SISTEMI EMBEDDED
AA 2012/2013

Fixed-size integer types
Bit Manipulation

Integer types

- 2 basic integer types: ***char, int***
- and some type-specifiers:
  - sign: *signed*, *unsigned*
  - size: *short*, *long*
- The actual size of an integer type depends on the compiler implementation
  - *sizeof*(*type*) returns the size (in number of bytes) used to represent the *type* argument
  - *sizeof*(*char*) ≤ *sizeof*(*short*) ≤ *sizeof*(*int*) ≤ *sizeof*(*long*)... ≤ *sizeof*(*long long*)

Fixed-size integers (1)

- In embedded system programming **integer size is important**
  - Controlling minimum and maximum values that can be stored in a variable
  - Increasing efficiency in memory utilization
  - Managing peripheral registers
- To increase software portability, fixed-size integer types can be defined in a header file using the *typedef* keyword

Fixed-size integers (2)

- **C99** update of the **ISO C standard** defines a set of standard names for signed and unsigned fixed-size integer types
  - 8-bit: int8_t, uint8_t
  - 16-bit: int16_t, uint16_t
  - 32-bit: int32_t, uint32_t
  - 64-bit: int64_t, uint64_t
- These types are defined in the library header file **stdint.h**

Fixed-size integers (3)

- Altera HAL provides the header file **alt_types.h** with definition of fixed-size integer types:

typedef signed char **alt_8**;

typedef unsigned char **alt_u8**;

typedef signed short **alt_16**;

typedef unsigned short **alt_u16**;

typedef signed long **alt_32**;

typedef unsigned long **alt_u32**;

typedef long long **alt_64**;

typedef unsigned long long **alt_u64**;

Logical operators

- Integer data can be interpreted as **logical values** in conditions (if, while, ...) or in logical expressions:
  - **=0, FALSE**
  - **ANY OTHER VALUE, TRUE**
- **Logical operators:**

AND	**&&**
OR	**||**
NOT	**!**

- Integer data can store the result of a logical expressions: 1 (TRUE), 0 (FALSE)

Bitwise operators (1)

- Operate on the bits of the operand/s

AND	**&**
OR	**|**
XOR	**^**
NOT	**~**
SHIFT LEFT	**<<**
SHIFT RIGHT	**>>**

Shift operators

- ***A << n***
  - The result is the bits of *A* moved to the left by *n* positions and padded on the right with 0
  - It is equivalent to multiply *A* by 2^n if the result can be represented
- ***A >> n***
  - The result is the bits of *A* moved to the right by *n* positions and padded on the left with 0 if type of *A* is unsigned or with the MSB of *A* if type is signed
  - It is equivalent to divide *A* by 2^n

Bit manipulation (1)

- **<<** and **|** operands can be used to create expressive binary constants by specifying the positions of the bits equal to 1
  - E.g. (1**<<**7) | (1**<<**5) | (1**<<**0) = 0xA1 (10100001)
  - Better not to use “magic numbers” as 7, 5 and 0. Use instead symbolic names to specify bit positions
    - For instance, the symbolic names can reflect the function of the bit within a peripheral register
  - (1<<X) can be encapsulated into a macro:
    - #define BIT(X) (1<<(X))

Bit manipulations (2)

- Altering only the bits in given positions
  - E.g. bits: 7, 5, 0
  - *#define MSK* = BIT(7) | BIT(5) | BIT(0)
- Clearing bits
  - *A* &= ~*MSK*;
- Setting bits
  - *A* |= *MSK*;
- Toggling bits
  - *A* ^= *MSK*;

Bit manipulations (3)

- Testing bits
  - E.g. do something if bit 0 (LSB) of *A* is set, regardeless of the other bits of *A*
  - *if* (A & BIT(0)) {
    /* some code here */
    }

Accessing Memory-mapped regs

- E.g. PIO peripheral (full set of regs)

Offset	Register Name		R/W	Fields				
				(n-1)	...	2	1	0
0	data	read access	R	Data value currently on PIO inputs.				
		write access	W	New value to drive on PIO outputs.				
1	direction *(1)*		R/W	Individual direction control for each I/O port. A value of 0 sets the direction to input; 1 sets the direction to output.				
2	interruptmask *(1)*		R/W	IRQ enable/disable for each input port. Setting a bit to 1 enables interrupts for the corresponding port.				
3	edgecapture *(1)*, *(2)*		R/W	Edge detection for each input port.				
4	outset		W	Specifies which bit of the output port to set.				
5	outclear		W	Specifies which output bit to clear.				

- We can define a C struct that overlays the peripherals regs

C struct overlay (1)

```
typedef struct {
        uint32_t data;                  / *offset 0 */
        uint32_t direction;             / *offset 4 */
        uint32_t int_mask;              / *offset 8 */
        uint32_t edge_capture;          / *offset 12 */
        uint32_t outset;                / *offset 16 */
        uint32_t outclear;              / *offset 20 */
 } volatile pio_t;

/* Define a pointer to MyPio PIO peripheral */
#define MY_PIO_BASE_ADDRESS    0x10000000 /* Base address of MyPio */

pio_t *pMyPio = (pio_t *) MY_PIO_BASE_ADDRESS;
```

C struct overlay (2)

```
/* Setting bit 7 without altering the other bits */
pMyPio->outset = BIT(7);

/* Clearing bit 3 without altering the other bits */
pMyPio->outclear = BIT(3);

/* Do something if bit 5 of the edge_capture reg is set*/
if(pMyPio->edge_capture & BIT(5)) {
   /* Some code here */
}
```

- What about?

```
/* Setting bit 7 without altering the other bits */
pMyPio->data |= BIT(7);

/* Clearing bit 3 without altering the other bits */
pMyPio->data &= BIT(3);
```