Microcontrollers: Programming and Interfacing

Stephen Jenks
EE, CpE, and CSE Senior Design Lecture
October 3, 2008
Overview

- What is a microcontroller?
- What is it used for?
- How do I program it?
- Can it talk to other devices?
- What do I need to program it?
- Debugging?
## Microprocessor vs. Microcontroller

<table>
<thead>
<tr>
<th>Difference</th>
<th>Microprocessor</th>
<th>Microcontroller</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>2 GHz</td>
<td>1-8 MHz</td>
</tr>
<tr>
<td>Data</td>
<td>32 or 64 bits</td>
<td>8 or 16 bits</td>
</tr>
<tr>
<td>Memory</td>
<td>4 MB Cache</td>
<td>1 KB RAM, 16 KB Flash</td>
</tr>
<tr>
<td>Power</td>
<td>100 Watts</td>
<td>&lt; 1mWatt at 1 MHz</td>
</tr>
</tbody>
</table>

### Diagram

- **Microcontroller**
  - Serial Peripheral Interface (SPI)
  - Serial Port
  - Analog Input
  - A/D
  - FLASH
  - RAM

- **North Bridge**
  - Digital I/O

- **Peripherals (PCI)**
  - Digital I/O
  - Digital I/O

- **Graphics (PCIe)**
  - Digital I/O

- **Microprocessor**
  - Serial Peripheral Interface (SPI)
  - Serial Port
  - Analog Input
  - A/D
  - FLASH
  - RAM
  - Memory
Microcontroller Uses

- Sensing
  - Analog signals (Built in A/D)
  - Digital signals (rates, RPM, position Gray code)
  - Input switch settings
- Actuation
  - Digital outputs (on/off, pulse width modulation)
  - LCD
- Communications
  - Peripheral devices
  - Wireless
  - Host computer
I/O pin programming

- I/O pins have 3 states
  - Input (read)
  - Output (write)
  - Tri-state (Hi-Z)
- Must set state
- Then read/write
- Snooping allowed
- Registers: (x is port)
  - DDRx – Data Direction
  - PORTx – Tri-state/pull-up/output
  - PINx – Pin state

unsigned char i;
...
/* Define pull-ups and set outputs high */
/* Define directions for port pins */
PORTB = (1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0);
DDRB = (1<<DDB3)|(1<<DDB2)|
       (1<<DDB1)|(1<<DDB0);
/* Insert nop for synchronization*/
___no_operation();
/* Read port pins */
i = PINB;

(From Atmel AVR ATMega169 Guide)
Sets pins 0-3 of Port B as output, rest as input,
Then sets pullups on pins 0 and 1, output of 1 on
pins 6 and 7. Pins 4 and 5 are tri-state.
int ADC_read(void) 
{
    char i;
    int ADC_temp;
    int ADCr = 0;

    // To save power, the voltage over the LDR and the NTC is turned off when not used
    // This is done by controlling the voltage from a I/O-pin (PORTF3)
    sbiBF(PORTF, PF3);  // Enable the VCP (VC-peripheral)
    sbiBF(DDRF, DDF3);

    sbiBF(ADCSRA, ADEN);  // Enable the ADC

    //do a dummy readout first
    ADCSRA |= (1<<ADSC);  // do single conversion
    while(!(ADCSRA & 0x10));  // wait for conversion done, ADIF flag active
... continued
```c
for(i=0;i<8;i++)          // do the ADC conversion 8 times for better accuracy
{
    ADCSRA |= (1<<ADSC);    // do single conversion
    while(!(ADCSRA & 0x10));  // wait for conversion done, ADIF flag active

    ADC_temp = ADCL;        // read out ADCL register
    ADC_temp += (ADCH << 8); // read out ADCH register

    ADCr += ADC_temp;      // accumulate result (8 samples) for later averaging
}
ADCr = ADCr >> 3;       // average the 8 samples

cbiBF(PORTF,PF3); // mt cbi(PORTF, PORTF3);  // disable the VCP
cbiBF(DDRF,DDF3); // mt cbi(DDRF, PORTF3);

cbiBF(ADCSRA, ADEN);    // disable the ADC
return ADCr;
}    /* From Atmel Sample code */
```
Serial Peripheral Interface
- Bidirectional serial link
- Can be fast (> 1Mbps)
- Master/slave roles
- Master-clocked

- Sends and receives bytes
- Requires setup (not shown)

```c
unsigned char SPI_Read()
{
    unsigned char input;
    // busy wait until SPIF bit set
    while (((SPSR & 0x80)==0) &&
           ((PINB & (1 << PB0)) == 0)) ;
    input = SPDR;
    return input;
}

void SPI_Write(const unsigned char c)
{
    SPDR = c;        // do the write
    // busy wait until SPIF bit set
    while ((SPSR & 0x80)==0) ;
}
```
void USART_Init(unsigned int baudrate)
{
    // Set baud rate
    UBRRH = (unsigned char)(baudrate>>8);
    UBRRL = (unsigned char)baudrate;

    // Enable 2x speed
    UCSRA = (1<<U2X);

    // Enable receiver. Transmitter stays disabled since
    UCSRB = (1<<RXEN)|(0<<TXEN)|(1<<RXCIE)|(0<<UDRIE);

    // Async. mode, 8N1
    UCSRC = (0<<UMSEL)|(0<<UPM0)|(0<<USBS)|
            (3<<UCSZ0)|(0<<UCPOL);
}

void Usart_Tx(char data)
{
    while (!(UCSRA & (1<<UDRE)));
    UDR = data;
}

char Usart_Rx(void)
{
    while (!(UCSRA & (1<<RXC)));
    return UDR;
}
Interrupts can be periodic or based on events

ISR(LCD_vect)
{
    unsigned char i;
    static int delay = 0;

    switch (currentState) {
    case SLEEP:
    case INIT:
        return;
    case NORMAL:
    case SPI_MASTER:
        for (i = 0; i < 6; i++) {
            ...
        }
    ...
Program on Host, Run on Target
- Write code in C or Assembly Language
- Toolset and compiler/assembler run on PC
- Download to Target via serial or USB

Target Environment
- No Operating System
- Boot loader in ROM
  - Jumps to program in FLASH upon startup
  - Loads programs from external port
Specific Programming Details

- Need Target-Specific toolkit
  - AVR Studio 4
  - Win-AVR (gcc, tools)
- Develop C code with AVR #includes
  - #defines all device and I/O registers, constants
  - ints are 16-bits, fakes longs, floats, etc.
  - No I/O (file, printf, etc.)
- Toolkit downloads compiled program to target
  - May need USB-to-Serial converter

October 3, 2008  Stephen Jenks  UCI EECS
Guidelines

- Not much memory
  - Static data structures (malloc may not be supported)
  - Avoid recursion unless you know depth is small
  - Process sensor data and discard it
- Not very powerful
  - Don’t use floating point (will be faked)
  - Don’t do expensive math (sin, cos) – use lookup tables instead
  - Probably can’t do FFTs or serious signal processing
- Battery powered
  - Sleep most of the time
  - Wake up via interrupts
    - On input change
    - Periodically
  - Avoid spinning
Debugging

- Very difficult to debug on target
  - Compile->download->test cycle takes a while
  - Limited interactivity
  - But for I/O, it may be required
  - Light LEDs, use LCD screen to show state
- Toolkit has simulator/source level debugger
  - Can step through code and watch I/O pins change
  - LCD module for Butterfly
  - All timing and interrupts out the window, but basic debugging works well.
For More Information

- AVR-GCC: www.avrfreaks.net/AVRGCC/
- Win-AVR: http://winavr.sourceforge.net/
- Arduino: http://www.arduino.cc/
- Spark Fun: http://www.sparkfun.com
- AVR Freaks: http://www.avrfreaks.net/