H  
const uint8\_t OLED\_F8x16[][16]=  
{  
 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  
 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,// 0  
   
 0x00,0x00,0x00,0xF8,0x00,0x00,0x00,0x00,  
 0x00,0x00,0x00,0x33,0x30,0x00,0x00,0x00,//! 1  
   
 0x00,0x10,0x0C,0x06,0x10,0x0C,0x06,0x00,  
 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//" 2  
   
 0x40,0xC0,0x78,0x40,0xC0,0x78,0x40,0x00,  
 0x04,0x3F,0x04,0x04,0x3F,0x04,0x04,0x00,//# 3  
   
 0x00,0x70,0x88,0xFC,0x08,0x30,0x00,0x00,  
 0x00,0x18,0x20,0xFF,0x21,0x1E,0x00,0x00,//$ 4  
   
 0xF0,0x08,0xF0,0x00,0xE0,0x18,0x00,0x00,  
 0x00,0x21,0x1C,0x03,0x1E,0x21,0x1E,0x00,//% 5  
   
 0x00,0xF0,0x08,0x88,0x70,0x00,0x00,0x00,  
 0x1E,0x21,0x23,0x24,0x19,0x27,0x21,0x10,//& 6  
   
 0x10,0x16,0x0E,0x00,0x00,0x00,0x00,0x00,  
 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//' 7  
   
 0x00,0x00,0x00,0xE0,0x18,0x04,0x02,0x00,  
 0x00,0x00,0x00,0x07,0x18,0x20,0x40,0x00,//( 8  
   
 0x00,0x02,0x04,0x18,0xE0,0x00,0x00,0x00,  
 0x00,0x40,0x20,0x18,0x07,0x00,0x00,0x00,//) 9  
   
 0x40,0x40,0x80,0xF0,0x80,0x40,0x40,0x00,  
 0x02,0x02,0x01,0x0F,0x01,0x02,0x02,0x00,//\* 10  
   
 0x00,0x00,0x00,0xF0,0x00,0x00,0x00,0x00,  
 0x01,0x01,0x01,0x1F,0x01,0x01,0x01,0x00,//+ 11  
   
 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  
 0x80,0xB0,0x70,0x00,0x00,0x00,0x00,0x00,//, 12  
   
 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  
 0x00,0x01,0x01,0x01,0x01,0x01,0x01,0x01,//- 13  
   
 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  
 0x00,0x30,0x30,0x00,0x00,0x00,0x00,0x00,//. 14  
   
 0x00,0x00,0x00,0x00,0x80,0x60,0x18,0x04,  
 0x00,0x60,0x18,0x06,0x01,0x00,0x00,0x00,/// 15  
   
 0x00,0xE0,0x10,0x08,0x08,0x10,0xE0,0x00,  
 0x00,0x0F,0x10,0x20,0x20,0x10,0x0F,0x00,//0 16  
   
 0x00,0x10,0x10,0xF8,0x00,0x00,0x00,0x00,  
 0x00,0x20,0x20,0x3F,0x20,0x20,0x00,0x00,//1 17  
   
 0x00,0x70,0x08,0x08,0x08,0x88,0x70,0x00,  
 0x00,0x30,0x28,0x24,0x22,0x21,0x30,0x00,//2 18  
   
 0x00,0x30,0x08,0x88,0x88,0x48,0x30,0x00,  
 0x00,0x18,0x20,0x20,0x20,0x11,0x0E,0x00,//3 19  
   
 0x00,0x00,0xC0,0x20,0x10,0xF8,0x00,0x00,  
 0x00,0x07,0x04,0x24,0x24,0x3F,0x24,0x00,//4 20  
   
 0x00,0xF8,0x08,0x88,0x88,0x08,0x08,0x00,  
 0x00,0x19,0x21,0x20,0x20,0x11,0x0E,0x00,//5 21  
   
 0x00,0xE0,0x10,0x88,0x88,0x18,0x00,0x00,  
 0x00,0x0F,0x11,0x20,0x20,0x11,0x0E,0x00,//6 22  
   
 0x00,0x38,0x08,0x08,0xC8,0x38,0x08,0x00,  
 0x00,0x00,0x00,0x3F,0x00,0x00,0x00,0x00,//7 23  
   
 0x00,0x70,0x88,0x08,0x08,0x88,0x70,0x00,  
 0x00,0x1C,0x22,0x21,0x21,0x22,0x1C,0x00,//8 24  
   
 0x00,0xE0,0x10,0x08,0x08,0x10,0xE0,0x00,  
 0x00,0x00,0x31,0x22,0x22,0x11,0x0F,0x00,//9 25  
   
 0x00,0x00,0x00,0xC0,0xC0,0x00,0x00,0x00,  
 0x00,0x00,0x00,0x30,0x30,0x00,0x00,0x00,//: 26  
   
 0x00,0x00,0x00,0x80,0x00,0x00,0x00,0x00,  
 0x00,0x00,0x80,0x60,0x00,0x00,0x00,0x00,//; 27  
   
 0x00,0x00,0x80,0x40,0x20,0x10,0x08,0x00,  
 0x00,0x01,0x02,0x04,0x08,0x10,0x20,0x00,//< 28  
   
 0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x00,  
 0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x00,//= 29  
   
 0x00,0x08,0x10,0x20,0x40,0x80,0x00,0x00,  
 0x00,0x20,0x10,0x08,0x04,0x02,0x01,0x00,//> 30  
   
 0x00,0x70,0x48,0x08,0x08,0x08,0xF0,0x00,  
 0x00,0x00,0x00,0x30,0x36,0x01,0x00,0x00,//? 31  
   
 0xC0,0x30,0xC8,0x28,0xE8,0x10,0xE0,0x00,  
 0x07,0x18,0x27,0x24,0x23,0x14,0x0B,0x00,//@ 32  
   
 0x00,0x00,0xC0,0x38,0xE0,0x00,0x00,0x00,  
 0x20,0x3C,0x23,0x02,0x02,0x27,0x38,0x20,//A 33  
   
 0x08,0xF8,0x88,0x88,0x88,0x70,0x00,0x00,  
 0x20,0x3F,0x20,0x20,0x20,0x11,0x0E,0x00,//B 34  
   
 0xC0,0x30,0x08,0x08,0x08,0x08,0x38,0x00,  
 0x07,0x18,0x20,0x20,0x20,0x10,0x08,0x00,//C 35  
   
 0x08,0xF8,0x08,0x08,0x08,0x10,0xE0,0x00,  
 0x20,0x3F,0x20,0x20,0x20,0x10,0x0F,0x00,//D 36  
   
 0x08,0xF8,0x88,0x88,0xE8,0x08,0x10,0x00,  
 0x20,0x3F,0x20,0x20,0x23,0x20,0x18,0x00,//E 37  
   
 0x08,0xF8,0x88,0x88,0xE8,0x08,0x10,0x00,  
 0x20,0x3F,0x20,0x00,0x03,0x00,0x00,0x00,//F 38  
   
 0xC0,0x30,0x08,0x08,0x08,0x38,0x00,0x00,  
 0x07,0x18,0x20,0x20,0x22,0x1E,0x02,0x00,//G 39  
   
 0x08,0xF8,0x08,0x00,0x00,0x08,0xF8,0x08,  
 0x20,0x3F,0x21,0x01,0x01,0x21,0x3F,0x20,//H 40  
   
 0x00,0x08,0x08,0xF8,0x08,0x08,0x00,0x00,  
 0x00,0x20,0x20,0x3F,0x20,0x20,0x00,0x00,//I 41  
   
 0x00,0x00,0x08,0x08,0xF8,0x08,0x08,0x00,  
 0xC0,0x80,0x80,0x80,0x7F,0x00,0x00,0x00,//J 42  
   
 0x08,0xF8,0x88,0xC0,0x28,0x18,0x08,0x00,  
 0x20,0x3F,0x20,0x01,0x26,0x38,0x20,0x00,//K 43  
   
 0x08,0xF8,0x08,0x00,0x00,0x00,0x00,0x00,  
 0x20,0x3F,0x20,0x20,0x20,0x20,0x30,0x00,//L 44  
   
 0x08,0xF8,0xF8,0x00,0xF8,0xF8,0x08,0x00,  
 0x20,0x3F,0x00,0x3F,0x00,0x3F,0x20,0x00,//M 45  
   
 0x08,0xF8,0x30,0xC0,0x00,0x08,0xF8,0x08,  
 0x20,0x3F,0x20,0x00,0x07,0x18,0x3F,0x00,//N 46  
   
 0xE0,0x10,0x08,0x08,0x08,0x10,0xE0,0x00,  
 0x0F,0x10,0x20,0x20,0x20,0x10,0x0F,0x00,//O 47  
   
 0x08,0xF8,0x08,0x08,0x08,0x08,0xF0,0x00,  
 0x20,0x3F,0x21,0x01,0x01,0x01,0x00,0x00,//P 48  
   
 0xE0,0x10,0x08,0x08,0x08,0x10,0xE0,0x00,  
 0x0F,0x18,0x24,0x24,0x38,0x50,0x4F,0x00,//Q 49  
   
 0x08,0xF8,0x88,0x88,0x88,0x88,0x70,0x00,  
 0x20,0x3F,0x20,0x00,0x03,0x0C,0x30,0x20,//R 50  
   
 0x00,0x70,0x88,0x08,0x08,0x08,0x38,0x00,  
 0x00,0x38,0x20,0x21,0x21,0x22,0x1C,0x00,//S 51  
   
 0x18,0x08,0x08,0xF8,0x08,0x08,0x18,0x00,  
 0x00,0x00,0x20,0x3F,0x20,0x00,0x00,0x00,//T 52  
   
 0x08,0xF8,0x08,0x00,0x00,0x08,0xF8,0x08,  
 0x00,0x1F,0x20,0x20,0x20,0x20,0x1F,0x00,//U 53  
   
 0x08,0x78,0x88,0x00,0x00,0xC8,0x38,0x08,  
 0x00,0x00,0x07,0x38,0x0E,0x01,0x00,0x00,//V 54  
   
 0xF8,0x08,0x00,0xF8,0x00,0x08,0xF8,0x00,  
 0x03,0x3C,0x07,0x00,0x07,0x3C,0x03,0x00,//W 55  
   
 0x08,0x18,0x68,0x80,0x80,0x68,0x18,0x08,  
 0x20,0x30,0x2C,0x03,0x03,0x2C,0x30,0x20,//X 56  
   
 0x08,0x38,0xC8,0x00,0xC8,0x38,0x08,0x00,  
 0x00,0x00,0x20,0x3F,0x20,0x00,0x00,0x00,//Y 57  
   
 0x10,0x08,0x08,0x08,0xC8,0x38,0x08,0x00,  
 0x20,0x38,0x26,0x21,0x20,0x20,0x18,0x00,//Z 58  
   
 0x00,0x00,0x00,0xFE,0x02,0x02,0x02,0x00,  
 0x00,0x00,0x00,0x7F,0x40,0x40,0x40,0x00,//[ 59  
   
 0x00,0x0C,0x30,0xC0,0x00,0x00,0x00,0x00,  
 0x00,0x00,0x00,0x01,0x06,0x38,0xC0,0x00,//\ 60  
   
 0x00,0x02,0x02,0x02,0xFE,0x00,0x00,0x00,  
 0x00,0x40,0x40,0x40,0x7F,0x00,0x00,0x00,//] 61  
   
 0x00,0x00,0x04,0x02,0x02,0x02,0x04,0x00,  
 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//^ 62  
   
 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  
 0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x80,//\_ 63  
   
 0x00,0x02,0x02,0x04,0x00,0x00,0x00,0x00,  
 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//` 64  
   
 0x00,0x00,0x80,0x80,0x80,0x80,0x00,0x00,  
 0x00,0x19,0x24,0x22,0x22,0x22,0x3F,0x20,//a 65  
   
 0x08,0xF8,0x00,0x80,0x80,0x00,0x00,0x00,  
 0x00,0x3F,0x11,0x20,0x20,0x11,0x0E,0x00,//b 66  
   
 0x00,0x00,0x00,0x80,0x80,0x80,0x00,0x00,  
 0x00,0x0E,0x11,0x20,0x20,0x20,0x11,0x00,//c 67  
   
 0x00,0x00,0x00,0x80,0x80,0x88,0xF8,0x00,  
 0x00,0x0E,0x11,0x20,0x20,0x10,0x3F,0x20,//d 68  
   
 0x00,0x00,0x80,0x80,0x80,0x80,0x00,0x00,  
 0x00,0x1F,0x22,0x22,0x22,0x22,0x13,0x00,//e 69  
   
 0x00,0x80,0x80,0xF0,0x88,0x88,0x88,0x18,  
 0x00,0x20,0x20,0x3F,0x20,0x20,0x00,0x00,//f 70  
   
 0x00,0x00,0x80,0x80,0x80,0x80,0x80,0x00,  
 0x00,0x6B,0x94,0x94,0x94,0x93,0x60,0x00,//g 71  
   
 0x08,0xF8,0x00,0x80,0x80,0x80,0x00,0x00,  
 0x20,0x3F,0x21,0x00,0x00,0x20,0x3F,0x20,//h 72  
   
 0x00,0x80,0x98,0x98,0x00,0x00,0x00,0x00,  
 0x00,0x20,0x20,0x3F,0x20,0x20,0x00,0x00,//i 73  
   
 0x00,0x00,0x00,0x80,0x98,0x98,0x00,0x00,  
 0x00,0xC0,0x80,0x80,0x80,0x7F,0x00,0x00,//j 74  
   
 0x08,0xF8,0x00,0x00,0x80,0x80,0x80,0x00,  
 0x20,0x3F,0x24,0x02,0x2D,0x30,0x20,0x00,//k 75  
   
 0x00,0x08,0x08,0xF8,0x00,0x00,0x00,0x00,  
 0x00,0x20,0x20,0x3F,0x20,0x20,0x00,0x00,//l 76  
   
 0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x00,  
 0x20,0x3F,0x20,0x00,0x3F,0x20,0x00,0x3F,//m 77  
   
 0x80,0x80,0x00,0x80,0x80,0x80,0x00,0x00,  
 0x20,0x3F,0x21,0x00,0x00,0x20,0x3F,0x20,//n 78  
   
 0x00,0x00,0x80,0x80,0x80,0x80,0x00,0x00,  
 0x00,0x1F,0x20,0x20,0x20,0x20,0x1F,0x00,//o 79  
   
 0x80,0x80,0x00,0x80,0x80,0x00,0x00,0x00,  
 0x80,0xFF,0xA1,0x20,0x20,0x11,0x0E,0x00,//p 80  
   
 0x00,0x00,0x00,0x80,0x80,0x80,0x80,0x00,  
 0x00,0x0E,0x11,0x20,0x20,0xA0,0xFF,0x80,//q 81  
   
 0x80,0x80,0x80,0x00,0x80,0x80,0x80,0x00,  
 0x20,0x20,0x3F,0x21,0x20,0x00,0x01,0x00,//r 82  
   
 0x00,0x00,0x80,0x80,0x80,0x80,0x80,0x00,  
 0x00,0x33,0x24,0x24,0x24,0x24,0x19,0x00,//s 83  
   
 0x00,0x80,0x80,0xE0,0x80,0x80,0x00,0x00,  
 0x00,0x00,0x00,0x1F,0x20,0x20,0x00,0x00,//t 84  
   
 0x80,0x80,0x00,0x00,0x00,0x80,0x80,0x00,  
 0x00,0x1F,0x20,0x20,0x20,0x10,0x3F,0x20,//u 85  
   
 0x80,0x80,0x80,0x00,0x00,0x80,0x80,0x80,  
 0x00,0x01,0x0E,0x30,0x08,0x06,0x01,0x00,//v 86  
   
 0x80,0x80,0x00,0x80,0x00,0x80,0x80,0x80,  
 0x0F,0x30,0x0C,0x03,0x0C,0x30,0x0F,0x00,//w 87  
   
 0x00,0x80,0x80,0x00,0x80,0x80,0x80,0x00,  
 0x00,0x20,0x31,0x2E,0x0E,0x31,0x20,0x00,//x 88  
   
 0x80,0x80,0x80,0x00,0x00,0x80,0x80,0x80,  
 0x80,0x81,0x8E,0x70,0x18,0x06,0x01,0x00,//y 89  
   
 0x00,0x80,0x80,0x80,0x80,0x80,0x80,0x00,  
 0x00,0x21,0x30,0x2C,0x22,0x21,0x30,0x00,//z 90  
   
 0x00,0x00,0x00,0x00,0x80,0x7C,0x02,0x02,  
 0x00,0x00,0x00,0x00,0x00,0x3F,0x40,0x40,//{ 91  
   
 0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,  
 0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,//| 92  
   
 0x00,0x02,0x02,0x7C,0x80,0x00,0x00,0x00,  
 0x00,0x40,0x40,0x3F,0x00,0x00,0x00,0x00,//} 93  
   
 0x00,0x06,0x01,0x01,0x02,0x02,0x04,0x04,  
 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//~ 94  
};  
const uint8\_t OLED\_F16x16[][32] =  
 {  
 {0x10, 0x60, 0x02, 0x8C, 0x00, 0x00, 0xFE, 0x92, 0x92, 0x92, 0x92, 0x92, 0xFE, 0x00, 0x00, 0x00, 0x04, 0x04, 0x7E, 0x01, 0x40, 0x7E, 0x42, 0x42, 0x7E, 0x42, 0x7E, 0x42, 0x42, 0x7E, 0x40, 0x00}, /\*"温",0\*/  
 {0x00, 0x00, 0xFC, 0x24, 0x24, 0x24, 0xFC, 0x25, 0x26, 0x24, 0xFC, 0x24, 0x24, 0x24, 0x04, 0x00, 0x40, 0x30, 0x8F, 0x80, 0x84, 0x4C, 0x55, 0x25, 0x25, 0x25, 0x55, 0x4C, 0x80, 0x80, 0x80, 0x00}, /\*"度",1\*/  
 {0x10, 0x60, 0x02, 0x8C, 0x00, 0xFE, 0x92, 0x92, 0x92, 0x92, 0x92, 0x92, 0xFE, 0x00, 0x00, 0x00, 0x04, 0x04, 0x7E, 0x01, 0x44, 0x48, 0x50, 0x7F, 0x40, 0x40, 0x7F, 0x50, 0x48, 0x44, 0x40, 0x00}, /\*"湿",2\*/  
 {0x00, 0x00, 0xFC, 0x24, 0x24, 0x24, 0xFC, 0x25, 0x26, 0x24, 0xFC, 0x24, 0x24, 0x24, 0x04, 0x00, 0x40, 0x30, 0x8F, 0x80, 0x84, 0x4C, 0x55, 0x25, 0x25, 0x25, 0x55, 0x4C, 0x80, 0x80, 0x80, 0x00}, /\*"度",3\*/  
 {0x06, 0x09, 0x09, 0xE6, 0xF8, 0x0C, 0x04, 0x02, 0x02, 0x02, 0x02, 0x02, 0x04, 0x1E, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x1F, 0x30, 0x20, 0x40, 0x40, 0x40, 0x40, 0x40, 0x20, 0x10, 0x00, 0x00}, /\*"℃",4\*/  
};  
#endif

PCA9685.h

#ifndef \_\_PCA9685\_H  
#define \_\_PCA9685\_H   
#include "main.h"  
#define pca\_adrr 0x80  
#define pca\_mode1 0x0  
#define pca\_pre 0xFE  
#define LED0\_ON\_L 0x6  
#define LED0\_ON\_H 0x7  
#define LED0\_OFF\_L 0x8  
#define LED0\_OFF\_H 0x9  
#define jdMIN 115 // minimum  
#define jdMAX 590 // maximum  
#define jd000 130 //0�ȶ�Ӧ4096���������ֵ  
#define jd180 520 //180�ȶ�Ӧ4096���������ֵ  
void pca\_write(uint8\_t adrr,uint8\_t data);  
uint8\_t pca\_read(uint8\_t adrr);  
void PCA\_MG9XX\_Init(float hz,uint8\_t angle);  
void pca\_setfreq(float freq);  
void pca\_setpwm(uint8\_t num, uint32\_t on, uint32\_t off);  
void PCA\_MG9XX(uint8\_t num,uint8\_t end\_angle);  
#endif

pid.h

#ifndef \_\_PID\_H   
#define \_\_PID\_H  
#include "main.h"  
#include "gpio.h"  
#define MAX\_INTEGRAL\_ERROR 200  
typedef struct  
{  
 float MotorA\_Velocity\_Kp;//�ٶȻ�Kp  
 float MotorA\_Velocity\_Ki;//�ٶȻ�Ki  
 float MotorA\_Velocity\_Kd;//�ٶȻ�Kd  
 float MotorA\_Velocity\_Target\_Val;//����ֵ  
 float MotorA\_Velocity\_Actual\_Val;//ʵ��ֵ  
 int MotorA\_Velocity\_Out;//�ٶȻ��������  
   
 float MotorB\_Velocity\_Kp;//�ٶȻ�Kp  
 float MotorB\_Velocity\_Ki;//�ٶȻ�Ki  
 float MotorB\_Velocity\_Kd;//�ٶȻ�Kd  
 float MotorB\_Velocity\_Target\_Val;//����ֵ  
 float MotorB\_Velocity\_Actual\_Val;//ʵ��ֵ  
 int MotorB\_Velocity\_Out;//�ٶȻ��������  
}PID\_InitTypedef;  
extern PID\_InitTypedef PID;  
extern void PID\_Param\_Init(void);  
extern float VelocityRing\_PID\_MOTORA\_Realize(float Velocity\_Actual\_Val);  
extern float VelocityRing\_PID\_MOTORB\_Realize(float Velocity\_Actual\_Val);  
extern float VelocityRing\_MOTORA\_Control(void);  
extern float VelocityRing\_MOTORB\_Control(void);  
#endif

PS2.h

#ifndef \_\_PS2\_H\_\_  
#define \_\_PS2\_H\_\_  
#include "main.h"  
#define PSS\_Lx 2 //��ҡ��X������  
#define PSS\_Ly 3  
#define PSS\_Rx 0  
#define PSS\_Ry 1  
#define PSB\_Left 0  
#define PSB\_Down 1  
#define PSB\_Right 2  
#define PSB\_Up 3  
#define PSB\_Start 4  
#define PSB\_Select 7  
#define PSB\_Square 8  
#define PSB\_Cross 9  
#define PSB\_Circle 10  
#define PSB\_Triangle 11  
#define PSB\_R1 12  
#define PSB\_L1 13  
#define PSB\_R2 14  
#define PSB\_L2 15  
typedef struct  
{  
 int32\_t LeftX;  
 int32\_t LeftY;  
 int32\_t RightX;  
 int32\_t RightY;  
} PS2COORD;  
extern PS2COORD COORD;  
typedef struct  
{  
 uint32\_t MODE0;  
 uint32\_t MODE1;  
 uint32\_t MODE2;  
 uint32\_t MODE3;  
 uint32\_t MODEFLAT;  
} PS2MODE;  
extern PS2MODE MODE;  
void PS2\_Get(void); //��ȡԭʼ����  
void delay\_us(uint32\_t udelay); //��������ӳ�  
void GetData(void); //�ܺ���  
void GetXY(void); //��ԭʼ������xy������ת����0-1000  
void CLear\_Date(void);//�������  
void All\_Button(void);//��ÿһ������״̬����������ȫ����״̬�洢  
void PS2Data(void);  
void COORD0(void);  
void COORD500(void);  
#endif

PWM\_motor.h

#ifndef \_\_PWM\_MOTOR\_H\_\_  
#define \_\_PWM\_MOTOR\_H\_\_  
#include "main.h"  
void PWM8\_SetCH1(uint16\_t Compare);  
void PWM8\_SetCH2(uint16\_t Compare);  
#endif

Servo.h

#ifndef \_\_SERVO\_H\_\_  
#define \_\_SERVO\_H\_\_  
#include "main.h"  
void Servo1\_SetAngle(float Angle1);  
void Servo2\_SetAngle(float Angle2);  
void Servo3\_SetAngle(float Angle3);  
void Servo4\_SetAngle(float Angle1);  
void PS2Servo(void);  
#endif

usart1.h

#ifndef \_\_USART1\_H\_\_  
#define \_\_USART1\_H\_\_  
#include "main.h"  
#define UART1\_BUFFER\_SIZE 100 // ��������С  
#define END\_CHAR '\n' // ��������  
   
#define HAL\_UART1\_Receive\_IT HAL\_UART\_Receive\_IT(&huart1, &rx\_buffer[rx\_index], 1) //������һ�ν���  
extern UART\_HandleTypeDef huart1;  
void usart1\_Init(void); //�������պ���  
void usart1\_process\_data(uint8\_t \*data, uint16\_t length);  
void HAL\_UART1\_RxCpltCallback(UART\_HandleTypeDef \*huart);  
#endif

cmsis\_armcc.h

#ifndef \_\_CMSIS\_ARMCC\_H  
#define \_\_CMSIS\_ARMCC\_H  
#if defined(\_\_ARMCC\_VERSION) && (\_\_ARMCC\_VERSION < 400677)  
 #error "Please use Arm Compiler Toolchain V4.0.677 or later!"  
#endif  
#if ((defined (\_\_TARGET\_ARCH\_6\_M ) && (\_\_TARGET\_ARCH\_6\_M == 1)) || \  
 (defined (\_\_TARGET\_ARCH\_6S\_M ) && (\_\_TARGET\_ARCH\_6S\_M == 1)) )  
 #define \_\_ARM\_ARCH\_6M\_\_ 1  
#endif  
#if (defined (\_\_TARGET\_ARCH\_7\_M ) && (\_\_TARGET\_ARCH\_7\_M == 1))  
 #define \_\_ARM\_ARCH\_7M\_\_ 1  
#endif  
#if (defined (\_\_TARGET\_ARCH\_7E\_M) && (\_\_TARGET\_ARCH\_7E\_M == 1))  
 #define \_\_ARM\_ARCH\_7EM\_\_ 1  
#endif  
#ifndef \_\_ASM  
 #define \_\_ASM \_\_asm  
#endif  
#ifndef \_\_INLINE  
 #define \_\_INLINE \_\_inline  
#endif  
#ifndef \_\_STATIC\_INLINE  
 #define \_\_STATIC\_INLINE static \_\_inline  
#endif  
#ifndef \_\_STATIC\_FORCEINLINE   
 #define \_\_STATIC\_FORCEINLINE static \_\_forceinline  
#endif   
#ifndef \_\_NO\_RETURN  
 #define \_\_NO\_RETURN \_\_declspec(noreturn)  
#endif  
#ifndef \_\_USED  
 #define \_\_USED \_\_attribute\_\_((used))  
#endif  
#ifndef \_\_WEAK  
 #define \_\_WEAK \_\_attribute\_\_((weak))  
#endif  
#ifndef \_\_PACKED  
 #define \_\_PACKED \_\_attribute\_\_((packed))  
#endif  
#ifndef \_\_PACKED\_STRUCT  
 #define \_\_PACKED\_STRUCT \_\_packed struct  
#endif  
#ifndef \_\_PACKED\_UNION  
 #define \_\_PACKED\_UNION \_\_packed union  
#endif  
#ifndef \_\_UNALIGNED\_UINT32 /\* deprecated \*/  
 #define \_\_UNALIGNED\_UINT32(x) (\*((\_\_packed uint32\_t \*)(x)))  
#endif  
#ifndef \_\_UNALIGNED\_UINT16\_WRITE  
 #define \_\_UNALIGNED\_UINT16\_WRITE(addr, val) ((\*((\_\_packed uint16\_t \*)(addr))) = (val))  
#endif  
#ifndef \_\_UNALIGNED\_UINT16\_READ  
 #define \_\_UNALIGNED\_UINT16\_READ(addr) (\*((const \_\_packed uint16\_t \*)(addr)))  
#endif  
#ifndef \_\_UNALIGNED\_UINT32\_WRITE  
 #define \_\_UNALIGNED\_UINT32\_WRITE(addr, val) ((\*((\_\_packed uint32\_t \*)(addr))) = (val))  
#endif  
#ifndef \_\_UNALIGNED\_UINT32\_READ  
 #define \_\_UNALIGNED\_UINT32\_READ(addr) (\*((const \_\_packed uint32\_t \*)(addr)))  
#endif  
#ifndef \_\_ALIGNED  
 #define \_\_ALIGNED(x) \_\_attribute\_\_((aligned(x)))  
#endif  
#ifndef \_\_RESTRICT  
 #define \_\_RESTRICT \_\_restrict  
#endif  
 \defgroup CMSIS\_Core\_RegAccFunctions CMSIS Core Register Access Functions  
 @{  
 \brief Enable IRQ Interrupts  
 \details Enables IRQ interrupts by clearing the I-bit in the CPSR.  
 Can only be executed in Privileged modes.  
 \brief Disable IRQ Interrupts  
 \details Disables IRQ interrupts by setting the I-bit in the CPSR.  
 Can only be executed in Privileged modes.  
 \brief Get Control Register  
 \details Returns the content of the Control Register.  
 \return Control Register value  
\_\_STATIC\_INLINE uint32\_t \_\_get\_CONTROL(void)  
{  
 register uint32\_t \_\_regControl \_\_ASM("control");  
 return(\_\_regControl);  
}  
 \brief Set Control Register  
 \details Writes the given value to the Control Register.  
 \param [in] control Control Register value to set  
\_\_STATIC\_INLINE void \_\_set\_CONTROL(uint32\_t control)  
{  
 register uint32\_t \_\_regControl \_\_ASM("control");  
 \_\_regControl = control;  
}  
 \brief Get IPSR Register  
 \details Returns the content of the IPSR Register.  
 \return IPSR Register value  
\_\_STATIC\_INLINE uint32\_t \_\_get\_IPSR(void)  
{  
 register uint32\_t \_\_regIPSR \_\_ASM("ipsr");  
 return(\_\_regIPSR);  
}  
 \brief Get APSR Register  
 \details Returns the content of the APSR Register.  
 \return APSR Register value  
\_\_STATIC\_INLINE uint32\_t \_\_get\_APSR(void)  
{  
 register uint32\_t \_\_regAPSR \_\_ASM("apsr");  
 return(\_\_regAPSR);  
}  
 \brief Get xPSR Register  
 \details Returns the content of the xPSR Register.  
 \return xPSR Register value  
\_\_STATIC\_INLINE uint32\_t \_\_get\_xPSR(void)  
{  
 register uint32\_t \_\_regXPSR \_\_ASM("xpsr");  
 return(\_\_regXPSR);  
}  
 \brief Get Process Stack Pointer  
 \details Returns the current value of the Process Stack Pointer (PSP).  
 \return PSP Register value  
\_\_STATIC\_INLINE uint32\_t \_\_get\_PSP(void)  
{  
 register uint32\_t \_\_regProcessStackPointer \_\_ASM("psp");  
 return(\_\_regProcessStackPointer);  
}  
 \brief Set Process Stack Pointer  
 \details Assigns the given value to the Process Stack Pointer (PSP).  
 \param [in] topOfProcStack Process Stack Pointer value to set  
\_\_STATIC\_INLINE void \_\_set\_PSP(uint32\_t topOfProcStack)  
{  
 register uint32\_t \_\_regProcessStackPointer \_\_ASM("psp");  
 \_\_regProcessStackPointer = topOfProcStack;  
}  
 \brief Get Main Stack Pointer  
 \details Returns the current value of the Main Stack Pointer (MSP).  
 \return MSP Register value  
\_\_STATIC\_INLINE uint32\_t \_\_get\_MSP(void)  
{  
 register uint32\_t \_\_regMainStackPointer \_\_ASM("msp");  
 return(\_\_regMainStackPointer);  
}  
 \brief Set Main Stack Pointer  
 \details Assigns the given value to the Main Stack Pointer (MSP).  
 \param [in] topOfMainStack Main Stack Pointer value to set  
\_\_STATIC\_INLINE void \_\_set\_MSP(uint32\_t topOfMainStack)  
{  
 register uint32\_t \_\_regMainStackPointer \_\_ASM("msp");  
 \_\_regMainStackPointer = topOfMainStack;  
}  
 \brief Get Priority Mask  
 \details Returns the current state of the priority mask bit from the Priority Mask Register.  
 \return Priority Mask value  
\_\_STATIC\_INLINE uint32\_t \_\_get\_PRIMASK(void)  
{  
 register uint32\_t \_\_regPriMask \_\_ASM("primask");  
 return(\_\_regPriMask);  
}  
 \brief Set Priority Mask  
 \details Assigns the given value to the Priority Mask Register.  
 \param [in] priMask Priority Mask  
\_\_STATIC\_INLINE void \_\_set\_PRIMASK(uint32\_t priMask)  
{  
 register uint32\_t \_\_regPriMask \_\_ASM("primask");  
 \_\_regPriMask = (priMask);  
}  
#if ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) )  
 \brief Enable FIQ  
 \details Enables FIQ interrupts by clearing the F-bit in the CPSR.  
 Can only be executed in Privileged modes.  
#define \_\_enable\_fault\_irq \_\_enable\_fiq  
 \brief Disable FIQ  
 \details Disables FIQ interrupts by setting the F-bit in the CPSR.  
 Can only be executed in Privileged modes.  
#define \_\_disable\_fault\_irq \_\_disable\_fiq  
 \brief Get Base Priority  
 \details Returns the current value of the Base Priority register.  
 \return Base Priority register value  
\_\_STATIC\_INLINE uint32\_t \_\_get\_BASEPRI(void)  
{  
 register uint32\_t \_\_regBasePri \_\_ASM("basepri");  
 return(\_\_regBasePri);  
}  
 \brief Set Base Priority  
 \details Assigns the given value to the Base Priority register.  
 \param [in] basePri Base Priority value to set  
\_\_STATIC\_INLINE void \_\_set\_BASEPRI(uint32\_t basePri)  
{  
 register uint32\_t \_\_regBasePri \_\_ASM("basepri");  
 \_\_regBasePri = (basePri & 0xFFU);  
}  
 \brief Set Base Priority with condition  
 \details Assigns the given value to the Base Priority register only if BASEPRI masking is disabled,  
 or the new value increases the BASEPRI priority level.  
 \param [in] basePri Base Priority value to set  
\_\_STATIC\_INLINE void \_\_set\_BASEPRI\_MAX(uint32\_t basePri)  
{  
 register uint32\_t \_\_regBasePriMax \_\_ASM("basepri\_max");  
 \_\_regBasePriMax = (basePri & 0xFFU);  
}  
 \brief Get Fault Mask  
 \details Returns the current value of the Fault Mask register.  
 \return Fault Mask register value  
\_\_STATIC\_INLINE uint32\_t \_\_get\_FAULTMASK(void)  
{  
 register uint32\_t \_\_regFaultMask \_\_ASM("faultmask");  
 return(\_\_regFaultMask);  
}  
 \brief Set Fault Mask  
 \details Assigns the given value to the Fault Mask register.  
 \param [in] faultMask Fault Mask value to set  
\_\_STATIC\_INLINE void \_\_set\_FAULTMASK(uint32\_t faultMask)  
{  
 register uint32\_t \_\_regFaultMask \_\_ASM("faultmask");  
 \_\_regFaultMask = (faultMask & (uint32\_t)1U);  
}  
#endif /\* ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) ) \*/  
 \brief Get FPSCR  
 \details Returns the current value of the Floating Point Status/Control register.  
 \return Floating Point Status/Control register value  
\_\_STATIC\_INLINE uint32\_t \_\_get\_FPSCR(void)  
{  
#if ((defined (\_\_FPU\_PRESENT) && (\_\_FPU\_PRESENT == 1U)) && \  
 (defined (\_\_FPU\_USED ) && (\_\_FPU\_USED == 1U)) )  
 register uint32\_t \_\_regfpscr \_\_ASM("fpscr");  
 return(\_\_regfpscr);  
#else  
 return(0U);  
#endif  
}  
 \brief Set FPSCR  
 \details Assigns the given value to the Floating Point Status/Control register.  
 \param [in] fpscr Floating Point Status/Control value to set  
\_\_STATIC\_INLINE void \_\_set\_FPSCR(uint32\_t fpscr)  
{  
#if ((defined (\_\_FPU\_PRESENT) && (\_\_FPU\_PRESENT == 1U)) && \  
 (defined (\_\_FPU\_USED ) && (\_\_FPU\_USED == 1U)) )  
 register uint32\_t \_\_regfpscr \_\_ASM("fpscr");  
 \_\_regfpscr = (fpscr);  
#else  
 (void)fpscr;  
#endif  
}  
 Access to dedicated instructions  
 @{  
 \brief No Operation  
 \details No Operation does nothing. This instruction can be used for code alignment purposes.  
#define \_\_NOP \_\_nop  
 \brief Wait For Interrupt  
 \details Wait For Interrupt is a hint instruction that suspends execution until one of a number of events occurs.  
#define \_\_WFI \_\_wfi  
 \brief Wait For Event  
 \details Wait For Event is a hint instruction that permits the processor to enter  
 a low-power state until one of a number of events occurs.  
#define \_\_WFE \_\_wfe  
 \brief Send Event  
 \details Send Event is a hint instruction. It causes an event to be signaled to the CPU.  
#define \_\_SEV \_\_sev  
 \brief Instruction Synchronization Barrier  
 \details Instruction Synchronization Barrier flushes the pipeline in the processor,  
 so that all instructions following the ISB are fetched from cache or memory,  
 after the instruction has been completed.  
#define \_\_ISB() do {\  
 \_\_schedule\_barrier();\  
 \_\_isb(0xF);\  
 \_\_schedule\_barrier();\  
 } while (0U)  
 \brief Data Synchronization Barrier  
 \details Acts as a special kind of Data Memory Barrier.  
 It completes when all explicit memory accesses before this instruction complete.  
#define \_\_DSB() do {\  
 \_\_schedule\_barrier();\  
 \_\_dsb(0xF);\  
 \_\_schedule\_barrier();\  
 } while (0U)  
 \brief Data Memory Barrier  
 \details Ensures the apparent order of the explicit memory operations before  
 and after the instruction, without ensuring their completion.  
#define \_\_DMB() do {\  
 \_\_schedule\_barrier();\  
 \_\_dmb(0xF);\  
 \_\_schedule\_barrier();\  
 } while (0U)  
   
 \brief Reverse byte order (32 bit)  
 \details Reverses the byte order in unsigned integer value. For example, 0x12345678 becomes 0x78563412.  
 \param [in] value Value to reverse  
 \return Reversed value  
#define \_\_REV \_\_rev  
 \brief Reverse byte order (16 bit)  
 \details Reverses the byte order within each halfword of a word. For example, 0x12345678 becomes 0x34127856.  
 \param [in] value Value to reverse  
 \return Reversed value  
#ifndef \_\_NO\_EMBEDDED\_ASM  
\_\_attribute\_\_((section(".rev16\_text"))) \_\_STATIC\_INLINE \_\_ASM uint32\_t \_\_REV16(uint32\_t value)  
{  
 rev16 r0, r0  
 bx lr  
}  
#endif  
 \brief Reverse byte order (16 bit)  
 \details Reverses the byte order in a 16-bit value and returns the signed 16-bit result. For example, 0x0080 becomes 0x8000.  
 \param [in] value Value to reverse  
 \return Reversed value  
#ifndef \_\_NO\_EMBEDDED\_ASM  
\_\_attribute\_\_((section(".revsh\_text"))) \_\_STATIC\_INLINE \_\_ASM int16\_t \_\_REVSH(int16\_t value)  
{  
 revsh r0, r0  
 bx lr  
}  
#endif  
 \brief Rotate Right in unsigned value (32 bit)  
 \details Rotate Right (immediate) provides the value of the contents of a register rotated by a variable number of bits.  
 \param [in] op1 Value to rotate  
 \param [in] op2 Number of Bits to rotate  
 \return Rotated value  
#define \_\_ROR \_\_ror  
 \brief Breakpoint  
 \details Causes the processor to enter Debug state.  
 Debug tools can use this to investigate system state when the instruction at a particular address is reached.  
 \param [in] value is ignored by the processor.  
 If required, a debugger can use it to store additional information about the breakpoint.  
#define \_\_BKPT(value) \_\_breakpoint(value)  
 \brief Reverse bit order of value  
 \details Reverses the bit order of the given value.  
 \param [in] value Value to reverse  
 \return Reversed value  
#if ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) )  
 #define \_\_RBIT \_\_rbit  
#else  
\_\_attribute\_\_((always\_inline)) \_\_STATIC\_INLINE uint32\_t \_\_RBIT(uint32\_t value)  
{  
 uint32\_t result;  
 uint32\_t s = (4U /\*sizeof(v)\*/ \* 8U) - 1U; /\* extra shift needed at end \*/  
 result = value; /\* r will be reversed bits of v; first get LSB of v \*/  
 for (value >>= 1U; value != 0U; value >>= 1U)  
 {  
 result <<= 1U;  
 result |= value & 1U;  
 s--;  
 }  
 result <<= s; /\* shift when v's highest bits are zero \*/  
 return result;  
}  
#endif  
 \brief Count leading zeros  
 \details Counts the number of leading zeros of a data value.  
 \param [in] value Value to count the leading zeros  
 \return number of leading zeros in value  
#define \_\_CLZ \_\_clz  
#if ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) )  
 \brief LDR Exclusive (8 bit)  
 \details Executes a exclusive LDR instruction for 8 bit value.  
 \param [in] ptr Pointer to data  
 \return value of type uint8\_t at (\*ptr)  
#if defined(\_\_ARMCC\_VERSION) && (\_\_ARMCC\_VERSION < 5060020)  
 #define \_\_LDREXB(ptr) ((uint8\_t ) \_\_ldrex(ptr))  
#else  
 #define \_\_LDREXB(ptr) \_Pragma("push") \_Pragma("diag\_suppress 3731") ((uint8\_t ) \_\_ldrex(ptr)) \_Pragma("pop")  
#endif  
 \brief LDR Exclusive (16 bit)  
 \details Executes a exclusive LDR instruction for 16 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint16\_t at (\*ptr)  
#if defined(\_\_ARMCC\_VERSION) && (\_\_ARMCC\_VERSION < 5060020)  
 #define \_\_LDREXH(ptr) ((uint16\_t) \_\_ldrex(ptr))  
#else  
 #define \_\_LDREXH(ptr) \_Pragma("push") \_Pragma("diag\_suppress 3731") ((uint16\_t) \_\_ldrex(ptr)) \_Pragma("pop")  
#endif  
 \brief LDR Exclusive (32 bit)  
 \details Executes a exclusive LDR instruction for 32 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint32\_t at (\*ptr)  
#if defined(\_\_ARMCC\_VERSION) && (\_\_ARMCC\_VERSION < 5060020)  
 #define \_\_LDREXW(ptr) ((uint32\_t ) \_\_ldrex(ptr))  
#else  
 #define \_\_LDREXW(ptr) \_Pragma("push") \_Pragma("diag\_suppress 3731") ((uint32\_t ) \_\_ldrex(ptr)) \_Pragma("pop")  
#endif  
 \brief STR Exclusive (8 bit)  
 \details Executes a exclusive STR instruction for 8 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
 \return 0 Function succeeded  
 \return 1 Function failed  
#if defined(\_\_ARMCC\_VERSION) && (\_\_ARMCC\_VERSION < 5060020)  
 #define \_\_STREXB(value, ptr) \_\_strex(value, ptr)  
#else  
 #define \_\_STREXB(value, ptr) \_Pragma("push") \_Pragma("diag\_suppress 3731") \_\_strex(value, ptr) \_Pragma("pop")  
#endif  
 \brief STR Exclusive (16 bit)  
 \details Executes a exclusive STR instruction for 16 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
 \return 0 Function succeeded  
 \return 1 Function failed  
#if defined(\_\_ARMCC\_VERSION) && (\_\_ARMCC\_VERSION < 5060020)  
 #define \_\_STREXH(value, ptr) \_\_strex(value, ptr)  
#else  
 #define \_\_STREXH(value, ptr) \_Pragma("push") \_Pragma("diag\_suppress 3731") \_\_strex(value, ptr) \_Pragma("pop")  
#endif  
 \brief STR Exclusive (32 bit)  
 \details Executes a exclusive STR instruction for 32 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
 \return 0 Function succeeded  
 \return 1 Function failed  
#if defined(\_\_ARMCC\_VERSION) && (\_\_ARMCC\_VERSION < 5060020)  
 #define \_\_STREXW(value, ptr) \_\_strex(value, ptr)  
#else  
 #define \_\_STREXW(value, ptr) \_Pragma("push") \_Pragma("diag\_suppress 3731") \_\_strex(value, ptr) \_Pragma("pop")  
#endif  
 \brief Remove the exclusive lock  
 \details Removes the exclusive lock which is created by LDREX.  
#define \_\_CLREX \_\_clrex  
 \brief Signed Saturate  
 \details Saturates a signed value.  
 \param [in] value Value to be saturated  
 \param [in] sat Bit position to saturate to (1..32)  
 \return Saturated value  
#define \_\_SSAT \_\_ssat  
 \brief Unsigned Saturate  
 \details Saturates an unsigned value.  
 \param [in] value Value to be saturated  
 \param [in] sat Bit position to saturate to (0..31)  
 \return Saturated value  
#define \_\_USAT \_\_usat  
 \brief Rotate Right with Extend (32 bit)  
 \details Moves each bit of a bitstring right by one bit.  
 The carry input is shifted in at the left end of the bitstring.  
 \param [in] value Value to rotate  
 \return Rotated value  
#ifndef \_\_NO\_EMBEDDED\_ASM  
\_\_attribute\_\_((section(".rrx\_text"))) \_\_STATIC\_INLINE \_\_ASM uint32\_t \_\_RRX(uint32\_t value)  
{  
 rrx r0, r0  
 bx lr  
}  
#endif  
 \brief LDRT Unprivileged (8 bit)  
 \details Executes a Unprivileged LDRT instruction for 8 bit value.  
 \param [in] ptr Pointer to data  
 \return value of type uint8\_t at (\*ptr)  
#define \_\_LDRBT(ptr) ((uint8\_t ) \_\_ldrt(ptr))  
 \brief LDRT Unprivileged (16 bit)  
 \details Executes a Unprivileged LDRT instruction for 16 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint16\_t at (\*ptr)  
#define \_\_LDRHT(ptr) ((uint16\_t) \_\_ldrt(ptr))  
 \brief LDRT Unprivileged (32 bit)  
 \details Executes a Unprivileged LDRT instruction for 32 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint32\_t at (\*ptr)  
#define \_\_LDRT(ptr) ((uint32\_t ) \_\_ldrt(ptr))  
 \brief STRT Unprivileged (8 bit)  
 \details Executes a Unprivileged STRT instruction for 8 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
#define \_\_STRBT(value, ptr) \_\_strt(value, ptr)  
 \brief STRT Unprivileged (16 bit)  
 \details Executes a Unprivileged STRT instruction for 16 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
#define \_\_STRHT(value, ptr) \_\_strt(value, ptr)  
 \brief STRT Unprivileged (32 bit)  
 \details Executes a Unprivileged STRT instruction for 32 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
#define \_\_STRT(value, ptr) \_\_strt(value, ptr)  
#else /\* ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) ) \*/  
 \brief Signed Saturate  
 \details Saturates a signed value.  
 \param [in] value Value to be saturated  
 \param [in] sat Bit position to saturate to (1..32)  
 \return Saturated value  
\_\_attribute\_\_((always\_inline)) \_\_STATIC\_INLINE int32\_t \_\_SSAT(int32\_t val, uint32\_t sat)  
{  
 if ((sat >= 1U) && (sat <= 32U))  
 {  
 const int32\_t max = (int32\_t)((1U << (sat - 1U)) - 1U);  
 const int32\_t min = -1 - max ;  
 if (val > max)  
 {  
 return max;  
 }  
 else if (val < min)  
 {  
 return min;  
 }  
 }  
 return val;  
}  
 \brief Unsigned Saturate  
 \details Saturates an unsigned value.  
 \param [in] value Value to be saturated  
 \param [in] sat Bit position to saturate to (0..31)  
 \return Saturated value  
\_\_attribute\_\_((always\_inline)) \_\_STATIC\_INLINE uint32\_t \_\_USAT(int32\_t val, uint32\_t sat)  
{  
 if (sat <= 31U)  
 {  
 const uint32\_t max = ((1U << sat) - 1U);  
 if (val > (int32\_t)max)  
 {  
 return max;  
 }  
 else if (val < 0)  
 {  
 return 0U;  
 }  
 }  
 return (uint32\_t)val;  
}  
#endif /\* ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) ) \*/  
 Access to dedicated SIMD instructions  
 @{  
#if ((defined (\_\_ARM\_ARCH\_7EM\_\_) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) )  
#define \_\_SADD8 \_\_sadd8  
#define \_\_QADD8 \_\_qadd8  
#define \_\_SHADD8 \_\_shadd8  
#define \_\_UADD8 \_\_uadd8  
#define \_\_UQADD8 \_\_uqadd8  
#define \_\_UHADD8 \_\_uhadd8  
#define \_\_SSUB8 \_\_ssub8  
#define \_\_QSUB8 \_\_qsub8  
#define \_\_SHSUB8 \_\_shsub8  
#define \_\_USUB8 \_\_usub8  
#define \_\_UQSUB8 \_\_uqsub8  
#define \_\_UHSUB8 \_\_uhsub8  
#define \_\_SADD16 \_\_sadd16  
#define \_\_QADD16 \_\_qadd16  
#define \_\_SHADD16 \_\_shadd16  
#define \_\_UADD16 \_\_uadd16  
#define \_\_UQADD16 \_\_uqadd16  
#define \_\_UHADD16 \_\_uhadd16  
#define \_\_SSUB16 \_\_ssub16  
#define \_\_QSUB16 \_\_qsub16  
#define \_\_SHSUB16 \_\_shsub16  
#define \_\_USUB16 \_\_usub16  
#define \_\_UQSUB16 \_\_uqsub16  
#define \_\_UHSUB16 \_\_uhsub16  
#define \_\_SASX \_\_sasx  
#define \_\_QASX \_\_qasx  
#define \_\_SHASX \_\_shasx  
#define \_\_UASX \_\_uasx  
#define \_\_UQASX \_\_uqasx  
#define \_\_UHASX \_\_uhasx  
#define \_\_SSAX \_\_ssax  
#define \_\_QSAX \_\_qsax  
#define \_\_SHSAX \_\_shsax  
#define \_\_USAX \_\_usax  
#define \_\_UQSAX \_\_uqsax  
#define \_\_UHSAX \_\_uhsax  
#define \_\_USAD8 \_\_usad8  
#define \_\_USADA8 \_\_usada8  
#define \_\_SSAT16 \_\_ssat16  
#define \_\_USAT16 \_\_usat16  
#define \_\_UXTB16 \_\_uxtb16  
#define \_\_UXTAB16 \_\_uxtab16  
#define \_\_SXTB16 \_\_sxtb16  
#define \_\_SXTAB16 \_\_sxtab16  
#define \_\_SMUAD \_\_smuad  
#define \_\_SMUADX \_\_smuadx  
#define \_\_SMLAD \_\_smlad  
#define \_\_SMLADX \_\_smladx  
#define \_\_SMLALD \_\_smlald  
#define \_\_SMLALDX \_\_smlaldx  
#define \_\_SMUSD \_\_smusd  
#define \_\_SMUSDX \_\_smusdx  
#define \_\_SMLSD \_\_smlsd  
#define \_\_SMLSDX \_\_smlsdx  
#define \_\_SMLSLD \_\_smlsld  
#define \_\_SMLSLDX \_\_smlsldx  
#define \_\_SEL \_\_sel  
#define \_\_QADD \_\_qadd  
#define \_\_QSUB \_\_qsub  
#define \_\_PKHBT(ARG1,ARG2,ARG3) ( ((((uint32\_t)(ARG1)) ) & 0x0000FFFFUL) | \  
 ((((uint32\_t)(ARG2)) << (ARG3)) & 0xFFFF0000UL) )  
#define \_\_PKHTB(ARG1,ARG2,ARG3) ( ((((uint32\_t)(ARG1)) ) & 0xFFFF0000UL) | \  
 ((((uint32\_t)(ARG2)) >> (ARG3)) & 0x0000FFFFUL) )  
#define \_\_SMMLA(ARG1,ARG2,ARG3) ( (int32\_t)((((int64\_t)(ARG1) \* (ARG2)) + \  
 ((int64\_t)(ARG3) << 32U) ) >> 32U))  
#endif /\* ((defined (\_\_ARM\_ARCH\_7EM\_\_) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) ) \*/  
#endif /\* \_\_CMSIS\_ARMCC\_H \*/

cmsis\_armclang.h

#ifndef \_\_CMSIS\_ARMCLANG\_H  
#define \_\_CMSIS\_ARMCLANG\_H  
#pragma clang system\_header /\* treat file as system include file \*/  
#ifndef \_\_ARM\_COMPAT\_H  
#include <arm\_compat.h> /\* Compatibility header for Arm Compiler 5 intrinsics \*/  
#endif  
#ifndef \_\_ASM  
 #define \_\_ASM \_\_asm  
#endif  
#ifndef \_\_INLINE  
 #define \_\_INLINE \_\_inline  
#endif  
#ifndef \_\_STATIC\_INLINE  
 #define \_\_STATIC\_INLINE static \_\_inline  
#endif  
#ifndef \_\_STATIC\_FORCEINLINE   
 #define \_\_STATIC\_FORCEINLINE \_\_attribute\_\_((always\_inline)) static \_\_inline  
#endif   
#ifndef \_\_NO\_RETURN  
 #define \_\_NO\_RETURN \_\_attribute\_\_((\_\_noreturn\_\_))  
#endif  
#ifndef \_\_USED  
 #define \_\_USED \_\_attribute\_\_((used))  
#endif  
#ifndef \_\_WEAK  
 #define \_\_WEAK \_\_attribute\_\_((weak))  
#endif  
#ifndef \_\_PACKED  
 #define \_\_PACKED \_\_attribute\_\_((packed, aligned(1)))  
#endif  
#ifndef \_\_PACKED\_STRUCT  
 #define \_\_PACKED\_STRUCT struct \_\_attribute\_\_((packed, aligned(1)))  
#endif  
#ifndef \_\_PACKED\_UNION  
 #define \_\_PACKED\_UNION union \_\_attribute\_\_((packed, aligned(1)))  
#endif  
#ifndef \_\_UNALIGNED\_UINT32 /\* deprecated \*/  
 #pragma clang diagnostic push  
 #pragma clang diagnostic ignored "-Wpacked"  
 struct \_\_attribute\_\_((packed)) T\_UINT32 { uint32\_t v; };  
 #pragma clang diagnostic pop  
 #define \_\_UNALIGNED\_UINT32(x) (((struct T\_UINT32 \*)(x))->v)  
#endif  
#ifndef \_\_UNALIGNED\_UINT16\_WRITE  
 #pragma clang diagnostic push  
 #pragma clang diagnostic ignored "-Wpacked"  
 \_\_PACKED\_STRUCT T\_UINT16\_WRITE { uint16\_t v; };  
 #pragma clang diagnostic pop  
 #define \_\_UNALIGNED\_UINT16\_WRITE(addr, val) (void)((((struct T\_UINT16\_WRITE \*)(void \*)(addr))->v) = (val))  
#endif  
#ifndef \_\_UNALIGNED\_UINT16\_READ  
 #pragma clang diagnostic push  
 #pragma clang diagnostic ignored "-Wpacked"  
 \_\_PACKED\_STRUCT T\_UINT16\_READ { uint16\_t v; };  
 #pragma clang diagnostic pop  
 #define \_\_UNALIGNED\_UINT16\_READ(addr) (((const struct T\_UINT16\_READ \*)(const void \*)(addr))->v)  
#endif  
#ifndef \_\_UNALIGNED\_UINT32\_WRITE  
 #pragma clang diagnostic push  
 #pragma clang diagnostic ignored "-Wpacked"  
 \_\_PACKED\_STRUCT T\_UINT32\_WRITE { uint32\_t v; };  
 #pragma clang diagnostic pop  
 #define \_\_UNALIGNED\_UINT32\_WRITE(addr, val) (void)((((struct T\_UINT32\_WRITE \*)(void \*)(addr))->v) = (val))  
#endif  
#ifndef \_\_UNALIGNED\_UINT32\_READ  
 #pragma clang diagnostic push  
 #pragma clang diagnostic ignored "-Wpacked"  
 \_\_PACKED\_STRUCT T\_UINT32\_READ { uint32\_t v; };  
 #pragma clang diagnostic pop  
 #define \_\_UNALIGNED\_UINT32\_READ(addr) (((const struct T\_UINT32\_READ \*)(const void \*)(addr))->v)  
#endif  
#ifndef \_\_ALIGNED  
 #define \_\_ALIGNED(x) \_\_attribute\_\_((aligned(x)))  
#endif  
#ifndef \_\_RESTRICT  
 #define \_\_RESTRICT \_\_restrict  
#endif  
 \defgroup CMSIS\_Core\_RegAccFunctions CMSIS Core Register Access Functions  
 @{  
 \brief Enable IRQ Interrupts  
 \details Enables IRQ interrupts by clearing the I-bit in the CPSR.  
 Can only be executed in Privileged modes.  
 \brief Disable IRQ Interrupts  
 \details Disables IRQ interrupts by setting the I-bit in the CPSR.  
 Can only be executed in Privileged modes.  
 \brief Get Control Register  
 \details Returns the content of the Control Register.  
 \return Control Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_CONTROL(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, control" : "=r" (result) );  
 return(result);  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Get Control Register (non-secure)  
 \details Returns the content of the non-secure Control Register when in secure mode.  
 \return non-secure Control Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_TZ\_get\_CONTROL\_NS(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, control\_ns" : "=r" (result) );  
 return(result);  
}  
#endif  
 \brief Set Control Register  
 \details Writes the given value to the Control Register.  
 \param [in] control Control Register value to set  
\_\_STATIC\_FORCEINLINE void \_\_set\_CONTROL(uint32\_t control)  
{  
 \_\_ASM volatile ("MSR control, %0" : : "r" (control) : "memory");  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Set Control Register (non-secure)  
 \details Writes the given value to the non-secure Control Register when in secure state.  
 \param [in] control Control Register value to set  
\_\_STATIC\_FORCEINLINE void \_\_TZ\_set\_CONTROL\_NS(uint32\_t control)  
{  
 \_\_ASM volatile ("MSR control\_ns, %0" : : "r" (control) : "memory");  
}  
#endif  
 \brief Get IPSR Register  
 \details Returns the content of the IPSR Register.  
 \return IPSR Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_IPSR(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, ipsr" : "=r" (result) );  
 return(result);  
}  
 \brief Get APSR Register  
 \details Returns the content of the APSR Register.  
 \return APSR Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_APSR(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, apsr" : "=r" (result) );  
 return(result);  
}  
 \brief Get xPSR Register  
 \details Returns the content of the xPSR Register.  
 \return xPSR Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_xPSR(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, xpsr" : "=r" (result) );  
 return(result);  
}  
 \brief Get Process Stack Pointer  
 \details Returns the current value of the Process Stack Pointer (PSP).  
 \return PSP Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_PSP(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, psp" : "=r" (result) );  
 return(result);  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Get Process Stack Pointer (non-secure)  
 \details Returns the current value of the non-secure Process Stack Pointer (PSP) when in secure state.  
 \return PSP Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_TZ\_get\_PSP\_NS(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, psp\_ns" : "=r" (result) );  
 return(result);  
}  
#endif  
 \brief Set Process Stack Pointer  
 \details Assigns the given value to the Process Stack Pointer (PSP).  
 \param [in] topOfProcStack Process Stack Pointer value to set  
\_\_STATIC\_FORCEINLINE void \_\_set\_PSP(uint32\_t topOfProcStack)  
{  
 \_\_ASM volatile ("MSR psp, %0" : : "r" (topOfProcStack) : );  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Set Process Stack Pointer (non-secure)  
 \details Assigns the given value to the non-secure Process Stack Pointer (PSP) when in secure state.  
 \param [in] topOfProcStack Process Stack Pointer value to set  
\_\_STATIC\_FORCEINLINE void \_\_TZ\_set\_PSP\_NS(uint32\_t topOfProcStack)  
{  
 \_\_ASM volatile ("MSR psp\_ns, %0" : : "r" (topOfProcStack) : );  
}  
#endif  
 \brief Get Main Stack Pointer  
 \details Returns the current value of the Main Stack Pointer (MSP).  
 \return MSP Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_MSP(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, msp" : "=r" (result) );  
 return(result);  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Get Main Stack Pointer (non-secure)  
 \details Returns the current value of the non-secure Main Stack Pointer (MSP) when in secure state.  
 \return MSP Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_TZ\_get\_MSP\_NS(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, msp\_ns" : "=r" (result) );  
 return(result);  
}  
#endif  
 \brief Set Main Stack Pointer  
 \details Assigns the given value to the Main Stack Pointer (MSP).  
 \param [in] topOfMainStack Main Stack Pointer value to set  
\_\_STATIC\_FORCEINLINE void \_\_set\_MSP(uint32\_t topOfMainStack)  
{  
 \_\_ASM volatile ("MSR msp, %0" : : "r" (topOfMainStack) : );  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Set Main Stack Pointer (non-secure)  
 \details Assigns the given value to the non-secure Main Stack Pointer (MSP) when in secure state.  
 \param [in] topOfMainStack Main Stack Pointer value to set  
\_\_STATIC\_FORCEINLINE void \_\_TZ\_set\_MSP\_NS(uint32\_t topOfMainStack)  
{  
 \_\_ASM volatile ("MSR msp\_ns, %0" : : "r" (topOfMainStack) : );  
}  
#endif  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Get Stack Pointer (non-secure)  
 \details Returns the current value of the non-secure Stack Pointer (SP) when in secure state.  
 \return SP Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_TZ\_get\_SP\_NS(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, sp\_ns" : "=r" (result) );  
 return(result);  
}  
 \brief Set Stack Pointer (non-secure)  
 \details Assigns the given value to the non-secure Stack Pointer (SP) when in secure state.  
 \param [in] topOfStack Stack Pointer value to set  
\_\_STATIC\_FORCEINLINE void \_\_TZ\_set\_SP\_NS(uint32\_t topOfStack)  
{  
 \_\_ASM volatile ("MSR sp\_ns, %0" : : "r" (topOfStack) : );  
}  
#endif  
 \brief Get Priority Mask  
 \details Returns the current state of the priority mask bit from the Priority Mask Register.  
 \return Priority Mask value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_PRIMASK(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, primask" : "=r" (result) );  
 return(result);  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Get Priority Mask (non-secure)  
 \details Returns the current state of the non-secure priority mask bit from the Priority Mask Register when in secure state.  
 \return Priority Mask value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_TZ\_get\_PRIMASK\_NS(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, primask\_ns" : "=r" (result) );  
 return(result);  
}  
#endif  
 \brief Set Priority Mask  
 \details Assigns the given value to the Priority Mask Register.  
 \param [in] priMask Priority Mask  
\_\_STATIC\_FORCEINLINE void \_\_set\_PRIMASK(uint32\_t priMask)  
{  
 \_\_ASM volatile ("MSR primask, %0" : : "r" (priMask) : "memory");  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Set Priority Mask (non-secure)  
 \details Assigns the given value to the non-secure Priority Mask Register when in secure state.  
 \param [in] priMask Priority Mask  
\_\_STATIC\_FORCEINLINE void \_\_TZ\_set\_PRIMASK\_NS(uint32\_t priMask)  
{  
 \_\_ASM volatile ("MSR primask\_ns, %0" : : "r" (priMask) : "memory");  
}  
#endif  
#if ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_ ) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) )  
 \brief Enable FIQ  
 \details Enables FIQ interrupts by clearing the F-bit in the CPSR.  
 Can only be executed in Privileged modes.  
#define \_\_enable\_fault\_irq \_\_enable\_fiq /\* see arm\_compat.h \*/  
 \brief Disable FIQ  
 \details Disables FIQ interrupts by setting the F-bit in the CPSR.  
 Can only be executed in Privileged modes.  
#define \_\_disable\_fault\_irq \_\_disable\_fiq /\* see arm\_compat.h \*/  
 \brief Get Base Priority  
 \details Returns the current value of the Base Priority register.  
 \return Base Priority register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_BASEPRI(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, basepri" : "=r" (result) );  
 return(result);  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Get Base Priority (non-secure)  
 \details Returns the current value of the non-secure Base Priority register when in secure state.  
 \return Base Priority register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_TZ\_get\_BASEPRI\_NS(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, basepri\_ns" : "=r" (result) );  
 return(result);  
}  
#endif  
 \brief Set Base Priority  
 \details Assigns the given value to the Base Priority register.  
 \param [in] basePri Base Priority value to set  
\_\_STATIC\_FORCEINLINE void \_\_set\_BASEPRI(uint32\_t basePri)  
{  
 \_\_ASM volatile ("MSR basepri, %0" : : "r" (basePri) : "memory");  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Set Base Priority (non-secure)  
 \details Assigns the given value to the non-secure Base Priority register when in secure state.  
 \param [in] basePri Base Priority value to set  
\_\_STATIC\_FORCEINLINE void \_\_TZ\_set\_BASEPRI\_NS(uint32\_t basePri)  
{  
 \_\_ASM volatile ("MSR basepri\_ns, %0" : : "r" (basePri) : "memory");  
}  
#endif  
 \brief Set Base Priority with condition  
 \details Assigns the given value to the Base Priority register only if BASEPRI masking is disabled,  
 or the new value increases the BASEPRI priority level.  
 \param [in] basePri Base Priority value to set  
\_\_STATIC\_FORCEINLINE void \_\_set\_BASEPRI\_MAX(uint32\_t basePri)  
{  
 \_\_ASM volatile ("MSR basepri\_max, %0" : : "r" (basePri) : "memory");  
}  
 \brief Get Fault Mask  
 \details Returns the current value of the Fault Mask register.  
 \return Fault Mask register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_FAULTMASK(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, faultmask" : "=r" (result) );  
 return(result);  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Get Fault Mask (non-secure)  
 \details Returns the current value of the non-secure Fault Mask register when in secure state.  
 \return Fault Mask register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_TZ\_get\_FAULTMASK\_NS(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, faultmask\_ns" : "=r" (result) );  
 return(result);  
}  
#endif  
 \brief Set Fault Mask  
 \details Assigns the given value to the Fault Mask register.  
 \param [in] faultMask Fault Mask value to set  
\_\_STATIC\_FORCEINLINE void \_\_set\_FAULTMASK(uint32\_t faultMask)  
{  
 \_\_ASM volatile ("MSR faultmask, %0" : : "r" (faultMask) : "memory");  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Set Fault Mask (non-secure)  
 \details Assigns the given value to the non-secure Fault Mask register when in secure state.  
 \param [in] faultMask Fault Mask value to set  
\_\_STATIC\_FORCEINLINE void \_\_TZ\_set\_FAULTMASK\_NS(uint32\_t faultMask)  
{  
 \_\_ASM volatile ("MSR faultmask\_ns, %0" : : "r" (faultMask) : "memory");  
}  
#endif  
#endif /\* ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_ ) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) ) \*/  
#if ((defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_BASE\_\_ ) && (\_\_ARM\_ARCH\_8M\_BASE\_\_ == 1)) )  
 \brief Get Process Stack Pointer Limit  
 Devices without ARMv8-M Main Extensions (i.e. Cortex-M23) lack the non-secure  
 Stack Pointer Limit register hence zero is returned always in non-secure  
 mode.  
   
 \details Returns the current value of the Process Stack Pointer Limit (PSPLIM).  
 \return PSPLIM Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_PSPLIM(void)  
{  
#if (!(defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) && \  
 (!defined (\_\_ARM\_FEATURE\_CMSE) || (\_\_ARM\_FEATURE\_CMSE < 3)))  
 return 0U;  
#else  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, psplim" : "=r" (result) );  
 return result;  
#endif  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Get Process Stack Pointer Limit (non-secure)  
 Devices without ARMv8-M Main Extensions (i.e. Cortex-M23) lack the non-secure  
 Stack Pointer Limit register hence zero is returned always in non-secure  
 mode.  
 \details Returns the current value of the non-secure Process Stack Pointer Limit (PSPLIM) when in secure state.  
 \return PSPLIM Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_TZ\_get\_PSPLIM\_NS(void)  
{  
#if (!(defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)))  
 return 0U;  
#else  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, psplim\_ns" : "=r" (result) );  
 return result;  
#endif  
}  
#endif  
 \brief Set Process Stack Pointer Limit  
 Devices without ARMv8-M Main Extensions (i.e. Cortex-M23) lack the non-secure  
 Stack Pointer Limit register hence the write is silently ignored in non-secure  
 mode.  
   
 \details Assigns the given value to the Process Stack Pointer Limit (PSPLIM).  
 \param [in] ProcStackPtrLimit Process Stack Pointer Limit value to set  
\_\_STATIC\_FORCEINLINE void \_\_set\_PSPLIM(uint32\_t ProcStackPtrLimit)  
{  
#if (!(defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) && \  
 (!defined (\_\_ARM\_FEATURE\_CMSE) || (\_\_ARM\_FEATURE\_CMSE < 3)))  
 (void)ProcStackPtrLimit;  
#else  
 \_\_ASM volatile ("MSR psplim, %0" : : "r" (ProcStackPtrLimit));  
#endif  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Set Process Stack Pointer (non-secure)  
 Devices without ARMv8-M Main Extensions (i.e. Cortex-M23) lack the non-secure  
 Stack Pointer Limit register hence the write is silently ignored in non-secure  
 mode.  
 \details Assigns the given value to the non-secure Process Stack Pointer Limit (PSPLIM) when in secure state.  
 \param [in] ProcStackPtrLimit Process Stack Pointer Limit value to set  
\_\_STATIC\_FORCEINLINE void \_\_TZ\_set\_PSPLIM\_NS(uint32\_t ProcStackPtrLimit)  
{  
#if (!(defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)))  
 (void)ProcStackPtrLimit;  
#else  
 \_\_ASM volatile ("MSR psplim\_ns, %0\n" : : "r" (ProcStackPtrLimit));  
#endif  
}  
#endif  
 \brief Get Main Stack Pointer Limit  
 Devices without ARMv8-M Main Extensions (i.e. Cortex-M23) lack the non-secure  
 Stack Pointer Limit register hence zero is returned always.  
 \details Returns the current value of the Main Stack Pointer Limit (MSPLIM).  
 \return MSPLIM Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_MSPLIM(void)  
{  
#if (!(defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) && \  
 (!defined (\_\_ARM\_FEATURE\_CMSE) || (\_\_ARM\_FEATURE\_CMSE < 3)))  
 return 0U;  
#else  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, msplim" : "=r" (result) );  
 return result;  
#endif  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Get Main Stack Pointer Limit (non-secure)  
 Devices without ARMv8-M Main Extensions (i.e. Cortex-M23) lack the non-secure  
 Stack Pointer Limit register hence zero is returned always.  
 \details Returns the current value of the non-secure Main Stack Pointer Limit(MSPLIM) when in secure state.  
 \return MSPLIM Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_TZ\_get\_MSPLIM\_NS(void)  
{  
#if (!(defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)))  
 return 0U;  
#else  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, msplim\_ns" : "=r" (result) );  
 return result;  
#endif  
}  
#endif  
 \brief Set Main Stack Pointer Limit  
 Devices without ARMv8-M Main Extensions (i.e. Cortex-M23) lack the non-secure  
 Stack Pointer Limit register hence the write is silently ignored.  
 \details Assigns the given value to the Main Stack Pointer Limit (MSPLIM).  
 \param [in] MainStackPtrLimit Main Stack Pointer Limit value to set  
\_\_STATIC\_FORCEINLINE void \_\_set\_MSPLIM(uint32\_t MainStackPtrLimit)  
{  
#if (!(defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) && \  
 (!defined (\_\_ARM\_FEATURE\_CMSE) || (\_\_ARM\_FEATURE\_CMSE < 3)))  
 (void)MainStackPtrLimit;  
#else  
 \_\_ASM volatile ("MSR msplim, %0" : : "r" (MainStackPtrLimit));  
#endif  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Set Main Stack Pointer Limit (non-secure)  
 Devices without ARMv8-M Main Extensions (i.e. Cortex-M23) lack the non-secure  
 Stack Pointer Limit register hence the write is silently ignored.  
 \details Assigns the given value to the non-secure Main Stack Pointer Limit (MSPLIM) when in secure state.  
 \param [in] MainStackPtrLimit Main Stack Pointer value to set  
\_\_STATIC\_FORCEINLINE void \_\_TZ\_set\_MSPLIM\_NS(uint32\_t MainStackPtrLimit)  
{  
#if (!(defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)))  
 (void)MainStackPtrLimit;  
#else  
 \_\_ASM volatile ("MSR msplim\_ns, %0" : : "r" (MainStackPtrLimit));  
#endif  
}  
#endif  
#endif /\* ((defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_BASE\_\_ ) && (\_\_ARM\_ARCH\_8M\_BASE\_\_ == 1)) ) \*/  
 \brief Get FPSCR  
 \details Returns the current value of the Floating Point Status/Control register.  
 \return Floating Point Status/Control register value  
#if ((defined (\_\_FPU\_PRESENT) && (\_\_FPU\_PRESENT == 1U)) && \  
 (defined (\_\_FPU\_USED ) && (\_\_FPU\_USED == 1U)) )  
#define \_\_get\_FPSCR (uint32\_t)\_\_builtin\_arm\_get\_fpscr  
#else  
#define \_\_get\_FPSCR() ((uint32\_t)0U)  
#endif  
 \brief Set FPSCR  
 \details Assigns the given value to the Floating Point Status/Control register.  
 \param [in] fpscr Floating Point Status/Control value to set  
#if ((defined (\_\_FPU\_PRESENT) && (\_\_FPU\_PRESENT == 1U)) && \  
 (defined (\_\_FPU\_USED ) && (\_\_FPU\_USED == 1U)) )  
#define \_\_set\_FPSCR \_\_builtin\_arm\_set\_fpscr  
#else  
#define \_\_set\_FPSCR(x) ((void)(x))  
#endif  
 Access to dedicated instructions  
 @{  
#if defined (\_\_thumb\_\_) && !defined (\_\_thumb2\_\_)  
#define \_\_CMSIS\_GCC\_OUT\_REG(r) "=l" (r)  
#define \_\_CMSIS\_GCC\_USE\_REG(r) "l" (r)  
#else  
#define \_\_CMSIS\_GCC\_OUT\_REG(r) "=r" (r)  
#define \_\_CMSIS\_GCC\_USE\_REG(r) "r" (r)  
#endif  
 \brief No Operation  
 \details No Operation does nothing. This instruction can be used for code alignment purposes.  
#define \_\_NOP \_\_builtin\_arm\_nop  
 \brief Wait For Interrupt  
 \details Wait For Interrupt is a hint instruction that suspends execution until one of a number of events occurs.  
#define \_\_WFI \_\_builtin\_arm\_wfi  
 \brief Wait For Event  
 \details Wait For Event is a hint instruction that permits the processor to enter  
 a low-power state until one of a number of events occurs.  
#define \_\_WFE \_\_builtin\_arm\_wfe  
 \brief Send Event  
 \details Send Event is a hint instruction. It causes an event to be signaled to the CPU.  
#define \_\_SEV \_\_builtin\_arm\_sev  
 \brief Instruction Synchronization Barrier  
 \details Instruction Synchronization Barrier flushes the pipeline in the processor,  
 so that all instructions following the ISB are fetched from cache or memory,  
 after the instruction has been completed.  
#define \_\_ISB() \_\_builtin\_arm\_isb(0xF);  
 \brief Data Synchronization Barrier  
 \details Acts as a special kind of Data Memory Barrier.  
 It completes when all explicit memory accesses before this instruction complete.  
#define \_\_DSB() \_\_builtin\_arm\_dsb(0xF);  
 \brief Data Memory Barrier  
 \details Ensures the apparent order of the explicit memory operations before  
 and after the instruction, without ensuring their completion.  
#define \_\_DMB() \_\_builtin\_arm\_dmb(0xF);  
 \brief Reverse byte order (32 bit)  
 \details Reverses the byte order in unsigned integer value. For example, 0x12345678 becomes 0x78563412.  
 \param [in] value Value to reverse  
 \return Reversed value  
#define \_\_REV(value) \_\_builtin\_bswap32(value)  
 \brief Reverse byte order (16 bit)  
 \details Reverses the byte order within each halfword of a word. For example, 0x12345678 becomes 0x34127856.  
 \param [in] value Value to reverse  
 \return Reversed value  
#define \_\_REV16(value) \_\_ROR(\_\_REV(value), 16)  
 \brief Reverse byte order (16 bit)  
 \details Reverses the byte order in a 16-bit value and returns the signed 16-bit result. For example, 0x0080 becomes 0x8000.  
 \param [in] value Value to reverse  
 \return Reversed value  
#define \_\_REVSH(value) (int16\_t)\_\_builtin\_bswap16(value)  
 \brief Rotate Right in unsigned value (32 bit)  
 \details Rotate Right (immediate) provides the value of the contents of a register rotated by a variable number of bits.  
 \param [in] op1 Value to rotate  
 \param [in] op2 Number of Bits to rotate  
 \return Rotated value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_ROR(uint32\_t op1, uint32\_t op2)  
{  
 op2 %= 32U;  
 if (op2 == 0U)  
 {  
 return op1;  
 }  
 return (op1 >> op2) | (op1 << (32U - op2));  
}  
 \brief Breakpoint  
 \details Causes the processor to enter Debug state.  
 Debug tools can use this to investigate system state when the instruction at a particular address is reached.  
 \param [in] value is ignored by the processor.  
 If required, a debugger can use it to store additional information about the breakpoint.  
#define \_\_BKPT(value) \_\_ASM volatile ("bkpt "#value)  
 \brief Reverse bit order of value  
 \details Reverses the bit order of the given value.  
 \param [in] value Value to reverse  
 \return Reversed value  
#define \_\_RBIT \_\_builtin\_arm\_rbit  
 \brief Count leading zeros  
 \details Counts the number of leading zeros of a data value.  
 \param [in] value Value to count the leading zeros  
 \return number of leading zeros in value  
#define \_\_CLZ (uint8\_t)\_\_builtin\_clz  
#if ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_ ) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_BASE\_\_ ) && (\_\_ARM\_ARCH\_8M\_BASE\_\_ == 1)) )  
 \brief LDR Exclusive (8 bit)  
 \details Executes a exclusive LDR instruction for 8 bit value.  
 \param [in] ptr Pointer to data  
 \return value of type uint8\_t at (\*ptr)  
#define \_\_LDREXB (uint8\_t)\_\_builtin\_arm\_ldrex  
 \brief LDR Exclusive (16 bit)  
 \details Executes a exclusive LDR instruction for 16 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint16\_t at (\*ptr)  
#define \_\_LDREXH (uint16\_t)\_\_builtin\_arm\_ldrex  
 \brief LDR Exclusive (32 bit)  
 \details Executes a exclusive LDR instruction for 32 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint32\_t at (\*ptr)  
#define \_\_LDREXW (uint32\_t)\_\_builtin\_arm\_ldrex  
 \brief STR Exclusive (8 bit)  
 \details Executes a exclusive STR instruction for 8 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
 \return 0 Function succeeded  
 \return 1 Function failed  
#define \_\_STREXB (uint32\_t)\_\_builtin\_arm\_strex  
 \brief STR Exclusive (16 bit)  
 \details Executes a exclusive STR instruction for 16 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
 \return 0 Function succeeded  
 \return 1 Function failed  
#define \_\_STREXH (uint32\_t)\_\_builtin\_arm\_strex  
 \brief STR Exclusive (32 bit)  
 \details Executes a exclusive STR instruction for 32 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
 \return 0 Function succeeded  
 \return 1 Function failed  
#define \_\_STREXW (uint32\_t)\_\_builtin\_arm\_strex  
 \brief Remove the exclusive lock  
 \details Removes the exclusive lock which is created by LDREX.  
#define \_\_CLREX \_\_builtin\_arm\_clrex  
#endif /\* ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_ ) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_BASE\_\_ ) && (\_\_ARM\_ARCH\_8M\_BASE\_\_ == 1)) ) \*/  
#if ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_ ) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) )  
 \brief Signed Saturate  
 \details Saturates a signed value.  
 \param [in] value Value to be saturated  
 \param [in] sat Bit position to saturate to (1..32)  
 \return Saturated value  
#define \_\_SSAT \_\_builtin\_arm\_ssat  
 \brief Unsigned Saturate  
 \details Saturates an unsigned value.  
 \param [in] value Value to be saturated  
 \param [in] sat Bit position to saturate to (0..31)  
 \return Saturated value  
#define \_\_USAT \_\_builtin\_arm\_usat  
 \brief Rotate Right with Extend (32 bit)  
 \details Moves each bit of a bitstring right by one bit.  
 The carry input is shifted in at the left end of the bitstring.  
 \param [in] value Value to rotate  
 \return Rotated value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_RRX(uint32\_t value)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("rrx %0, %1" : \_\_CMSIS\_GCC\_OUT\_REG (result) : \_\_CMSIS\_GCC\_USE\_REG (value) );  
 return(result);  
}  
 \brief LDRT Unprivileged (8 bit)  
 \details Executes a Unprivileged LDRT instruction for 8 bit value.  
 \param [in] ptr Pointer to data  
 \return value of type uint8\_t at (\*ptr)  
\_\_STATIC\_FORCEINLINE uint8\_t \_\_LDRBT(volatile uint8\_t \*ptr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("ldrbt %0, %1" : "=r" (result) : "Q" (\*ptr) );  
 return ((uint8\_t) result); /\* Add explicit type cast here \*/  
}  
 \brief LDRT Unprivileged (16 bit)  
 \details Executes a Unprivileged LDRT instruction for 16 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint16\_t at (\*ptr)  
\_\_STATIC\_FORCEINLINE uint16\_t \_\_LDRHT(volatile uint16\_t \*ptr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("ldrht %0, %1" : "=r" (result) : "Q" (\*ptr) );  
 return ((uint16\_t) result); /\* Add explicit type cast here \*/  
}  
 \brief LDRT Unprivileged (32 bit)  
 \details Executes a Unprivileged LDRT instruction for 32 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint32\_t at (\*ptr)  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_LDRT(volatile uint32\_t \*ptr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("ldrt %0, %1" : "=r" (result) : "Q" (\*ptr) );  
 return(result);  
}  
 \brief STRT Unprivileged (8 bit)  
 \details Executes a Unprivileged STRT instruction for 8 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
\_\_STATIC\_FORCEINLINE void \_\_STRBT(uint8\_t value, volatile uint8\_t \*ptr)  
{  
 \_\_ASM volatile ("strbt %1, %0" : "=Q" (\*ptr) : "r" ((uint32\_t)value) );  
}  
 \brief STRT Unprivileged (16 bit)  
 \details Executes a Unprivileged STRT instruction for 16 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
\_\_STATIC\_FORCEINLINE void \_\_STRHT(uint16\_t value, volatile uint16\_t \*ptr)  
{  
 \_\_ASM volatile ("strht %1, %0" : "=Q" (\*ptr) : "r" ((uint32\_t)value) );  
}  
 \brief STRT Unprivileged (32 bit)  
 \details Executes a Unprivileged STRT instruction for 32 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
\_\_STATIC\_FORCEINLINE void \_\_STRT(uint32\_t value, volatile uint32\_t \*ptr)  
{  
 \_\_ASM volatile ("strt %1, %0" : "=Q" (\*ptr) : "r" (value) );  
}  
#else /\* ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_ ) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) ) \*/  
 \brief Signed Saturate  
 \details Saturates a signed value.  
 \param [in] value Value to be saturated  
 \param [in] sat Bit position to saturate to (1..32)  
 \return Saturated value  
\_\_STATIC\_FORCEINLINE int32\_t \_\_SSAT(int32\_t val, uint32\_t sat)  
{  
 if ((sat >= 1U) && (sat <= 32U))  
 {  
 const int32\_t max = (int32\_t)((1U << (sat - 1U)) - 1U);  
 const int32\_t min = -1 - max ;  
 if (val > max)  
 {  
 return max;  
 }  
 else if (val < min)  
 {  
 return min;  
 }  
 }  
 return val;  
}  
 \brief Unsigned Saturate  
 \details Saturates an unsigned value.  
 \param [in] value Value to be saturated  
 \param [in] sat Bit position to saturate to (0..31)  
 \return Saturated value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_USAT(int32\_t val, uint32\_t sat)  
{  
 if (sat <= 31U)  
 {  
 const uint32\_t max = ((1U << sat) - 1U);  
 if (val > (int32\_t)max)  
 {  
 return max;  
 }  
 else if (val < 0)  
 {  
 return 0U;  
 }  
 }  
 return (uint32\_t)val;  
}  
#endif /\* ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_ ) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) ) \*/  
#if ((defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_BASE\_\_ ) && (\_\_ARM\_ARCH\_8M\_BASE\_\_ == 1)) )  
 \brief Load-Acquire (8 bit)  
 \details Executes a LDAB instruction for 8 bit value.  
 \param [in] ptr Pointer to data  
 \return value of type uint8\_t at (\*ptr)  
\_\_STATIC\_FORCEINLINE uint8\_t \_\_LDAB(volatile uint8\_t \*ptr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("ldab %0, %1" : "=r" (result) : "Q" (\*ptr) );  
 return ((uint8\_t) result);  
}  
 \brief Load-Acquire (16 bit)  
 \details Executes a LDAH instruction for 16 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint16\_t at (\*ptr)  
\_\_STATIC\_FORCEINLINE uint16\_t \_\_LDAH(volatile uint16\_t \*ptr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("ldah %0, %1" : "=r" (result) : "Q" (\*ptr) );  
 return ((uint16\_t) result);  
}  
 \brief Load-Acquire (32 bit)  
 \details Executes a LDA instruction for 32 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint32\_t at (\*ptr)  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_LDA(volatile uint32\_t \*ptr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("lda %0, %1" : "=r" (result) : "Q" (\*ptr) );  
 return(result);  
}  
 \brief Store-Release (8 bit)  
 \details Executes a STLB instruction for 8 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
\_\_STATIC\_FORCEINLINE void \_\_STLB(uint8\_t value, volatile uint8\_t \*ptr)  
{  
 \_\_ASM volatile ("stlb %1, %0" : "=Q" (\*ptr) : "r" ((uint32\_t)value) );  
}  
 \brief Store-Release (16 bit)  
 \details Executes a STLH instruction for 16 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
\_\_STATIC\_FORCEINLINE void \_\_STLH(uint16\_t value, volatile uint16\_t \*ptr)  
{  
 \_\_ASM volatile ("stlh %1, %0" : "=Q" (\*ptr) : "r" ((uint32\_t)value) );  
}  
 \brief Store-Release (32 bit)  
 \details Executes a STL instruction for 32 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
\_\_STATIC\_FORCEINLINE void \_\_STL(uint32\_t value, volatile uint32\_t \*ptr)  
{  
 \_\_ASM volatile ("stl %1, %0" : "=Q" (\*ptr) : "r" ((uint32\_t)value) );  
}  
 \brief Load-Acquire Exclusive (8 bit)  
 \details Executes a LDAB exclusive instruction for 8 bit value.  
 \param [in] ptr Pointer to data  
 \return value of type uint8\_t at (\*ptr)  
#define \_\_LDAEXB (uint8\_t)\_\_builtin\_arm\_ldaex  
 \brief Load-Acquire Exclusive (16 bit)  
 \details Executes a LDAH exclusive instruction for 16 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint16\_t at (\*ptr)  
#define \_\_LDAEXH (uint16\_t)\_\_builtin\_arm\_ldaex  
 \brief Load-Acquire Exclusive (32 bit)  
 \details Executes a LDA exclusive instruction for 32 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint32\_t at (\*ptr)  
#define \_\_LDAEX (uint32\_t)\_\_builtin\_arm\_ldaex  
 \brief Store-Release Exclusive (8 bit)  
 \details Executes a STLB exclusive instruction for 8 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
 \return 0 Function succeeded  
 \return 1 Function failed  
#define \_\_STLEXB (uint32\_t)\_\_builtin\_arm\_stlex  
 \brief Store-Release Exclusive (16 bit)  
 \details Executes a STLH exclusive instruction for 16 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
 \return 0 Function succeeded  
 \return 1 Function failed  
#define \_\_STLEXH (uint32\_t)\_\_builtin\_arm\_stlex  
 \brief Store-Release Exclusive (32 bit)  
 \details Executes a STL exclusive instruction for 32 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
 \return 0 Function succeeded  
 \return 1 Function failed  
#define \_\_STLEX (uint32\_t)\_\_builtin\_arm\_stlex  
#endif /\* ((defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_BASE\_\_ ) && (\_\_ARM\_ARCH\_8M\_BASE\_\_ == 1)) ) \*/  
 Access to dedicated SIMD instructions  
 @{  
#if (defined (\_\_ARM\_FEATURE\_DSP) && (\_\_ARM\_FEATURE\_DSP == 1))  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SADD8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("sadd8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_QADD8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("qadd8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SHADD8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("shadd8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UADD8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uadd8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UQADD8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uqadd8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UHADD8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uhadd8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SSUB8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("ssub8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_QSUB8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("qsub8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SHSUB8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("shsub8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_USUB8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("usub8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UQSUB8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uqsub8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UHSUB8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uhsub8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SADD16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("sadd16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_QADD16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("qadd16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SHADD16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("shadd16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UADD16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uadd16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UQADD16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uqadd16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UHADD16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uhadd16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SSUB16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("ssub16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_QSUB16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("qsub16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SHSUB16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("shsub16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_USUB16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("usub16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UQSUB16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uqsub16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UHSUB16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uhsub16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SASX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("sasx %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_QASX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("qasx %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SHASX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("shasx %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UASX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uasx %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UQASX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uqasx %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UHASX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uhasx %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SSAX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("ssax %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_QSAX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("qsax %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SHSAX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("shsax %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_USAX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("usax %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UQSAX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uqsax %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UHSAX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uhsax %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_USAD8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("usad8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_USADA8(uint32\_t op1, uint32\_t op2, uint32\_t op3)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("usada8 %0, %1, %2, %3" : "=r" (result) : "r" (op1), "r" (op2), "r" (op3) );  
 return(result);  
}  
#define \_\_SSAT16(ARG1,ARG2) \  
({ \  
 int32\_t \_\_RES, \_\_ARG1 = (ARG1); \  
 \_\_ASM ("ssat16 %0, %1, %2" : "=r" (\_\_RES) : "I" (ARG2), "r" (\_\_ARG1) ); \  
 \_\_RES; \  
 })  
#define \_\_USAT16(ARG1,ARG2) \  
({ \  
 uint32\_t \_\_RES, \_\_ARG1 = (ARG1); \  
 \_\_ASM ("usat16 %0, %1, %2" : "=r" (\_\_RES) : "I" (ARG2), "r" (\_\_ARG1) ); \  
 \_\_RES; \  
 })  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UXTB16(uint32\_t op1)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uxtb16 %0, %1" : "=r" (result) : "r" (op1));  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UXTAB16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uxtab16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SXTB16(uint32\_t op1)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("sxtb16 %0, %1" : "=r" (result) : "r" (op1));  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SXTAB16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("sxtab16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SMUAD (uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("smuad %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SMUADX (uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("smuadx %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SMLAD (uint32\_t op1, uint32\_t op2, uint32\_t op3)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("smlad %0, %1, %2, %3" : "=r" (result) : "r" (op1), "r" (op2), "r" (op3) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SMLADX (uint32\_t op1, uint32\_t op2, uint32\_t op3)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("smladx %0, %1, %2, %3" : "=r" (result) : "r" (op1), "r" (op2), "r" (op3) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint64\_t \_\_SMLALD (uint32\_t op1, uint32\_t op2, uint64\_t acc)  
{  
 union llreg\_u{  
 uint32\_t w32[2];  
 uint64\_t w64;  
 } llr;  
 llr.w64 = acc;  
#ifndef \_\_ARMEB\_\_ /\* Little endian \*/  
 \_\_ASM volatile ("smlald %0, %1, %2, %3" : "=r" (llr.w32[0]), "=r" (llr.w32[1]): "r" (op1), "r" (op2) , "0" (llr.w32[0]), "1" (llr.w32[1]) );  
#else /\* Big endian \*/  
 \_\_ASM volatile ("smlald %0, %1, %2, %3" : "=r" (llr.w32[1]), "=r" (llr.w32[0]): "r" (op1), "r" (op2) , "0" (llr.w32[1]), "1" (llr.w32[0]) );  
#endif  
 return(llr.w64);  
}  
\_\_STATIC\_FORCEINLINE uint64\_t \_\_SMLALDX (uint32\_t op1, uint32\_t op2, uint64\_t acc)  
{  
 union llreg\_u{  
 uint32\_t w32[2];  
 uint64\_t w64;  
 } llr;  
 llr.w64 = acc;  
#ifndef \_\_ARMEB\_\_ /\* Little endian \*/  
 \_\_ASM volatile ("smlaldx %0, %1, %2, %3" : "=r" (llr.w32[0]), "=r" (llr.w32[1]): "r" (op1), "r" (op2) , "0" (llr.w32[0]), "1" (llr.w32[1]) );  
#else /\* Big endian \*/  
 \_\_ASM volatile ("smlaldx %0, %1, %2, %3" : "=r" (llr.w32[1]), "=r" (llr.w32[0]): "r" (op1), "r" (op2) , "0" (llr.w32[1]), "1" (llr.w32[0]) );  
#endif  
 return(llr.w64);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SMUSD (uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("smusd %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SMUSDX (uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("smusdx %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SMLSD (uint32\_t op1, uint32\_t op2, uint32\_t op3)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("smlsd %0, %1, %2, %3" : "=r" (result) : "r" (op1), "r" (op2), "r" (op3) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SMLSDX (uint32\_t op1, uint32\_t op2, uint32\_t op3)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("smlsdx %0, %1, %2, %3" : "=r" (result) : "r" (op1), "r" (op2), "r" (op3) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint64\_t \_\_SMLSLD (uint32\_t op1, uint32\_t op2, uint64\_t acc)  
{  
 union llreg\_u{  
 uint32\_t w32[2];  
 uint64\_t w64;  
 } llr;  
 llr.w64 = acc;  
#ifndef \_\_ARMEB\_\_ /\* Little endian \*/  
 \_\_ASM volatile ("smlsld %0, %1, %2, %3" : "=r" (llr.w32[0]), "=r" (llr.w32[1]): "r" (op1), "r" (op2) , "0" (llr.w32[0]), "1" (llr.w32[1]) );  
#else /\* Big endian \*/  
 \_\_ASM volatile ("smlsld %0, %1, %2, %3" : "=r" (llr.w32[1]), "=r" (llr.w32[0]): "r" (op1), "r" (op2) , "0" (llr.w32[1]), "1" (llr.w32[0]) );  
#endif  
 return(llr.w64);  
}  
\_\_STATIC\_FORCEINLINE uint64\_t \_\_SMLSLDX (uint32\_t op1, uint32\_t op2, uint64\_t acc)  
{  
 union llreg\_u{  
 uint32\_t w32[2];  
 uint64\_t w64;  
 } llr;  
 llr.w64 = acc;  
#ifndef \_\_ARMEB\_\_ /\* Little endian \*/  
 \_\_ASM volatile ("smlsldx %0, %1, %2, %3" : "=r" (llr.w32[0]), "=r" (llr.w32[1]): "r" (op1), "r" (op2) , "0" (llr.w32[0]), "1" (llr.w32[1]) );  
#else /\* Big endian \*/  
 \_\_ASM volatile ("smlsldx %0, %1, %2, %3" : "=r" (llr.w32[1]), "=r" (llr.w32[0]): "r" (op1), "r" (op2) , "0" (llr.w32[1]), "1" (llr.w32[0]) );  
#endif  
 return(llr.w64);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SEL (uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("sel %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE int32\_t \_\_QADD( int32\_t op1, int32\_t op2)  
{  
 int32\_t result;  
 \_\_ASM volatile ("qadd %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE int32\_t \_\_QSUB( int32\_t op1, int32\_t op2)  
{  
 int32\_t result;  
 \_\_ASM volatile ("qsub %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
#if 0  
#define \_\_PKHBT(ARG1,ARG2,ARG3) \  
({ \  
 uint32\_t \_\_RES, \_\_ARG1 = (ARG1), \_\_ARG2 = (ARG2); \  
 \_\_ASM ("pkhbt %0, %1, %2, lsl %3" : "=r" (\_\_RES) : "r" (\_\_ARG1), "r" (\_\_ARG2), "I" (ARG3) ); \  
 \_\_RES; \  
 })  
#define \_\_PKHTB(ARG1,ARG2,ARG3) \  
({ \  
 uint32\_t \_\_RES, \_\_ARG1 = (ARG1), \_\_ARG2 = (ARG2); \  
 if (ARG3 == 0) \  
 \_\_ASM ("pkhtb %0, %1, %2" : "=r" (\_\_RES) : "r" (\_\_ARG1), "r" (\_\_ARG2) ); \  
 else \  
 \_\_ASM ("pkhtb %0, %1, %2, asr %3" : "=r" (\_\_RES) : "r" (\_\_ARG1), "r" (\_\_ARG2), "I" (ARG3) ); \  
 \_\_RES; \  
 })  
#endif  
#define \_\_PKHBT(ARG1,ARG2,ARG3) ( ((((uint32\_t)(ARG1)) ) & 0x0000FFFFUL) | \  
 ((((uint32\_t)(ARG2)) << (ARG3)) & 0xFFFF0000UL) )  
#define \_\_PKHTB(ARG1,ARG2,ARG3) ( ((((uint32\_t)(ARG1)) ) & 0xFFFF0000UL) | \  
 ((((uint32\_t)(ARG2)) >> (ARG3)) & 0x0000FFFFUL) )  
\_\_STATIC\_FORCEINLINE int32\_t \_\_SMMLA (int32\_t op1, int32\_t op2, int32\_t op3)  
{  
 int32\_t result;  
 \_\_ASM volatile ("smmla %0, %1, %2, %3" : "=r" (result): "r" (op1), "r" (op2), "r" (op3) );  
 return(result);  
}  
#endif /\* (\_\_ARM\_FEATURE\_DSP == 1) \*/  
#endif /\* \_\_CMSIS\_ARMCLANG\_H \*/

cmsis\_compiler.h

#ifndef \_\_CMSIS\_COMPILER\_H  
#define \_\_CMSIS\_COMPILER\_H  
#include <stdint.h>  
#if defined ( \_\_CC\_ARM )  
 #include "cmsis\_armcc.h"  
#elif defined (\_\_ARMCC\_VERSION) && (\_\_ARMCC\_VERSION >= 6010050)  
 #include "cmsis\_armclang.h"  
#elif defined ( \_\_GNUC\_\_ )  
 #include "cmsis\_gcc.h"  
#elif defined ( \_\_ICCARM\_\_ )  
 #include <cmsis\_iccarm.h>  
#elif defined ( \_\_TI\_ARM\_\_ )  
 #include <cmsis\_ccs.h>  
 #ifndef \_\_ASM  
 #define \_\_ASM \_\_asm  
 #endif  
 #ifndef \_\_INLINE  
 #define \_\_INLINE inline  
 #endif  
 #ifndef \_\_STATIC\_INLINE  
 #define \_\_STATIC\_INLINE static inline  
 #endif  
 #ifndef \_\_STATIC\_FORCEINLINE  
 #define \_\_STATIC\_FORCEINLINE \_\_STATIC\_INLINE  
 #endif  
 #ifndef \_\_NO\_RETURN  
 #define \_\_NO\_RETURN \_\_attribute\_\_((noreturn))  
 #endif  
 #ifndef \_\_USED  
 #define \_\_USED \_\_attribute\_\_((used))  
 #endif  
 #ifndef \_\_WEAK  
 #define \_\_WEAK \_\_attribute\_\_((weak))  
 #endif  
 #ifndef \_\_PACKED  
 #define \_\_PACKED \_\_attribute\_\_((packed))  
 #endif  
 #ifndef \_\_PACKED\_STRUCT  
 #define \_\_PACKED\_STRUCT struct \_\_attribute\_\_((packed))  
 #endif  
 #ifndef \_\_PACKED\_UNION  
 #define \_\_PACKED\_UNION union \_\_attribute\_\_((packed))  
 #endif  
 #ifndef \_\_UNALIGNED\_UINT32 /\* deprecated \*/  
 struct \_\_attribute\_\_((packed)) T\_UINT32 { uint32\_t v; };  
 #define \_\_UNALIGNED\_UINT32(x) (((struct T\_UINT32 \*)(x))->v)  
 #endif  
 #ifndef \_\_UNALIGNED\_UINT16\_WRITE  
 \_\_PACKED\_STRUCT T\_UINT16\_WRITE { uint16\_t v; };  
 #define \_\_UNALIGNED\_UINT16\_WRITE(addr, val) (void)((((struct T\_UINT16\_WRITE \*)(void\*)(addr))->v) = (val))  
 #endif  
 #ifndef \_\_UNALIGNED\_UINT16\_READ  
 \_\_PACKED\_STRUCT T\_UINT16\_READ { uint16\_t v; };  
 #define \_\_UNALIGNED\_UINT16\_READ(addr) (((const struct T\_UINT16\_READ \*)(const void \*)(addr))->v)  
 #endif  
 #ifndef \_\_UNALIGNED\_UINT32\_WRITE  
 \_\_PACKED\_STRUCT T\_UINT32\_WRITE { uint32\_t v; };  
 #define \_\_UNALIGNED\_UINT32\_WRITE(addr, val) (void)((((struct T\_UINT32\_WRITE \*)(void \*)(addr))->v) = (val))  
 #endif  
 #ifndef \_\_UNALIGNED\_UINT32\_READ  
 \_\_PACKED\_STRUCT T\_UINT32\_READ { uint32\_t v; };  
 #define \_\_UNALIGNED\_UINT32\_READ(addr) (((const struct T\_UINT32\_READ \*)(const void \*)(addr))->v)  
 #endif  
 #ifndef \_\_ALIGNED  
 #define \_\_ALIGNED(x) \_\_attribute\_\_((aligned(x)))  
 #endif  
 #ifndef \_\_RESTRICT  
 #warning No compiler specific solution for \_\_RESTRICT. \_\_RESTRICT is ignored.  
 #define \_\_RESTRICT  
 #endif  
#elif defined ( \_\_TASKING\_\_ )  
 #ifndef \_\_ASM  
 #define \_\_ASM \_\_asm  
 #endif  
 #ifndef \_\_INLINE  
 #define \_\_INLINE inline  
 #endif  
 #ifndef \_\_STATIC\_INLINE  
 #define \_\_STATIC\_INLINE static inline  
 #endif  
 #ifndef \_\_STATIC\_FORCEINLINE  
 #define \_\_STATIC\_FORCEINLINE \_\_STATIC\_INLINE  
 #endif  
 #ifndef \_\_NO\_RETURN  
 #define \_\_NO\_RETURN \_\_attribute\_\_((noreturn))  
 #endif  
 #ifndef \_\_USED  
 #define \_\_USED \_\_attribute\_\_((used))  
 #endif  
 #ifndef \_\_WEAK  
 #define \_\_WEAK \_\_attribute\_\_((weak))  
 #endif  
 #ifndef \_\_PACKED  
 #define \_\_PACKED \_\_packed\_\_  
 #endif  
 #ifndef \_\_PACKED\_STRUCT  
 #define \_\_PACKED\_STRUCT struct \_\_packed\_\_  
 #endif  
 #ifndef \_\_PACKED\_UNION  
 #define \_\_PACKED\_UNION union \_\_packed\_\_  
 #endif  
 #ifndef \_\_UNALIGNED\_UINT32 /\* deprecated \*/  
 struct \_\_packed\_\_ T\_UINT32 { uint32\_t v; };  
 #define \_\_UNALIGNED\_UINT32(x) (((struct T\_UINT32 \*)(x))->v)  
 #endif  
 #ifndef \_\_UNALIGNED\_UINT16\_WRITE  
 \_\_PACKED\_STRUCT T\_UINT16\_WRITE { uint16\_t v; };  
 #define \_\_UNALIGNED\_UINT16\_WRITE(addr, val) (void)((((struct T\_UINT16\_WRITE \*)(void \*)(addr))->v) = (val))  
 #endif  
 #ifndef \_\_UNALIGNED\_UINT16\_READ  
 \_\_PACKED\_STRUCT T\_UINT16\_READ { uint16\_t v; };  
 #define \_\_UNALIGNED\_UINT16\_READ(addr) (((const struct T\_UINT16\_READ \*)(const void \*)(addr))->v)  
 #endif  
 #ifndef \_\_UNALIGNED\_UINT32\_WRITE  
 \_\_PACKED\_STRUCT T\_UINT32\_WRITE { uint32\_t v; };  
 #define \_\_UNALIGNED\_UINT32\_WRITE(addr, val) (void)((((struct T\_UINT32\_WRITE \*)(void \*)(addr))->v) = (val))  
 #endif  
 #ifndef \_\_UNALIGNED\_UINT32\_READ  
 \_\_PACKED\_STRUCT T\_UINT32\_READ { uint32\_t v; };  
 #define \_\_UNALIGNED\_UINT32\_READ(addr) (((const struct T\_UINT32\_READ \*)(const void \*)(addr))->v)  
 #endif  
 #ifndef \_\_ALIGNED  
 #define \_\_ALIGNED(x) \_\_align(x)  
 #endif  
 #ifndef \_\_RESTRICT  
 #warning No compiler specific solution for \_\_RESTRICT. \_\_RESTRICT is ignored.  
 #define \_\_RESTRICT  
 #endif  
#elif defined ( \_\_CSMC\_\_ )  
 #include <cmsis\_csm.h>  
 #ifndef \_\_ASM  
 #define \_\_ASM \_asm  
 #endif  
 #ifndef \_\_INLINE  
 #define \_\_INLINE inline  
 #endif  
 #ifndef \_\_STATIC\_INLINE  
 #define \_\_STATIC\_INLINE static inline  
 #endif  
 #ifndef \_\_STATIC\_FORCEINLINE  
 #define \_\_STATIC\_FORCEINLINE \_\_STATIC\_INLINE  
 #endif  
 #ifndef \_\_NO\_RETURN  
 #define \_\_NO\_RETURN  
 #endif  
 #ifndef \_\_USED  
 #warning No compiler specific solution for \_\_USED. \_\_USED is ignored.  
 #define \_\_USED  
 #endif  
 #ifndef \_\_WEAK  
 #define \_\_WEAK \_\_weak  
 #endif  
 #ifndef \_\_PACKED  
 #define \_\_PACKED @packed  
 #endif  
 #ifndef \_\_PACKED\_STRUCT  
 #define \_\_PACKED\_STRUCT @packed struct  
 #endif  
 #ifndef \_\_PACKED\_UNION  
 #define \_\_PACKED\_UNION @packed union  
 #endif  
 #ifndef \_\_UNALIGNED\_UINT32 /\* deprecated \*/  
 @packed struct T\_UINT32 { uint32\_t v; };  
 #define \_\_UNALIGNED\_UINT32(x) (((struct T\_UINT32 \*)(x))->v)  
 #endif  
 #ifndef \_\_UNALIGNED\_UINT16\_WRITE  
 \_\_PACKED\_STRUCT T\_UINT16\_WRITE { uint16\_t v; };  
 #define \_\_UNALIGNED\_UINT16\_WRITE(addr, val) (void)((((struct T\_UINT16\_WRITE \*)(void \*)(addr))->v) = (val))  
 #endif  
 #ifndef \_\_UNALIGNED\_UINT16\_READ  
 \_\_PACKED\_STRUCT T\_UINT16\_READ { uint16\_t v; };  
 #define \_\_UNALIGNED\_UINT16\_READ(addr) (((const struct T\_UINT16\_READ \*)(const void \*)(addr))->v)  
 #endif  
 #ifndef \_\_UNALIGNED\_UINT32\_WRITE  
 \_\_PACKED\_STRUCT T\_UINT32\_WRITE { uint32\_t v; };  
 #define \_\_UNALIGNED\_UINT32\_WRITE(addr, val) (void)((((struct T\_UINT32\_WRITE \*)(void \*)(addr))->v) = (val))  
 #endif  
 #ifndef \_\_UNALIGNED\_UINT32\_READ  
 \_\_PACKED\_STRUCT T\_UINT32\_READ { uint32\_t v; };  
 #define \_\_UNALIGNED\_UINT32\_READ(addr) (((const struct T\_UINT32\_READ \*)(const void \*)(addr))->v)  
 #endif  
 #ifndef \_\_ALIGNED  
 #warning No compiler specific solution for \_\_ALIGNED. \_\_ALIGNED is ignored.  
 #define \_\_ALIGNED(x)  
 #endif  
 #ifndef \_\_RESTRICT  
 #warning No compiler specific solution for \_\_RESTRICT. \_\_RESTRICT is ignored.  
 #define \_\_RESTRICT  
 #endif  
#else  
 #error Unknown compiler.  
#endif  
#endif /\* \_\_CMSIS\_COMPILER\_H \*/

cmsis\_gcc.h

#ifndef \_\_CMSIS\_GCC\_H  
#define \_\_CMSIS\_GCC\_H  
#pragma GCC diagnostic push  
#pragma GCC diagnostic ignored "-Wsign-conversion"  
#pragma GCC diagnostic ignored "-Wconversion"  
#pragma GCC diagnostic ignored "-Wunused-parameter"  
#ifndef \_\_has\_builtin  
 #define \_\_has\_builtin(x) (0)  
#endif  
#ifndef \_\_ASM  
 #define \_\_ASM \_\_asm  
#endif  
#ifndef \_\_INLINE  
 #define \_\_INLINE inline  
#endif  
#ifndef \_\_STATIC\_INLINE  
 #define \_\_STATIC\_INLINE static inline  
#endif  
#ifndef \_\_STATIC\_FORCEINLINE   
 #define \_\_STATIC\_FORCEINLINE \_\_attribute\_\_((always\_inline)) static inline  
#endif   
#ifndef \_\_NO\_RETURN  
 #define \_\_NO\_RETURN \_\_attribute\_\_((\_\_noreturn\_\_))  
#endif  
#ifndef \_\_USED  
 #define \_\_USED \_\_attribute\_\_((used))  
#endif  
#ifndef \_\_WEAK  
 #define \_\_WEAK \_\_attribute\_\_((weak))  
#endif  
#ifndef \_\_PACKED  
 #define \_\_PACKED \_\_attribute\_\_((packed, aligned(1)))  
#endif  
#ifndef \_\_PACKED\_STRUCT  
 #define \_\_PACKED\_STRUCT struct \_\_attribute\_\_((packed, aligned(1)))  
#endif  
#ifndef \_\_PACKED\_UNION  
 #define \_\_PACKED\_UNION union \_\_attribute\_\_((packed, aligned(1)))  
#endif  
#ifndef \_\_UNALIGNED\_UINT32 /\* deprecated \*/  
 #pragma GCC diagnostic push  
 #pragma GCC diagnostic ignored "-Wpacked"  
 #pragma GCC diagnostic ignored "-Wattributes"  
 struct \_\_attribute\_\_((packed)) T\_UINT32 { uint32\_t v; };  
 #pragma GCC diagnostic pop  
 #define \_\_UNALIGNED\_UINT32(x) (((struct T\_UINT32 \*)(x))->v)  
#endif  
#ifndef \_\_UNALIGNED\_UINT16\_WRITE  
 #pragma GCC diagnostic push  
 #pragma GCC diagnostic ignored "-Wpacked"  
 #pragma GCC diagnostic ignored "-Wattributes"  
 \_\_PACKED\_STRUCT T\_UINT16\_WRITE { uint16\_t v; };  
 #pragma GCC diagnostic pop  
 #define \_\_UNALIGNED\_UINT16\_WRITE(addr, val) (void)((((struct T\_UINT16\_WRITE \*)(void \*)(addr))->v) = (val))  
#endif  
#ifndef \_\_UNALIGNED\_UINT16\_READ  
 #pragma GCC diagnostic push  
 #pragma GCC diagnostic ignored "-Wpacked"  
 #pragma GCC diagnostic ignored "-Wattributes"  
 \_\_PACKED\_STRUCT T\_UINT16\_READ { uint16\_t v; };  
 #pragma GCC diagnostic pop  
 #define \_\_UNALIGNED\_UINT16\_READ(addr) (((const struct T\_UINT16\_READ \*)(const void \*)(addr))->v)  
#endif  
#ifndef \_\_UNALIGNED\_UINT32\_WRITE  
 #pragma GCC diagnostic push  
 #pragma GCC diagnostic ignored "-Wpacked"  
 #pragma GCC diagnostic ignored "-Wattributes"  
 \_\_PACKED\_STRUCT T\_UINT32\_WRITE { uint32\_t v; };  
 #pragma GCC diagnostic pop  
 #define \_\_UNALIGNED\_UINT32\_WRITE(addr, val) (void)((((struct T\_UINT32\_WRITE \*)(void \*)(addr))->v) = (val))  
#endif  
#ifndef \_\_UNALIGNED\_UINT32\_READ  
 #pragma GCC diagnostic push  
 #pragma GCC diagnostic ignored "-Wpacked"  
 #pragma GCC diagnostic ignored "-Wattributes"  
 \_\_PACKED\_STRUCT T\_UINT32\_READ { uint32\_t v; };  
 #pragma GCC diagnostic pop  
 #define \_\_UNALIGNED\_UINT32\_READ(addr) (((const struct T\_UINT32\_READ \*)(const void \*)(addr))->v)  
#endif  
#ifndef \_\_ALIGNED  
 #define \_\_ALIGNED(x) \_\_attribute\_\_((aligned(x)))  
#endif  
#ifndef \_\_RESTRICT  
 #define \_\_RESTRICT \_\_restrict  
#endif  
 \defgroup CMSIS\_Core\_RegAccFunctions CMSIS Core Register Access Functions  
 @{  
 \brief Enable IRQ Interrupts  
 \details Enables IRQ interrupts by clearing the I-bit in the CPSR.  
 Can only be executed in Privileged modes.  
\_\_STATIC\_FORCEINLINE void \_\_enable\_irq(void)  
{  
 \_\_ASM volatile ("cpsie i" : : : "memory");  
}  
 \brief Disable IRQ Interrupts  
 \details Disables IRQ interrupts by setting the I-bit in the CPSR.  
 Can only be executed in Privileged modes.  
\_\_STATIC\_FORCEINLINE void \_\_disable\_irq(void)  
{  
 \_\_ASM volatile ("cpsid i" : : : "memory");  
}  
 \brief Get Control Register  
 \details Returns the content of the Control Register.  
 \return Control Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_CONTROL(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, control" : "=r" (result) );  
 return(result);  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Get Control Register (non-secure)  
 \details Returns the content of the non-secure Control Register when in secure mode.  
 \return non-secure Control Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_TZ\_get\_CONTROL\_NS(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, control\_ns" : "=r" (result) );  
 return(result);  
}  
#endif  
 \brief Set Control Register  
 \details Writes the given value to the Control Register.  
 \param [in] control Control Register value to set  
\_\_STATIC\_FORCEINLINE void \_\_set\_CONTROL(uint32\_t control)  
{  
 \_\_ASM volatile ("MSR control, %0" : : "r" (control) : "memory");  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Set Control Register (non-secure)  
 \details Writes the given value to the non-secure Control Register when in secure state.  
 \param [in] control Control Register value to set  
\_\_STATIC\_FORCEINLINE void \_\_TZ\_set\_CONTROL\_NS(uint32\_t control)  
{  
 \_\_ASM volatile ("MSR control\_ns, %0" : : "r" (control) : "memory");  
}  
#endif  
 \brief Get IPSR Register  
 \details Returns the content of the IPSR Register.  
 \return IPSR Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_IPSR(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, ipsr" : "=r" (result) );  
 return(result);  
}  
 \brief Get APSR Register  
 \details Returns the content of the APSR Register.  
 \return APSR Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_APSR(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, apsr" : "=r" (result) );  
 return(result);  
}  
 \brief Get xPSR Register  
 \details Returns the content of the xPSR Register.  
 \return xPSR Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_xPSR(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, xpsr" : "=r" (result) );  
 return(result);  
}  
 \brief Get Process Stack Pointer  
 \details Returns the current value of the Process Stack Pointer (PSP).  
 \return PSP Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_PSP(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, psp" : "=r" (result) );  
 return(result);  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Get Process Stack Pointer (non-secure)  
 \details Returns the current value of the non-secure Process Stack Pointer (PSP) when in secure state.  
 \return PSP Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_TZ\_get\_PSP\_NS(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, psp\_ns" : "=r" (result) );  
 return(result);  
}  
#endif  
 \brief Set Process Stack Pointer  
 \details Assigns the given value to the Process Stack Pointer (PSP).  
 \param [in] topOfProcStack Process Stack Pointer value to set  
\_\_STATIC\_FORCEINLINE void \_\_set\_PSP(uint32\_t topOfProcStack)  
{  
 \_\_ASM volatile ("MSR psp, %0" : : "r" (topOfProcStack) : );  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Set Process Stack Pointer (non-secure)  
 \details Assigns the given value to the non-secure Process Stack Pointer (PSP) when in secure state.  
 \param [in] topOfProcStack Process Stack Pointer value to set  
\_\_STATIC\_FORCEINLINE void \_\_TZ\_set\_PSP\_NS(uint32\_t topOfProcStack)  
{  
 \_\_ASM volatile ("MSR psp\_ns, %0" : : "r" (topOfProcStack) : );  
}  
#endif  
 \brief Get Main Stack Pointer  
 \details Returns the current value of the Main Stack Pointer (MSP).  
 \return MSP Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_MSP(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, msp" : "=r" (result) );  
 return(result);  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Get Main Stack Pointer (non-secure)  
 \details Returns the current value of the non-secure Main Stack Pointer (MSP) when in secure state.  
 \return MSP Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_TZ\_get\_MSP\_NS(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, msp\_ns" : "=r" (result) );  
 return(result);  
}  
#endif  
 \brief Set Main Stack Pointer  
 \details Assigns the given value to the Main Stack Pointer (MSP).  
 \param [in] topOfMainStack Main Stack Pointer value to set  
\_\_STATIC\_FORCEINLINE void \_\_set\_MSP(uint32\_t topOfMainStack)  
{  
 \_\_ASM volatile ("MSR msp, %0" : : "r" (topOfMainStack) : );  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Set Main Stack Pointer (non-secure)  
 \details Assigns the given value to the non-secure Main Stack Pointer (MSP) when in secure state.  
 \param [in] topOfMainStack Main Stack Pointer value to set  
\_\_STATIC\_FORCEINLINE void \_\_TZ\_set\_MSP\_NS(uint32\_t topOfMainStack)  
{  
 \_\_ASM volatile ("MSR msp\_ns, %0" : : "r" (topOfMainStack) : );  
}  
#endif  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Get Stack Pointer (non-secure)  
 \details Returns the current value of the non-secure Stack Pointer (SP) when in secure state.  
 \return SP Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_TZ\_get\_SP\_NS(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, sp\_ns" : "=r" (result) );  
 return(result);  
}  
 \brief Set Stack Pointer (non-secure)  
 \details Assigns the given value to the non-secure Stack Pointer (SP) when in secure state.  
 \param [in] topOfStack Stack Pointer value to set  
\_\_STATIC\_FORCEINLINE void \_\_TZ\_set\_SP\_NS(uint32\_t topOfStack)  
{  
 \_\_ASM volatile ("MSR sp\_ns, %0" : : "r" (topOfStack) : );  
}  
#endif  
 \brief Get Priority Mask  
 \details Returns the current state of the priority mask bit from the Priority Mask Register.  
 \return Priority Mask value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_PRIMASK(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, primask" : "=r" (result) :: "memory");  
 return(result);  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Get Priority Mask (non-secure)  
 \details Returns the current state of the non-secure priority mask bit from the Priority Mask Register when in secure state.  
 \return Priority Mask value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_TZ\_get\_PRIMASK\_NS(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, primask\_ns" : "=r" (result) :: "memory");  
 return(result);  
}  
#endif  
 \brief Set Priority Mask  
 \details Assigns the given value to the Priority Mask Register.  
 \param [in] priMask Priority Mask  
\_\_STATIC\_FORCEINLINE void \_\_set\_PRIMASK(uint32\_t priMask)  
{  
 \_\_ASM volatile ("MSR primask, %0" : : "r" (priMask) : "memory");  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Set Priority Mask (non-secure)  
 \details Assigns the given value to the non-secure Priority Mask Register when in secure state.  
 \param [in] priMask Priority Mask  
\_\_STATIC\_FORCEINLINE void \_\_TZ\_set\_PRIMASK\_NS(uint32\_t priMask)  
{  
 \_\_ASM volatile ("MSR primask\_ns, %0" : : "r" (priMask) : "memory");  
}  
#endif  
#if ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_ ) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) )  
 \brief Enable FIQ  
 \details Enables FIQ interrupts by clearing the F-bit in the CPSR.  
 Can only be executed in Privileged modes.  
\_\_STATIC\_FORCEINLINE void \_\_enable\_fault\_irq(void)  
{  
 \_\_ASM volatile ("cpsie f" : : : "memory");  
}  
 \brief Disable FIQ  
 \details Disables FIQ interrupts by setting the F-bit in the CPSR.  
 Can only be executed in Privileged modes.  
\_\_STATIC\_FORCEINLINE void \_\_disable\_fault\_irq(void)  
{  
 \_\_ASM volatile ("cpsid f" : : : "memory");  
}  
 \brief Get Base Priority  
 \details Returns the current value of the Base Priority register.  
 \return Base Priority register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_BASEPRI(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, basepri" : "=r" (result) );  
 return(result);  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Get Base Priority (non-secure)  
 \details Returns the current value of the non-secure Base Priority register when in secure state.  
 \return Base Priority register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_TZ\_get\_BASEPRI\_NS(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, basepri\_ns" : "=r" (result) );  
 return(result);  
}  
#endif  
 \brief Set Base Priority  
 \details Assigns the given value to the Base Priority register.  
 \param [in] basePri Base Priority value to set  
\_\_STATIC\_FORCEINLINE void \_\_set\_BASEPRI(uint32\_t basePri)  
{  
 \_\_ASM volatile ("MSR basepri, %0" : : "r" (basePri) : "memory");  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Set Base Priority (non-secure)  
 \details Assigns the given value to the non-secure Base Priority register when in secure state.  
 \param [in] basePri Base Priority value to set  
\_\_STATIC\_FORCEINLINE void \_\_TZ\_set\_BASEPRI\_NS(uint32\_t basePri)  
{  
 \_\_ASM volatile ("MSR basepri\_ns, %0" : : "r" (basePri) : "memory");  
}  
#endif  
 \brief Set Base Priority with condition  
 \details Assigns the given value to the Base Priority register only if BASEPRI masking is disabled,  
 or the new value increases the BASEPRI priority level.  
 \param [in] basePri Base Priority value to set  
\_\_STATIC\_FORCEINLINE void \_\_set\_BASEPRI\_MAX(uint32\_t basePri)  
{  
 \_\_ASM volatile ("MSR basepri\_max, %0" : : "r" (basePri) : "memory");  
}  
 \brief Get Fault Mask  
 \details Returns the current value of the Fault Mask register.  
 \return Fault Mask register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_FAULTMASK(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, faultmask" : "=r" (result) );  
 return(result);  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Get Fault Mask (non-secure)  
 \details Returns the current value of the non-secure Fault Mask register when in secure state.  
 \return Fault Mask register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_TZ\_get\_FAULTMASK\_NS(void)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, faultmask\_ns" : "=r" (result) );  
 return(result);  
}  
#endif  
 \brief Set Fault Mask  
 \details Assigns the given value to the Fault Mask register.  
 \param [in] faultMask Fault Mask value to set  
\_\_STATIC\_FORCEINLINE void \_\_set\_FAULTMASK(uint32\_t faultMask)  
{  
 \_\_ASM volatile ("MSR faultmask, %0" : : "r" (faultMask) : "memory");  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Set Fault Mask (non-secure)  
 \details Assigns the given value to the non-secure Fault Mask register when in secure state.  
 \param [in] faultMask Fault Mask value to set  
\_\_STATIC\_FORCEINLINE void \_\_TZ\_set\_FAULTMASK\_NS(uint32\_t faultMask)  
{  
 \_\_ASM volatile ("MSR faultmask\_ns, %0" : : "r" (faultMask) : "memory");  
}  
#endif  
#endif /\* ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_ ) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) ) \*/  
#if ((defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_BASE\_\_ ) && (\_\_ARM\_ARCH\_8M\_BASE\_\_ == 1)) )  
 \brief Get Process Stack Pointer Limit  
 Devices without ARMv8-M Main Extensions (i.e. Cortex-M23) lack the non-secure  
 Stack Pointer Limit register hence zero is returned always in non-secure  
 mode.  
   
 \details Returns the current value of the Process Stack Pointer Limit (PSPLIM).  
 \return PSPLIM Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_PSPLIM(void)  
{  
#if (!(defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) && \  
 (!defined (\_\_ARM\_FEATURE\_CMSE) || (\_\_ARM\_FEATURE\_CMSE < 3)))  
 return 0U;  
#else  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, psplim" : "=r" (result) );  
 return result;  
#endif  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Get Process Stack Pointer Limit (non-secure)  
 Devices without ARMv8-M Main Extensions (i.e. Cortex-M23) lack the non-secure  
 Stack Pointer Limit register hence zero is returned always.  
 \details Returns the current value of the non-secure Process Stack Pointer Limit (PSPLIM) when in secure state.  
 \return PSPLIM Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_TZ\_get\_PSPLIM\_NS(void)  
{  
#if (!(defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)))  
 return 0U;  
#else  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, psplim\_ns" : "=r" (result) );  
 return result;  
#endif  
}  
#endif  
 \brief Set Process Stack Pointer Limit  
 Devices without ARMv8-M Main Extensions (i.e. Cortex-M23) lack the non-secure  
 Stack Pointer Limit register hence the write is silently ignored in non-secure  
 mode.  
   
 \details Assigns the given value to the Process Stack Pointer Limit (PSPLIM).  
 \param [in] ProcStackPtrLimit Process Stack Pointer Limit value to set  
\_\_STATIC\_FORCEINLINE void \_\_set\_PSPLIM(uint32\_t ProcStackPtrLimit)  
{  
#if (!(defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) && \  
 (!defined (\_\_ARM\_FEATURE\_CMSE) || (\_\_ARM\_FEATURE\_CMSE < 3)))  
 (void)ProcStackPtrLimit;  
#else  
 \_\_ASM volatile ("MSR psplim, %0" : : "r" (ProcStackPtrLimit));  
#endif  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Set Process Stack Pointer (non-secure)  
 Devices without ARMv8-M Main Extensions (i.e. Cortex-M23) lack the non-secure  
 Stack Pointer Limit register hence the write is silently ignored.  
 \details Assigns the given value to the non-secure Process Stack Pointer Limit (PSPLIM) when in secure state.  
 \param [in] ProcStackPtrLimit Process Stack Pointer Limit value to set  
\_\_STATIC\_FORCEINLINE void \_\_TZ\_set\_PSPLIM\_NS(uint32\_t ProcStackPtrLimit)  
{  
#if (!(defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)))  
 (void)ProcStackPtrLimit;  
#else  
 \_\_ASM volatile ("MSR psplim\_ns, %0\n" : : "r" (ProcStackPtrLimit));  
#endif  
}  
#endif  
 \brief Get Main Stack Pointer Limit  
 Devices without ARMv8-M Main Extensions (i.e. Cortex-M23) lack the non-secure  
 Stack Pointer Limit register hence zero is returned always in non-secure  
 mode.  
 \details Returns the current value of the Main Stack Pointer Limit (MSPLIM).  
 \return MSPLIM Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_MSPLIM(void)  
{  
#if (!(defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) && \  
 (!defined (\_\_ARM\_FEATURE\_CMSE) || (\_\_ARM\_FEATURE\_CMSE < 3)))  
 return 0U;  
#else  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, msplim" : "=r" (result) );  
 return result;  
#endif  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Get Main Stack Pointer Limit (non-secure)  
 Devices without ARMv8-M Main Extensions (i.e. Cortex-M23) lack the non-secure  
 Stack Pointer Limit register hence zero is returned always.  
 \details Returns the current value of the non-secure Main Stack Pointer Limit(MSPLIM) when in secure state.  
 \return MSPLIM Register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_TZ\_get\_MSPLIM\_NS(void)  
{  
#if (!(defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)))  
 return 0U;  
#else  
 uint32\_t result;  
 \_\_ASM volatile ("MRS %0, msplim\_ns" : "=r" (result) );  
 return result;  
#endif  
}  
#endif  
 \brief Set Main Stack Pointer Limit  
 Devices without ARMv8-M Main Extensions (i.e. Cortex-M23) lack the non-secure  
 Stack Pointer Limit register hence the write is silently ignored in non-secure  
 mode.  
 \details Assigns the given value to the Main Stack Pointer Limit (MSPLIM).  
 \param [in] MainStackPtrLimit Main Stack Pointer Limit value to set  
\_\_STATIC\_FORCEINLINE void \_\_set\_MSPLIM(uint32\_t MainStackPtrLimit)  
{  
#if (!(defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) && \  
 (!defined (\_\_ARM\_FEATURE\_CMSE) || (\_\_ARM\_FEATURE\_CMSE < 3)))  
 (void)MainStackPtrLimit;  
#else  
 \_\_ASM volatile ("MSR msplim, %0" : : "r" (MainStackPtrLimit));  
#endif  
}  
#if (defined (\_\_ARM\_FEATURE\_CMSE ) && (\_\_ARM\_FEATURE\_CMSE == 3))  
 \brief Set Main Stack Pointer Limit (non-secure)  
 Devices without ARMv8-M Main Extensions (i.e. Cortex-M23) lack the non-secure  
 Stack Pointer Limit register hence the write is silently ignored.  
 \details Assigns the given value to the non-secure Main Stack Pointer Limit (MSPLIM) when in secure state.  
 \param [in] MainStackPtrLimit Main Stack Pointer value to set  
\_\_STATIC\_FORCEINLINE void \_\_TZ\_set\_MSPLIM\_NS(uint32\_t MainStackPtrLimit)  
{  
#if (!(defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)))  
 (void)MainStackPtrLimit;  
#else  
 \_\_ASM volatile ("MSR msplim\_ns, %0" : : "r" (MainStackPtrLimit));  
#endif  
}  
#endif  
#endif /\* ((defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_BASE\_\_ ) && (\_\_ARM\_ARCH\_8M\_BASE\_\_ == 1)) ) \*/  
 \brief Get FPSCR  
 \details Returns the current value of the Floating Point Status/Control register.  
 \return Floating Point Status/Control register value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_get\_FPSCR(void)  
{  
#if ((defined (\_\_FPU\_PRESENT) && (\_\_FPU\_PRESENT == 1U)) && \  
 (defined (\_\_FPU\_USED ) && (\_\_FPU\_USED == 1U)) )  
#if \_\_has\_builtin(\_\_builtin\_arm\_get\_fpscr)   
 return \_\_builtin\_arm\_get\_fpscr();  
#else  
 uint32\_t result;  
 \_\_ASM volatile ("VMRS %0, fpscr" : "=r" (result) );  
 return(result);  
#endif  
#else  
 return(0U);  
#endif  
}  
 \brief Set FPSCR  
 \details Assigns the given value to the Floating Point Status/Control register.  
 \param [in] fpscr Floating Point Status/Control value to set  
\_\_STATIC\_FORCEINLINE void \_\_set\_FPSCR(uint32\_t fpscr)  
{  
#if ((defined (\_\_FPU\_PRESENT) && (\_\_FPU\_PRESENT == 1U)) && \  
 (defined (\_\_FPU\_USED ) && (\_\_FPU\_USED == 1U)) )  
#if \_\_has\_builtin(\_\_builtin\_arm\_set\_fpscr)  
 \_\_builtin\_arm\_set\_fpscr(fpscr);  
#else  
 \_\_ASM volatile ("VMSR fpscr, %0" : : "r" (fpscr) : "vfpcc", "memory");  
#endif  
#else  
 (void)fpscr;  
#endif  
}  
 Access to dedicated instructions  
 @{  
#if defined (\_\_thumb\_\_) && !defined (\_\_thumb2\_\_)  
#define \_\_CMSIS\_GCC\_OUT\_REG(r) "=l" (r)  
#define \_\_CMSIS\_GCC\_RW\_REG(r) "+l" (r)  
#define \_\_CMSIS\_GCC\_USE\_REG(r) "l" (r)  
#else  
#define \_\_CMSIS\_GCC\_OUT\_REG(r) "=r" (r)  
#define \_\_CMSIS\_GCC\_RW\_REG(r) "+r" (r)  
#define \_\_CMSIS\_GCC\_USE\_REG(r) "r" (r)  
#endif  
 \brief No Operation  
 \details No Operation does nothing. This instruction can be used for code alignment purposes.  
#define \_\_NOP() \_\_ASM volatile ("nop")  
 \brief Wait For Interrupt  
 \details Wait For Interrupt is a hint instruction that suspends execution until one of a number of events occurs.  
#define \_\_WFI() \_\_ASM volatile ("wfi")  
 \brief Wait For Event  
 \details Wait For Event is a hint instruction that permits the processor to enter  
 a low-power state until one of a number of events occurs.  
#define \_\_WFE() \_\_ASM volatile ("wfe")  
 \brief Send Event  
 \details Send Event is a hint instruction. It causes an event to be signaled to the CPU.  
#define \_\_SEV() \_\_ASM volatile ("sev")  
 \brief Instruction Synchronization Barrier  
 \details Instruction Synchronization Barrier flushes the pipeline in the processor,  
 so that all instructions following the ISB are fetched from cache or memory,  
 after the instruction has been completed.  
\_\_STATIC\_FORCEINLINE void \_\_ISB(void)  
{  
 \_\_ASM volatile ("isb 0xF":::"memory");  
}  
 \brief Data Synchronization Barrier  
 \details Acts as a special kind of Data Memory Barrier.  
 It completes when all explicit memory accesses before this instruction complete.  
\_\_STATIC\_FORCEINLINE void \_\_DSB(void)  
{  
 \_\_ASM volatile ("dsb 0xF":::"memory");  
}  
 \brief Data Memory Barrier  
 \details Ensures the apparent order of the explicit memory operations before  
 and after the instruction, without ensuring their completion.  
\_\_STATIC\_FORCEINLINE void \_\_DMB(void)  
{  
 \_\_ASM volatile ("dmb 0xF":::"memory");  
}  
 \brief Reverse byte order (32 bit)  
 \details Reverses the byte order in unsigned integer value. For example, 0x12345678 becomes 0x78563412.  
 \param [in] value Value to reverse  
 \return Reversed value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_REV(uint32\_t value)  
{  
#if (\_\_GNUC\_\_ > 4) || (\_\_GNUC\_\_ == 4 && \_\_GNUC\_MINOR\_\_ >= 5)  
 return \_\_builtin\_bswap32(value);  
#else  
 uint32\_t result;  
 \_\_ASM volatile ("rev %0, %1" : \_\_CMSIS\_GCC\_OUT\_REG (result) : \_\_CMSIS\_GCC\_USE\_REG (value) );  
 return result;  
#endif  
}  
 \brief Reverse byte order (16 bit)  
 \details Reverses the byte order within each halfword of a word. For example, 0x12345678 becomes 0x34127856.  
 \param [in] value Value to reverse  
 \return Reversed value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_REV16(uint32\_t value)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("rev16 %0, %1" : \_\_CMSIS\_GCC\_OUT\_REG (result) : \_\_CMSIS\_GCC\_USE\_REG (value) );  
 return result;  
}  
 \brief Reverse byte order (16 bit)  
 \details Reverses the byte order in a 16-bit value and returns the signed 16-bit result. For example, 0x0080 becomes 0x8000.  
 \param [in] value Value to reverse  
 \return Reversed value  
\_\_STATIC\_FORCEINLINE int16\_t \_\_REVSH(int16\_t value)  
{  
#if (\_\_GNUC\_\_ > 4) || (\_\_GNUC\_\_ == 4 && \_\_GNUC\_MINOR\_\_ >= 8)  
 return (int16\_t)\_\_builtin\_bswap16(value);  
#else  
 int16\_t result;  
 \_\_ASM volatile ("revsh %0, %1" : \_\_CMSIS\_GCC\_OUT\_REG (result) : \_\_CMSIS\_GCC\_USE\_REG (value) );  
 return result;  
#endif  
}  
 \brief Rotate Right in unsigned value (32 bit)  
 \details Rotate Right (immediate) provides the value of the contents of a register rotated by a variable number of bits.  
 \param [in] op1 Value to rotate  
 \param [in] op2 Number of Bits to rotate  
 \return Rotated value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_ROR(uint32\_t op1, uint32\_t op2)  
{  
 op2 %= 32U;  
 if (op2 == 0U)  
 {  
 return op1;  
 }  
 return (op1 >> op2) | (op1 << (32U - op2));  
}  
 \brief Breakpoint  
 \details Causes the processor to enter Debug state.  
 Debug tools can use this to investigate system state when the instruction at a particular address is reached.  
 \param [in] value is ignored by the processor.  
 If required, a debugger can use it to store additional information about the breakpoint.  
#define \_\_BKPT(value) \_\_ASM volatile ("bkpt "#value)  
 \brief Reverse bit order of value  
 \details Reverses the bit order of the given value.  
 \param [in] value Value to reverse  
 \return Reversed value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_RBIT(uint32\_t value)  
{  
 uint32\_t result;  
#if ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_ ) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) )  
 \_\_ASM volatile ("rbit %0, %1" : "=r" (result) : "r" (value) );  
#else  
 uint32\_t s = (4U /\*sizeof(v)\*/ \* 8U) - 1U; /\* extra shift needed at end \*/  
 result = value; /\* r will be reversed bits of v; first get LSB of v \*/  
 for (value >>= 1U; value != 0U; value >>= 1U)  
 {  
 result <<= 1U;  
 result |= value & 1U;  
 s--;  
 }  
 result <<= s; /\* shift when v's highest bits are zero \*/  
#endif  
 return result;  
}  
 \brief Count leading zeros  
 \details Counts the number of leading zeros of a data value.  
 \param [in] value Value to count the leading zeros  
 \return number of leading zeros in value  
#define \_\_CLZ (uint8\_t)\_\_builtin\_clz  
#if ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_ ) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_BASE\_\_ ) && (\_\_ARM\_ARCH\_8M\_BASE\_\_ == 1)) )  
 \brief LDR Exclusive (8 bit)  
 \details Executes a exclusive LDR instruction for 8 bit value.  
 \param [in] ptr Pointer to data  
 \return value of type uint8\_t at (\*ptr)  
\_\_STATIC\_FORCEINLINE uint8\_t \_\_LDREXB(volatile uint8\_t \*addr)  
{  
 uint32\_t result;  
#if (\_\_GNUC\_\_ > 4) || (\_\_GNUC\_\_ == 4 && \_\_GNUC\_MINOR\_\_ >= 8)  
 \_\_ASM volatile ("ldrexb %0, %1" : "=r" (result) : "Q" (\*addr) );  
#else  
 accepted by assembler. So has to use following less efficient pattern.  
 \_\_ASM volatile ("ldrexb %0, [%1]" : "=r" (result) : "r" (addr) : "memory" );  
#endif  
 return ((uint8\_t) result); /\* Add explicit type cast here \*/  
}  
 \brief LDR Exclusive (16 bit)  
 \details Executes a exclusive LDR instruction for 16 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint16\_t at (\*ptr)  
\_\_STATIC\_FORCEINLINE uint16\_t \_\_LDREXH(volatile uint16\_t \*addr)  
{  
 uint32\_t result;  
#if (\_\_GNUC\_\_ > 4) || (\_\_GNUC\_\_ == 4 && \_\_GNUC\_MINOR\_\_ >= 8)  
 \_\_ASM volatile ("ldrexh %0, %1" : "=r" (result) : "Q" (\*addr) );  
#else  
 accepted by assembler. So has to use following less efficient pattern.  
 \_\_ASM volatile ("ldrexh %0, [%1]" : "=r" (result) : "r" (addr) : "memory" );  
#endif  
 return ((uint16\_t) result); /\* Add explicit type cast here \*/  
}  
 \brief LDR Exclusive (32 bit)  
 \details Executes a exclusive LDR instruction for 32 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint32\_t at (\*ptr)  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_LDREXW(volatile uint32\_t \*addr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("ldrex %0, %1" : "=r" (result) : "Q" (\*addr) );  
 return(result);  
}  
 \brief STR Exclusive (8 bit)  
 \details Executes a exclusive STR instruction for 8 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
 \return 0 Function succeeded  
 \return 1 Function failed  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_STREXB(uint8\_t value, volatile uint8\_t \*addr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("strexb %0, %2, %1" : "=&r" (result), "=Q" (\*addr) : "r" ((uint32\_t)value) );  
 return(result);  
}  
 \brief STR Exclusive (16 bit)  
 \details Executes a exclusive STR instruction for 16 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
 \return 0 Function succeeded  
 \return 1 Function failed  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_STREXH(uint16\_t value, volatile uint16\_t \*addr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("strexh %0, %2, %1" : "=&r" (result), "=Q" (\*addr) : "r" ((uint32\_t)value) );  
 return(result);  
}  
 \brief STR Exclusive (32 bit)  
 \details Executes a exclusive STR instruction for 32 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
 \return 0 Function succeeded  
 \return 1 Function failed  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_STREXW(uint32\_t value, volatile uint32\_t \*addr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("strex %0, %2, %1" : "=&r" (result), "=Q" (\*addr) : "r" (value) );  
 return(result);  
}  
 \brief Remove the exclusive lock  
 \details Removes the exclusive lock which is created by LDREX.  
\_\_STATIC\_FORCEINLINE void \_\_CLREX(void)  
{  
 \_\_ASM volatile ("clrex" ::: "memory");  
}  
#endif /\* ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_ ) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_BASE\_\_ ) && (\_\_ARM\_ARCH\_8M\_BASE\_\_ == 1)) ) \*/  
#if ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_ ) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) )  
 \brief Signed Saturate  
 \details Saturates a signed value.  
 \param [in] ARG1 Value to be saturated  
 \param [in] ARG2 Bit position to saturate to (1..32)  
 \return Saturated value  
#define \_\_SSAT(ARG1,ARG2) \  
\_\_extension\_\_ \  
({ \  
 int32\_t \_\_RES, \_\_ARG1 = (ARG1); \  
 \_\_ASM ("ssat %0, %1, %2" : "=r" (\_\_RES) : "I" (ARG2), "r" (\_\_ARG1) ); \  
 \_\_RES; \  
 })  
 \brief Unsigned Saturate  
 \details Saturates an unsigned value.  
 \param [in] ARG1 Value to be saturated  
 \param [in] ARG2 Bit position to saturate to (0..31)  
 \return Saturated value  
#define \_\_USAT(ARG1,ARG2) \  
 \_\_extension\_\_ \  
({ \  
 uint32\_t \_\_RES, \_\_ARG1 = (ARG1); \  
 \_\_ASM ("usat %0, %1, %2" : "=r" (\_\_RES) : "I" (ARG2), "r" (\_\_ARG1) ); \  
 \_\_RES; \  
 })  
 \brief Rotate Right with Extend (32 bit)  
 \details Moves each bit of a bitstring right by one bit.  
 The carry input is shifted in at the left end of the bitstring.  
 \param [in] value Value to rotate  
 \return Rotated value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_RRX(uint32\_t value)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("rrx %0, %1" : \_\_CMSIS\_GCC\_OUT\_REG (result) : \_\_CMSIS\_GCC\_USE\_REG (value) );  
 return(result);  
}  
 \brief LDRT Unprivileged (8 bit)  
 \details Executes a Unprivileged LDRT instruction for 8 bit value.  
 \param [in] ptr Pointer to data  
 \return value of type uint8\_t at (\*ptr)  
\_\_STATIC\_FORCEINLINE uint8\_t \_\_LDRBT(volatile uint8\_t \*ptr)  
{  
 uint32\_t result;  
#if (\_\_GNUC\_\_ > 4) || (\_\_GNUC\_\_ == 4 && \_\_GNUC\_MINOR\_\_ >= 8)  
 \_\_ASM volatile ("ldrbt %0, %1" : "=r" (result) : "Q" (\*ptr) );  
#else  
 accepted by assembler. So has to use following less efficient pattern.  
 \_\_ASM volatile ("ldrbt %0, [%1]" : "=r" (result) : "r" (ptr) : "memory" );  
#endif  
 return ((uint8\_t) result); /\* Add explicit type cast here \*/  
}  
 \brief LDRT Unprivileged (16 bit)  
 \details Executes a Unprivileged LDRT instruction for 16 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint16\_t at (\*ptr)  
\_\_STATIC\_FORCEINLINE uint16\_t \_\_LDRHT(volatile uint16\_t \*ptr)  
{  
 uint32\_t result;  
#if (\_\_GNUC\_\_ > 4) || (\_\_GNUC\_\_ == 4 && \_\_GNUC\_MINOR\_\_ >= 8)  
 \_\_ASM volatile ("ldrht %0, %1" : "=r" (result) : "Q" (\*ptr) );  
#else  
 accepted by assembler. So has to use following less efficient pattern.  
 \_\_ASM volatile ("ldrht %0, [%1]" : "=r" (result) : "r" (ptr) : "memory" );  
#endif  
 return ((uint16\_t) result); /\* Add explicit type cast here \*/  
}  
 \brief LDRT Unprivileged (32 bit)  
 \details Executes a Unprivileged LDRT instruction for 32 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint32\_t at (\*ptr)  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_LDRT(volatile uint32\_t \*ptr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("ldrt %0, %1" : "=r" (result) : "Q" (\*ptr) );  
 return(result);  
}  
 \brief STRT Unprivileged (8 bit)  
 \details Executes a Unprivileged STRT instruction for 8 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
\_\_STATIC\_FORCEINLINE void \_\_STRBT(uint8\_t value, volatile uint8\_t \*ptr)  
{  
 \_\_ASM volatile ("strbt %1, %0" : "=Q" (\*ptr) : "r" ((uint32\_t)value) );  
}  
 \brief STRT Unprivileged (16 bit)  
 \details Executes a Unprivileged STRT instruction for 16 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
\_\_STATIC\_FORCEINLINE void \_\_STRHT(uint16\_t value, volatile uint16\_t \*ptr)  
{  
 \_\_ASM volatile ("strht %1, %0" : "=Q" (\*ptr) : "r" ((uint32\_t)value) );  
}  
 \brief STRT Unprivileged (32 bit)  
 \details Executes a Unprivileged STRT instruction for 32 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
\_\_STATIC\_FORCEINLINE void \_\_STRT(uint32\_t value, volatile uint32\_t \*ptr)  
{  
 \_\_ASM volatile ("strt %1, %0" : "=Q" (\*ptr) : "r" (value) );  
}  
#else /\* ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_ ) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) ) \*/  
 \brief Signed Saturate  
 \details Saturates a signed value.  
 \param [in] value Value to be saturated  
 \param [in] sat Bit position to saturate to (1..32)  
 \return Saturated value  
\_\_STATIC\_FORCEINLINE int32\_t \_\_SSAT(int32\_t val, uint32\_t sat)  
{  
 if ((sat >= 1U) && (sat <= 32U))  
 {  
 const int32\_t max = (int32\_t)((1U << (sat - 1U)) - 1U);  
 const int32\_t min = -1 - max ;  
 if (val > max)  
 {  
 return max;  
 }  
 else if (val < min)  
 {  
 return min;  
 }  
 }  
 return val;  
}  
 \brief Unsigned Saturate  
 \details Saturates an unsigned value.  
 \param [in] value Value to be saturated  
 \param [in] sat Bit position to saturate to (0..31)  
 \return Saturated value  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_USAT(int32\_t val, uint32\_t sat)  
{  
 if (sat <= 31U)  
 {  
 const uint32\_t max = ((1U << sat) - 1U);  
 if (val > (int32\_t)max)  
 {  
 return max;  
 }  
 else if (val < 0)  
 {  
 return 0U;  
 }  
 }  
 return (uint32\_t)val;  
}  
#endif /\* ((defined (\_\_ARM\_ARCH\_7M\_\_ ) && (\_\_ARM\_ARCH\_7M\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_7EM\_\_ ) && (\_\_ARM\_ARCH\_7EM\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) ) \*/  
#if ((defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_BASE\_\_ ) && (\_\_ARM\_ARCH\_8M\_BASE\_\_ == 1)) )  
 \brief Load-Acquire (8 bit)  
 \details Executes a LDAB instruction for 8 bit value.  
 \param [in] ptr Pointer to data  
 \return value of type uint8\_t at (\*ptr)  
\_\_STATIC\_FORCEINLINE uint8\_t \_\_LDAB(volatile uint8\_t \*ptr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("ldab %0, %1" : "=r" (result) : "Q" (\*ptr) );  
 return ((uint8\_t) result);  
}  
 \brief Load-Acquire (16 bit)  
 \details Executes a LDAH instruction for 16 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint16\_t at (\*ptr)  
\_\_STATIC\_FORCEINLINE uint16\_t \_\_LDAH(volatile uint16\_t \*ptr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("ldah %0, %1" : "=r" (result) : "Q" (\*ptr) );  
 return ((uint16\_t) result);  
}  
 \brief Load-Acquire (32 bit)  
 \details Executes a LDA instruction for 32 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint32\_t at (\*ptr)  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_LDA(volatile uint32\_t \*ptr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("lda %0, %1" : "=r" (result) : "Q" (\*ptr) );  
 return(result);  
}  
 \brief Store-Release (8 bit)  
 \details Executes a STLB instruction for 8 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
\_\_STATIC\_FORCEINLINE void \_\_STLB(uint8\_t value, volatile uint8\_t \*ptr)  
{  
 \_\_ASM volatile ("stlb %1, %0" : "=Q" (\*ptr) : "r" ((uint32\_t)value) );  
}  
 \brief Store-Release (16 bit)  
 \details Executes a STLH instruction for 16 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
\_\_STATIC\_FORCEINLINE void \_\_STLH(uint16\_t value, volatile uint16\_t \*ptr)  
{  
 \_\_ASM volatile ("stlh %1, %0" : "=Q" (\*ptr) : "r" ((uint32\_t)value) );  
}  
 \brief Store-Release (32 bit)  
 \details Executes a STL instruction for 32 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
\_\_STATIC\_FORCEINLINE void \_\_STL(uint32\_t value, volatile uint32\_t \*ptr)  
{  
 \_\_ASM volatile ("stl %1, %0" : "=Q" (\*ptr) : "r" ((uint32\_t)value) );  
}  
 \brief Load-Acquire Exclusive (8 bit)  
 \details Executes a LDAB exclusive instruction for 8 bit value.  
 \param [in] ptr Pointer to data  
 \return value of type uint8\_t at (\*ptr)  
\_\_STATIC\_FORCEINLINE uint8\_t \_\_LDAEXB(volatile uint8\_t \*ptr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("ldaexb %0, %1" : "=r" (result) : "Q" (\*ptr) );  
 return ((uint8\_t) result);  
}  
 \brief Load-Acquire Exclusive (16 bit)  
 \details Executes a LDAH exclusive instruction for 16 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint16\_t at (\*ptr)  
\_\_STATIC\_FORCEINLINE uint16\_t \_\_LDAEXH(volatile uint16\_t \*ptr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("ldaexh %0, %1" : "=r" (result) : "Q" (\*ptr) );  
 return ((uint16\_t) result);  
}  
 \brief Load-Acquire Exclusive (32 bit)  
 \details Executes a LDA exclusive instruction for 32 bit values.  
 \param [in] ptr Pointer to data  
 \return value of type uint32\_t at (\*ptr)  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_LDAEX(volatile uint32\_t \*ptr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("ldaex %0, %1" : "=r" (result) : "Q" (\*ptr) );  
 return(result);  
}  
 \brief Store-Release Exclusive (8 bit)  
 \details Executes a STLB exclusive instruction for 8 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
 \return 0 Function succeeded  
 \return 1 Function failed  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_STLEXB(uint8\_t value, volatile uint8\_t \*ptr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("stlexb %0, %2, %1" : "=&r" (result), "=Q" (\*ptr) : "r" ((uint32\_t)value) );  
 return(result);  
}  
 \brief Store-Release Exclusive (16 bit)  
 \details Executes a STLH exclusive instruction for 16 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
 \return 0 Function succeeded  
 \return 1 Function failed  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_STLEXH(uint16\_t value, volatile uint16\_t \*ptr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("stlexh %0, %2, %1" : "=&r" (result), "=Q" (\*ptr) : "r" ((uint32\_t)value) );  
 return(result);  
}  
 \brief Store-Release Exclusive (32 bit)  
 \details Executes a STL exclusive instruction for 32 bit values.  
 \param [in] value Value to store  
 \param [in] ptr Pointer to location  
 \return 0 Function succeeded  
 \return 1 Function failed  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_STLEX(uint32\_t value, volatile uint32\_t \*ptr)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("stlex %0, %2, %1" : "=&r" (result), "=Q" (\*ptr) : "r" ((uint32\_t)value) );  
 return(result);  
}  
#endif /\* ((defined (\_\_ARM\_ARCH\_8M\_MAIN\_\_ ) && (\_\_ARM\_ARCH\_8M\_MAIN\_\_ == 1)) || \  
 (defined (\_\_ARM\_ARCH\_8M\_BASE\_\_ ) && (\_\_ARM\_ARCH\_8M\_BASE\_\_ == 1)) ) \*/  
 Access to dedicated SIMD instructions  
 @{  
#if (defined (\_\_ARM\_FEATURE\_DSP) && (\_\_ARM\_FEATURE\_DSP == 1))  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SADD8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("sadd8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_QADD8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("qadd8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SHADD8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("shadd8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UADD8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uadd8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UQADD8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uqadd8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UHADD8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uhadd8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SSUB8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("ssub8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_QSUB8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("qsub8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SHSUB8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("shsub8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_USUB8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("usub8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UQSUB8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uqsub8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UHSUB8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uhsub8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SADD16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("sadd16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_QADD16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("qadd16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SHADD16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("shadd16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UADD16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uadd16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UQADD16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uqadd16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UHADD16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uhadd16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SSUB16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("ssub16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_QSUB16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("qsub16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SHSUB16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("shsub16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_USUB16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("usub16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UQSUB16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uqsub16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UHSUB16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uhsub16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SASX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("sasx %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_QASX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("qasx %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SHASX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("shasx %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UASX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uasx %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UQASX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uqasx %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UHASX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uhasx %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SSAX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("ssax %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_QSAX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("qsax %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SHSAX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("shsax %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_USAX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("usax %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UQSAX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uqsax %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UHSAX(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uhsax %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_USAD8(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("usad8 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_USADA8(uint32\_t op1, uint32\_t op2, uint32\_t op3)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("usada8 %0, %1, %2, %3" : "=r" (result) : "r" (op1), "r" (op2), "r" (op3) );  
 return(result);  
}  
#define \_\_SSAT16(ARG1,ARG2) \  
({ \  
 int32\_t \_\_RES, \_\_ARG1 = (ARG1); \  
 \_\_ASM ("ssat16 %0, %1, %2" : "=r" (\_\_RES) : "I" (ARG2), "r" (\_\_ARG1) ); \  
 \_\_RES; \  
 })  
#define \_\_USAT16(ARG1,ARG2) \  
({ \  
 uint32\_t \_\_RES, \_\_ARG1 = (ARG1); \  
 \_\_ASM ("usat16 %0, %1, %2" : "=r" (\_\_RES) : "I" (ARG2), "r" (\_\_ARG1) ); \  
 \_\_RES; \  
 })  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UXTB16(uint32\_t op1)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uxtb16 %0, %1" : "=r" (result) : "r" (op1));  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_UXTAB16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("uxtab16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SXTB16(uint32\_t op1)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("sxtb16 %0, %1" : "=r" (result) : "r" (op1));  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SXTAB16(uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("sxtab16 %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SMUAD (uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("smuad %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SMUADX (uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("smuadx %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SMLAD (uint32\_t op1, uint32\_t op2, uint32\_t op3)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("smlad %0, %1, %2, %3" : "=r" (result) : "r" (op1), "r" (op2), "r" (op3) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SMLADX (uint32\_t op1, uint32\_t op2, uint32\_t op3)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("smladx %0, %1, %2, %3" : "=r" (result) : "r" (op1), "r" (op2), "r" (op3) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint64\_t \_\_SMLALD (uint32\_t op1, uint32\_t op2, uint64\_t acc)  
{  
 union llreg\_u{  
 uint32\_t w32[2];  
 uint64\_t w64;  
 } llr;  
 llr.w64 = acc;  
#ifndef \_\_ARMEB\_\_ /\* Little endian \*/  
 \_\_ASM volatile ("smlald %0, %1, %2, %3" : "=r" (llr.w32[0]), "=r" (llr.w32[1]): "r" (op1), "r" (op2) , "0" (llr.w32[0]), "1" (llr.w32[1]) );  
#else /\* Big endian \*/  
 \_\_ASM volatile ("smlald %0, %1, %2, %3" : "=r" (llr.w32[1]), "=r" (llr.w32[0]): "r" (op1), "r" (op2) , "0" (llr.w32[1]), "1" (llr.w32[0]) );  
#endif  
 return(llr.w64);  
}  
\_\_STATIC\_FORCEINLINE uint64\_t \_\_SMLALDX (uint32\_t op1, uint32\_t op2, uint64\_t acc)  
{  
 union llreg\_u{  
 uint32\_t w32[2];  
 uint64\_t w64;  
 } llr;  
 llr.w64 = acc;  
#ifndef \_\_ARMEB\_\_ /\* Little endian \*/  
 \_\_ASM volatile ("smlaldx %0, %1, %2, %3" : "=r" (llr.w32[0]), "=r" (llr.w32[1]): "r" (op1), "r" (op2) , "0" (llr.w32[0]), "1" (llr.w32[1]) );  
#else /\* Big endian \*/  
 \_\_ASM volatile ("smlaldx %0, %1, %2, %3" : "=r" (llr.w32[1]), "=r" (llr.w32[0]): "r" (op1), "r" (op2) , "0" (llr.w32[1]), "1" (llr.w32[0]) );  
#endif  
 return(llr.w64);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SMUSD (uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("smusd %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SMUSDX (uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("smusdx %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SMLSD (uint32\_t op1, uint32\_t op2, uint32\_t op3)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("smlsd %0, %1, %2, %3" : "=r" (result) : "r" (op1), "r" (op2), "r" (op3) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SMLSDX (uint32\_t op1, uint32\_t op2, uint32\_t op3)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("smlsdx %0, %1, %2, %3" : "=r" (result) : "r" (op1), "r" (op2), "r" (op3) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE uint64\_t \_\_SMLSLD (uint32\_t op1, uint32\_t op2, uint64\_t acc)  
{  
 union llreg\_u{  
 uint32\_t w32[2];  
 uint64\_t w64;  
 } llr;  
 llr.w64 = acc;  
#ifndef \_\_ARMEB\_\_ /\* Little endian \*/  
 \_\_ASM volatile ("smlsld %0, %1, %2, %3" : "=r" (llr.w32[0]), "=r" (llr.w32[1]): "r" (op1), "r" (op2) , "0" (llr.w32[0]), "1" (llr.w32[1]) );  
#else /\* Big endian \*/  
 \_\_ASM volatile ("smlsld %0, %1, %2, %3" : "=r" (llr.w32[1]), "=r" (llr.w32[0]): "r" (op1), "r" (op2) , "0" (llr.w32[1]), "1" (llr.w32[0]) );  
#endif  
 return(llr.w64);  
}  
\_\_STATIC\_FORCEINLINE uint64\_t \_\_SMLSLDX (uint32\_t op1, uint32\_t op2, uint64\_t acc)  
{  
 union llreg\_u{  
 uint32\_t w32[2];  
 uint64\_t w64;  
 } llr;  
 llr.w64 = acc;  
#ifndef \_\_ARMEB\_\_ /\* Little endian \*/  
 \_\_ASM volatile ("smlsldx %0, %1, %2, %3" : "=r" (llr.w32[0]), "=r" (llr.w32[1]): "r" (op1), "r" (op2) , "0" (llr.w32[0]), "1" (llr.w32[1]) );  
#else /\* Big endian \*/  
 \_\_ASM volatile ("smlsldx %0, %1, %2, %3" : "=r" (llr.w32[1]), "=r" (llr.w32[0]): "r" (op1), "r" (op2) , "0" (llr.w32[1]), "1" (llr.w32[0]) );  
#endif  
 return(llr.w64);  
}  
\_\_STATIC\_FORCEINLINE uint32\_t \_\_SEL (uint32\_t op1, uint32\_t op2)  
{  
 uint32\_t result;  
 \_\_ASM volatile ("sel %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE int32\_t \_\_QADD( int32\_t op1, int32\_t op2)  
{  
 int32\_t result;  
 \_\_ASM volatile ("qadd %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
\_\_STATIC\_FORCEINLINE int32\_t \_\_QSUB( int32\_t op1, int32\_t op2)  
{  
 int32\_t result;  
 \_\_ASM volatile ("qsub %0, %1, %2" : "=r" (result) : "r" (op1), "r" (op2) );  
 return(result);  
}  
#if 0  
#define \_\_PKHBT(ARG1,ARG2,ARG3) \  
({ \  
 uint32\_t \_\_RES, \_\_ARG1 = (ARG1), \_\_ARG2 = (ARG2); \  
 \_\_ASM ("pkhbt %0, %1, %2, lsl %3" : "=r" (\_\_RES) : "r" (\_\_ARG1), "r" (\_\_ARG2), "I" (ARG3) ); \  
 \_\_RES; \  
 })  
#define \_\_PKHTB(ARG1,ARG2,ARG3) \  
({ \  
 uint32\_t \_\_RES, \_\_ARG1 = (ARG1), \_\_ARG2 = (ARG2); \  
 if (ARG3 == 0) \  
 \_\_ASM ("pkhtb %0, %1, %2" : "=r" (\_\_RES) : "r" (\_\_ARG1), "r" (\_\_ARG2) ); \  
 else \  
 \_\_ASM ("pkhtb %0, %1, %2, asr %3" : "=r" (\_\_RES) : "r" (\_\_ARG1), "r" (\_\_ARG2), "I" (ARG3) ); \  
 \_\_RES; \  
 })  
#endif  
#define \_\_PKHBT(ARG1,ARG2,ARG3) ( ((((uint32\_t)(ARG1)) ) & 0x0000FFFFUL) | \  
 ((((uint32\_t)(ARG2)) << (ARG3)) & 0xFFFF0000UL) )  
#define \_\_PKHTB(ARG1,ARG2,ARG3) ( ((((uint32\_t)(ARG1)) ) & 0xFFFF0000UL) | \  
 ((((uint32\_t)(ARG2)) >> (ARG3)) & 0x0000FFFFUL) )  
\_\_STATIC\_FORCEINLINE int32\_t \_\_SMMLA (int32\_t op1, int32\_t op2, int32\_t op3)  
{  
 int32\_t result;  
 \_\_ASM volatile ("smmla %0, %1, %2, %3" : "=r" (result): "r" (op1), "r" (op2), "r" (op3) );  
 return(result);  
}  
#endif /\* (\_\_ARM\_FEATURE\_DSP == 1) \*/  
#pragma GCC diagnostic pop  
#endif /\* \_\_CMSIS\_GCC\_H \*/