Structures in C

Code:

```
typedef struct point
{
  uint16_t x;
  uint8_t y;

}point;

void delay(void)
{
  int volatile count = 0;

  while(count < 1000000)
  {
    count++;
  }
}


int main()
{
  point p1, p2;

  p1.x = sizeof(point);
  p2.y = p1.x - 3u;
```

In "watch" view the p1 variable shows that the size of the structure is 4 bytes instead of 3 bytes:

Watch 1

Expression	Value	Location	Type
p1	<struct>	0x2000'0fe8	point
x	4	0x2000'0fe8	uint16_t
y	'\0' (0x00)	0x2000'0fea	uint8_t
<click to add>			

The elements of the structure is saved in:

Stack 1

CSTACK

Location	Data	Variable
0x2000'0ff0	0x4	p1.x
0x2000'0ff2	0x1	p1.y
0x2000'0ff3	0x0	p1 <pad>
0x2000'0ff4	0xf7	
0x2000'0ff8	0xfef5'eda5	
0x2000'0ffc	0xfef5'eda5	

This is because the structure has been padded by adding another byte at address 0x20000ff3 (an odd address)

To see why the padding bit is added we use __packed before struct:

```
typedef __packed struct point
{
  uint8_t y;
  uint16_t x;

}point;
```

This lets us remove the padding bits:

Location	Data	Variable
0x2000'0ff0	0xaa	p1.y
0x2000'0ff1	0x3	p1.x
0x2000'0ff3	0x0	no more pad
0x2000'0ff4	0xf3	
0x2000'0ff8	0xfef5'eda5	
0x2000'0ffc	0xfef5'eda5	

The padding is used as a mechanism to keep data aligned in even addresses. Without it, data is saved at an odd address.

Using CMSIs Libraries

This is a library of a set of structures to access various peripherals of the ARM MCU. For example the GPIO registers in the datasheet are simply a structure block with each register as a structure element at an offset from the base address. The structure for the GPIO block:

Register Map

Table 10-6. GPIO Register Map

Offset	Name	Type	Reset	Description
0x000	GPIODATA	RW	0x0000.0000	GPIO Data
0x400	GPIODIR	RW	0x0000.0000	GPIO Direction
0x404	GPIOIS	RW	0x0000.0000	GPIO Interrupt Sense
0x408	GPIOIBE	RW	0x0000.0000	GPIO Interrupt Both Edges
0x40C	GPIOIEV	RW	0x0000.0000	GPIO Interrupt Event
0x410	GPIOIM	RW	0x0000.0000	GPIO Interrupt Mask
0x414	GPIORIS	RO	0x0000.0000	GPIO Raw Interrupt Status
0x418	GPIOMIS	RO	0x0000.0000	GPIO Masked Interrupt Status
0x41C	GPIOICR	W1C	0x0000.0000	GPIO Interrupt Clear
0x420	GPIOAFSEL	RW	-	GPIO Alternate Function Select
0x500	GPIODR2R	RW	0x0000.00FF	GPIO 2-mA Drive Select
0x504	GPIODR4R	RW	0x0000.0000	GPIO 4-mA Drive Select
0x508	GPIODR8R	RW	0x0000.0000	GPIO 8-mA Drive Select
0x50C	GPIOODR	RW	0x0000.0000	GPIO Open Drain Select
0x510	GPIOPUR	RW	-	GPIO Pull-Up Select
0x514	GPIOPDR	RW	0x0000.0000	GPIO Pull-Down Select
0x518	GPIOSLR	RW	0x0000.0000	GPIO Slew Rate Control Select
0x51C	GPIODEN	RW	-	GPIO Digital Enable
0x520	GPIOLOCK	RW	0x0000.0001	GPIO Lock
0x524	GPIOCR	-	-	GPIO Commit
0x528	GPIOAMSEL	RW	0x0000.0000	GPIO Analog Mode Select
0x52C	GPIOPCTL	RW	-	GPIO Port Control

It's accompanying structure

To access 256 data registers

```
// <g> General Purpose Input/Output (GPIO)
typedef struct
{
    __IO uint32_t DATA_Bits[255];   // Bit specific data registers
    __IO uint32_t DATA;             // Data register.
    __IO uint32_t DIR;              // Data direction register.
    __IO uint32_t IS;               // Interrupt sense register.
    __IO uint32_t IBE;              // Interrupt both edges register.
    __IO uint32_t IEV;              // Interrupt event register.
    __IO uint32_t IM;               // Interrupt mask register.
    __I  uint32_t RIS;              // Raw interrupt status register.
    __I  uint32_t MIS;              // Masked interrupt status reg.
    __O  uint32_t ICR;              // Interrupt clear register.
    __IO uint32_t AFSEL;            // Mode control select register.
         uint8_t  RESERVED1[220];
    __IO uint32_t DR2R;             // 2ma drive select register.
    __IO uint32_t DR4R;             // 4ma drive select register.
    __IO uint32_t DR8R;             // 8ma drive select register.
    __IO uint32_t ODR;              // Open drain select register.
    __IO uint32_t PUR;              // Pull up select register.
    __IO uint32_t PDR;              // Pull down select register.
    __IO uint32_t SLR;              // Slew rate control enable reg.
    __IO uint32_t DEN;              // Digital input enable register.
    __IO uint32_t LOCK;             // Lock register.
    __O  uint32_t CR;               // Commit register.
    __IO uint32_t AMSEL;            // Analog Mode Select
    __IO uint32_t PCTL;             // Port Control
    __IO uint32_t ADCCTL;           // ADC Control
    __IO uint32_t DMACTL;           // DMA Control
} GPIO_Type;
// </g>
```

The Reserved[220] array is used to keep offset alignment of header file as same to the GPIO datasheet registers.

Note:

I/O → RW in datasheet

I → RO in datasheet

O → W1C in datasheet

At the end of the "tm4c_cmsis.h" header file, we find pointers to the structures we have defined till now, for example GPIOA refers to a pointer that is meant to point to the base address of Port A "GPIO_PORTA_BASE" which a macro for the address 0x40004000 (port A's APB address).

```
#define PERIPH_BASE             (0x40000000UL)

#define WATCHDOG_BASE           0x40000000  // Watchdog
#define GPIO_PORTA_BASE         0x40004000  // GPIO Port A
#define GPIO_PORTB_BASE         0x40005000  // GPIO Port B
#define GPIO_PORTC_BASE         0x40006000  // GPIO Port C
#define GPIO_PORTD_BASE         0x40007000  // GPIO Port D
#define SSI0_BASE               0x40008000  // SSI0
#define SSI1_BASE               0x40009000  // SSI1
#define UART0_BASE              0x4000C000  // UART0
#define UART1_BASE              0x4000D000  // UART1
#define UART2_BASE              0x4000E000  // UART2
#define I2C0_MASTER_BASE        0x40020000  // I2C0 Master
#define I2C0_SLAVE_BASE         0x40020800  // I2C0 Slave
#define I2C1_MASTER_BASE        0x40021000  // I2C1 Master
#define I2C1_SLAVE_BASE         0x40021800  // I2C1 Slave
#define GPIO_PORTE_BASE         0x40024000  // GPIO Port E
#define GPIO_PORTF_BASE         0x40025000  // GPIO Port F
#define GPIO_PORTG_BASE         0x40026000  // GPIO Port G
#define GPIO_PORTH_BASE         0x40027000  // GPIO Port H
#define PWM_BASE                0x40028000  // PWM
#define PWM_GEN_0_OFFSET        0x00000040  // PWM0 base
#define PWM_GEN_1_OFFSET        0x00000080  // PWM1 base
#define PWM_GEN_2_OFFSET        0x000000C0  // PWM2 base
#define PWM_GEN_3_OFFSET        0x00000100  // PWM3 base
#define QEI0_BASE               0x4002C000  // QEI0
#define QEI1_BASE               0x4002D000  // QEI1
#define TIMER0_BASE             0x40030000  // Timer0
#define TIMER1_BASE             0x40031000  // Timer1
#define TIMER2_BASE             0x40032000  // Timer2
#define TIMER3_BASE             0x40033000  // Timer3
#define ADC_BASE                0x40038000  // ADC
```

```
/*****************************************************************
/*                          Peripheral declaration
/*****************************************************************

#define SYSCTL              ((SYSCTL_Type *)SYSCTL_BASE)

#define GPIOA               ((GPIO_Type *)GPIO_PORTA_BASE)
#define GPIOB               ((GPIO_Type *)GPIO_PORTB_BASE)
#define GPIOC               ((GPIO_Type *)GPIO_PORTC_BASE)
#define GPIOD               ((GPIO_Type *)GPIO_PORTD_BASE)
#define GPIOE               ((GPIO_Type *)GPIO_PORTE_BASE)
#define GPIOF               ((GPIO_Type *)GPIO_PORTF_BASE)
#define GPIOG               ((GPIO_Type *)GPIO_PORTG_BASE)
#define GPIOA_HS            ((GPIO_Type *)GPIO_PORTA_AHB_BASE)
#define GPIOB_HS            ((GPIO_Type *)GPIO_PORTB_AHB_BASE)
#define GPIOC_HS            ((GPIO_Type *)GPIO_PORTC_AHB_BASE)
#define GPIOD_HS            ((GPIO_Type *)GPIO_PORTD_AHB_BASE)
#define GPIOE_HS            ((GPIO_Type *)GPIO_PORTE_AHB_BASE)
#define GPIOF_HS            ((GPIO_Type *)GPIO_PORTF_AHB_BASE)
#define GPIOG_HS            ((GPIO_Type *)GPIO_PORTG_AHB_BASE)

#define TIMER0              ((TIMER_Type *)TIMER0_BASE)
#define TIMER1              ((TIMER_Type *)TIMER1_BASE)
#define TIMER2              ((TIMER_Type *)TIMER2_BASE)
#define TIMER3              ((TIMER_Type *)TIMER3_BASE)

#define ADC                 ((ADC_Type *)ADC_BASE)

#define COMP                ((COMP_Type *)COMP_BASE)
```

We can now rewrite our blink program using new CMSIS compliant practices.

There is also another header file "core_cm4.h" that also includes structures such as the NVIC:

main.c * tm4c_cmsis.h core_cm4.h ×

```
  \defgroup   CMSIS_NVIC  Nested Vectored Interrupt Controller (NVIC)
  \brief      Type definitions for the NVIC Registers
  @{
 */

/**
  \brief  Structure type to access the Nested Vectored Interrupt Controller (NVIC).
 */
typedef struct
{
  __IOM uint32_t ISER[8U];               /*!< Offset: 0x000 (R/W)  Interrupt Set Enable Register */
        uint32_t RESERVED0[24U];
  __IOM uint32_t ICER[8U];               /*!< Offset: 0x080 (R/W)  Interrupt Clear Enable Register */
        uint32_t RESERVED1[24U];
  __IOM uint32_t ISPR[8U];               /*!< Offset: 0x100 (R/W)  Interrupt Set Pending Register */
        uint32_t RESERVED2[24U];
  __IOM uint32_t ICPR[8U];               /*!< Offset: 0x180 (R/W)  Interrupt Clear Pending Register */
        uint32_t RESERVED3[24U];
  __IOM uint32_t IABR[8U];               /*!< Offset: 0x200 (R/W)  Interrupt Active bit Register */
        uint32_t RESERVED4[56U];
  __IOM uint8_t  IP[240U];               /*!< Offset: 0x300 (R/W)  Interrupt Priority Register (8Bit wide) */
        uint32_t RESERVED5[644U];
  __OM  uint32_t STIR;                   /*!< Offset: 0xE00 ( /W)  Software Trigger Interrupt Register */
}  NVIC_Type;
```