

Bit Twiddling



Topics

- 1) What are the bitwise operators?
- 2) What is a bit flags and masks?
- 3) How to:
 - a) Read / set single bits.
 - b) Read / set multiple bits.
- 4) Can C access bits better than just bitwise?

Bitwise and Bitmasks

- Bitwise operators
 - `|` is OR - Set selected bits
 - `&` is AND - ..
 - `~` is NOT - Invert all bits
 - `^` is XOR - Invert selected bits.
- Bit Flags
 - Store multiple binary conditions in a multi-bit value.
 - Ex: encoding the state of 8 LEDs in one char.
- Mask
 - Used to..
 - Has all 1's for bits of that field; 0 elsewhere.

Running Example

- STAT: GPIO Status Reg

Bit	15	14	13	12	11	10	9	8
	LED3	LED2	LED1	LED0	BTN3	BTN2	BTN1	BTN0
R or W	R/W	R/W	R/W	R/W	R	R	R	R

Bit	7	6	5	4	3	2	1	0
	SPD2	SPD1	SPD0	-	-	-	-	FLASH
R or W	R/W	R/W	R/W	R	R	R	R	R/W

- LEDx: Set (1) when on
- BUTTONx: Read 0 when pressed; 1 otherwise.
- SPD2-0: Flash speed; between 0 (slow) and 7 (fast)
- FLASH: 1 means flashing; 0 means solid (on).

Running Example

Bit	15	14	13	12	11	10	9	8
	LED3	LED2	LED1	LED0	BTN3	BTN2	BTN1	BTN0

Bit	7	6	5	4	3	2	1	0
-----	---	---	---	---	---	---	---	---

SPD2	SPD1	SPD0	-	-	-	-	FLASH
------	------	------	---	---	---	---	-------

- What does this value mean? 0xC2A7

BIT Numbers and Masks

- Bit Numbers
 - #define LED3_BIT 15
#define LED2_BIT 14
...
 - #define BTN3_BIT 11
...
 - #define SPD2_BIT 7
#define SPD1_BIT 6
#define SPD0_BIT 5
 - #define FLASH_BIT 0
- Convert Bit Number to Mask
 - #define LED0_MASK (1 << LED0_BIT)

Reading a Bit

- Read an LED State
 - `bool isLed0On = ..`
- Read a Button State
 - `bool isBtn0Pressed = ..`
- As Macros
 - `#define IS_LED_ON(pin) \`
 `((STAT & (1 << (pin))) != 0)`
 - `#define IS_BUTTON_PRESSED(pin) \`
 `((STAT & (1 << (pin))) == 0)`

Reading Bits

- Read Multiple Bits
 - `#define LED_MASK 0xF000;`
 - `bool isAnyLEDOn = ..`
 - `bool areAllLEDsOn = ..`
- Read Multiple Active-Low Bits
 - `#define BTN_MASK 0x0F00`
 - `bool isAnyButtonPressed = ..`
 - `bool areAllButtonsPressed =
(STAT & BTN_MASK) == 0;`

Drive Bits

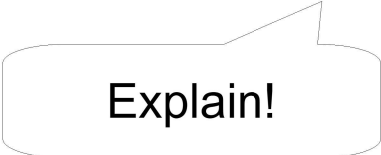
- Turn on LED 2
STAT..
- Turn off LED 2
STAT..
- Turn off LEDs 1 and 2
STAT &= ~(1<<LED2_BIT | 1<<LED1_BIT);
- Turn on / off all LEDs
STAT |= LED_MASK;
STAT &= ~LED_MASK;
- Turn off all LEDs but LED2 (leave it)
STAT..

Toggle Bits

- // Toggle LED0:
STAT
- // Toggle all LEDs:
STAT ^= LED_MASK;

Multi-Bit Fields

- Read value
 - #define SPD_MASK 0x00E0
 - int speed =
- Set value
 - void setFlashSpeed(int speed) {
 int newSpeed = (speed << SPD0_BIT)
 & SPD_MASK;
 STAT = (STAT & ~SPD_MASK) | newSpeed;
}



Explain!

Common Errors

- `~` vs `!`, `&` vs `&&`, `|` vs `||`
- `&=` vs `&= ~(..)`
- bit # vs mask: `LED1_BIT` vs `(1<<LED1_BIT)`

- use `(1<<x)` not `pow(2,x)`
- use `(1<<x) | (1<<y)`, not `1 << (x | y)`
- `b &= ~(1<<x)` is not `b = ~(1<<x)`

Real World Example: ATMEL CAN128

8-bit Timer/Counter Register Description

Timer/Counter2 Control Register A– TCCR2A

Bit	7	6	5	4	3	2	1	0	
	FOC2A	WGM20	COM2A1	COM2A0	WGM21	CS22	CS21	CS20	TCCR2A
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – FOC2A: Force Output Compare A**

The FOC2A bit is only active when the WGM bits specify a non-PWM mode. However, for ensuring compatibility with future devices, this bit must be set to zero when TCCR2A is written when operating in PWM mode. When writing a logical one to the FOC2A bit, an immediate compare match is forced on the Waveform Generation unit. The OC2A output is changed according to its COM2A1:0 bits setting. Note that the FOC2A bit is implemented as a strobe. Therefore it is the value present in the COM2A1:0 bits that determines the effect of the forced compare.

A FOC2A strobe will not generate any interrupt, nor will it clear the timer in CTC mode using OCR2A as TOP.

The FOC2A bit is always read as zero.

- **Bit 6, 3 – WGM21:0: Waveform Generation Mode**

These bits control the counting sequence of the counter, the source for the maximum (TOP)

Harder Exercises

- Decrement the current speed (SPD) by 1. Don't decrement if already 0.
- Write a function to make it seem like an LED is bouncing back and forth.
- Write a function that does:
If button N is pressed, turn on LEDs 0 - N.

C-Bit Fields

C Bit-Fields

- Declare fields in a struct with sizes (# bits)
 - Compiler pushes fields together to conserve space.

- Ex:

Represent a colour with 8 bits each for red, green, blue; and 1 bit for transparent:

```
struct colour_s {  
    unsigned int red           : 8;  
    unsigned int green        : 8;  
    unsigned int blue         : 8;  
    unsigned int transparent   : 1;  
};
```

- Entire struct needs only one unsigned int (32-bits)

Bit-field Details

- Access fields by name:
 - `struct colour_s border = {0xff, 0xff, 0x00, 1};`
`printf("Red %d\n", border.red);`
 - `border.transparent = 1;`

When assigning a value, ensure you don't have more bits than expected

- **WARNING:**
The order the fields get packed..

- Is the first field in the LSB, or is the last field in the LSB?

Code is non-portable: Must retest on new hardware or compiler.

OK for platform specific hardware access; poor for applications needing cross-platform binary data compatibility

STAT Example

```
struct stat_s {  
    unsigned int flash: 1;  
    unsigned int      : 4; // Unused bits ..  
    unsigned int spd  : 3;  
    unsigned int btn   : 4;  
    unsigned int led   : 4;  
    ..  
};
```

Unnamed fields take up unused space to line fields up as required

Must test to ensure fields don't need to be..

```
#define STAT_ADDR 0xC800153C  
struct stat_s *pSTAT = (struct stat_s *) STAT_ADDR;
```

```
int main() {  
    pSTAT->flash = 1;  
    if (pSTAT->btn == 0x0F) {  
        pSTAT->spd += 2;  
    }  
    pSTAT->led = pSTAT->btn;  
    return 0;  
}
```