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**

<table>
<thead>
<tr>
<th>Bit</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R or W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R or W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
</tr>
</tbody>
</table>

- **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

<table>
<thead>
<tr>
<th>Bit</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>LED3</td>
<td>LED2</td>
<td>LED1</td>
<td>LED0</td>
<td>BTN3</td>
<td>BTN2</td>
<td>BTN1</td>
<td>BTN0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPD2</td>
<td>SPD1</td>
<td>SPD0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>FLASH</td>
<td></td>
</tr>
</tbody>
</table>

- 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`
  ```c
  int speed =
  ```

- **Set value**
  ```c
  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)
- (a & ~b) is not (~a & b)
Real World Example: ATMEL CAN128

8-bit Timer/Counter Register Description

Timer/Counter2 Control Register A– TCCR2A

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **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:
  ```c
  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 that 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;
};

#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;
}