

EDL: Practical PIC Tutorial

Mike Chi
2007

Overview

1. Overview of PIC families
2. Hardware Design
3. Programming
4. System Example I: Simple LED Flasher
5. System Example II: Wireless Video Sensor

PIC Families

	PIC16	PIC18	PIC24/dsPIC	SX-28/48
Speed	20MHz 5 MIPS	40MHz 10 MIPS	80MHz 40 MIPS	75 MHz 75MIPS
Memory	256-4K Flash Program 32-256 bytes SRAM	64K Flash Program 4K SRAM	256K Flash Program 30K SRAM	4K Flash Program 262 bytes Banked Registers
Peripherals	ADC/UART/I2C/USB	ADC/USB/I2C/SPI/ UART/TCP/IP...etc	ADC/USB/I2C/SPI/ UART/TCP/IP/PWM/ AC97...etc	I/O Pins Only
Programming	ASM*	ASM/C I/O Libs	ASM/C I/O Libs, Signal Processing Libs	ASM*
Notes	8-bit	8-bit	16-bit dsPIC includes additional math capabilities (ie. single instruction MAC)	8-bit Very fast, Very Simple

*3rd Party C Compilers Exist

Useful Integrated Peripherals

- ADC - 10 or 12-bit up to 1MSPS
- Timer/Counter
- Analog Comparator
- UART
- General Purpose I/O Lines

PIC Architecture

- RISC Instruction Set
 - 8 and 16-bit
- Hardware memory mapped to registers
 - Device Configuration
 - Port/Peripheral/Communications

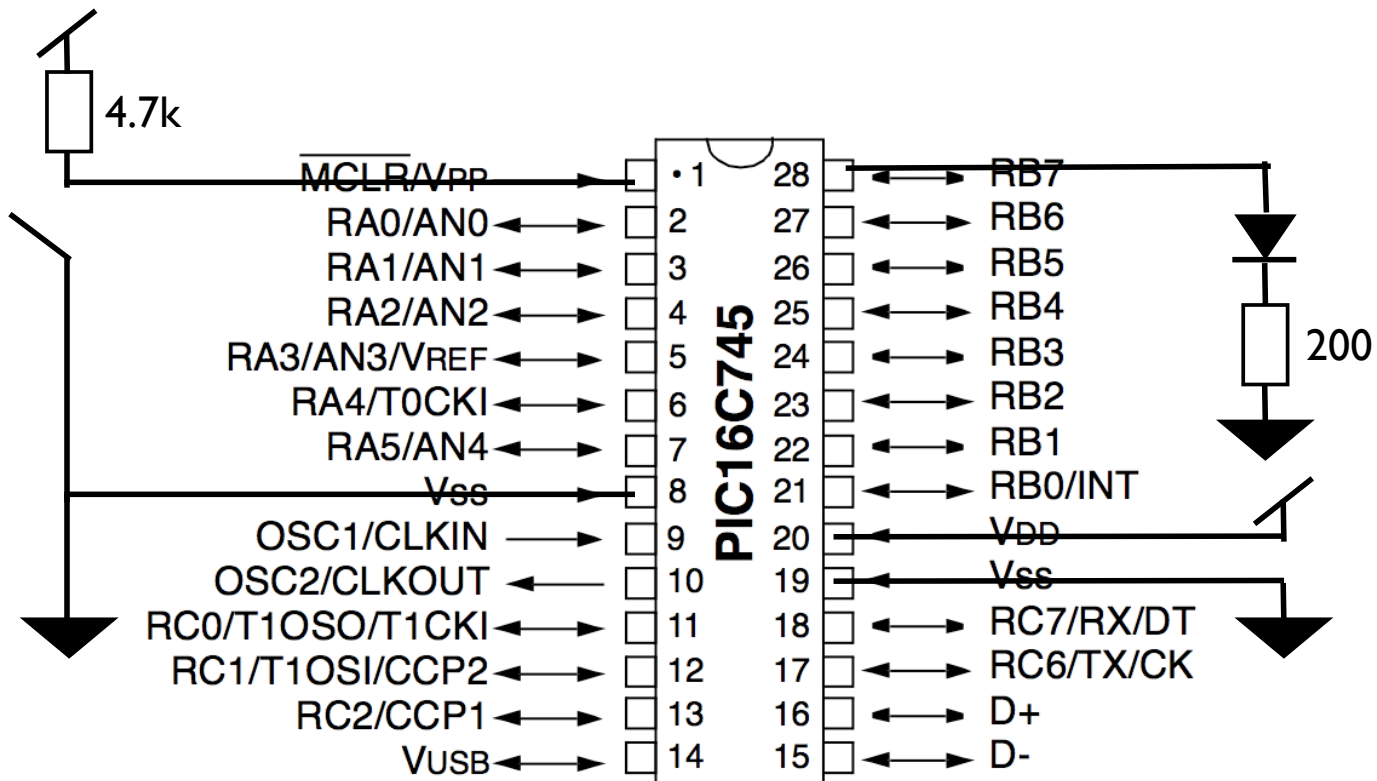
Tools You Need

- Circuit Board
 - Custom PCB
 - Demo Board
- MPLAB Programmer
- MPLAB Assembler / C Compiler
 - Downloadable

Clocking the PIC

- All PICs need oscillators to execute instructions
 - Internal (not available on all models) up to 8 MHz
 - External up to 40 MHz, 75 MHz on the SX
 - 16-Bit PICs have internal frequency multipliers and PLLs
- Instruction Cycle
 - PIC16/18 - 4 clocks/instruction
 - PIC24/dsPIC - 2 clocks/instruction
 - SX - 1 clock/instruction

Simple LED Flasher




```

;LED Flasher
LIST p=16C745      ;tell assembler what chip we are using
include "P16C745.inc"      ;include the defaults for the chip
__config 0x3D18      ;sets the configuration settings
                    ;(oscillator type etc.)

org 0x0000         ;org sets the origin, 0x0000 for the 16F628,
                    ;this is where the program starts running

movlw    0x07
movwf    CMCON      ;turn comparators off (make it like a 16F84)

bsf      STATUS, RP0      ;select bank 1
movlw    b'00000000'      ;set PortB all outputs
movwf    TRISB
movwf    TRISA          ;set PortA all outputs
bcf      STATUS, RP0      ;select bank 0

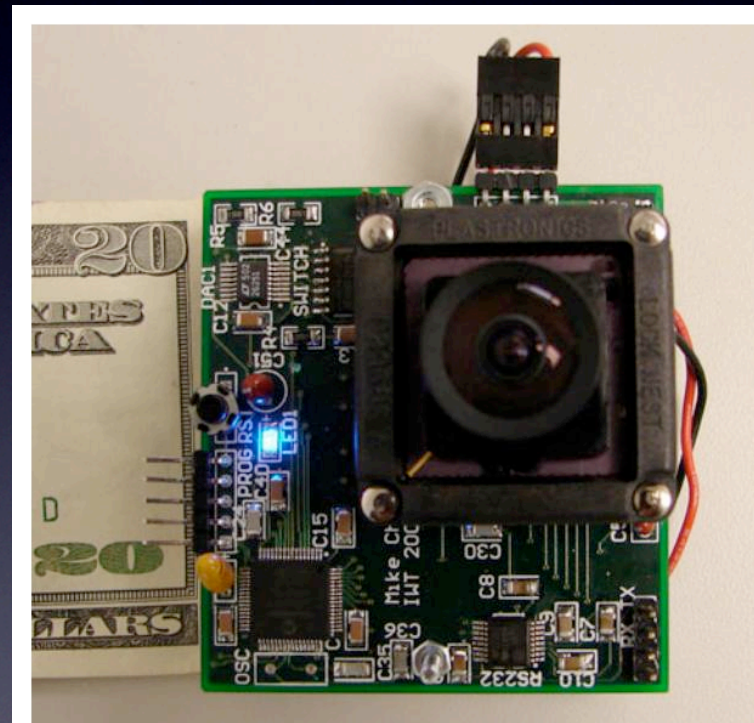
Loop
movlw    0xff
movwf    PORTA          ;set all bits on
movwf    PORTB
nop      ;the nop's make up the time taken by the goto
nop      ;giving a square wave output
movlw    0x00
movwf    PORTA
movwf    PORTB          ;set all bits off
goto     Loop           ;go back and do it again

end

```

Wireless Video Sensor

- Design Requirements
 1. Low power - 4 AA Batteries
 2. Compact - Size of 4 AA Batteries
 3. Low bandwidth - 1 Kb/sec
 4. Internal image processing/ compression



Specs

- dsPIC33
 - 40 MHz (20MIPS)
 - 16 KB RAM for 90x90 Image
 - DCT based image/video compression (Transform, Entropy Coding)
 - Image processing, gamma correction, histogram equalization
 - Internal ADC/UART
- MPLAB C30
 - Prototype algorithms on PC (gcc)
 - Easy integration with MPLAB C30 (also gcc based)

C Differences

- 100% ANSI C Compliant
 - even printf works - writes to UART1 (nice for debugging)
- Device headers include
 - Register names
 - Configuration bits
- Can use registers as 16-bit variables or by bit field
 - LATA = 0xFFFFF - writes to entire register
 - LATAbits.LATA5 = 1 - sets bit 6 of LATA high

Initialization - main.c

```
#include <p33J128GP706.h> //headers for dsPIC33J1280GP706
_FOSCSEL (FNOSC_FRCPLL);
_FOSC(FCKSM_CSDCMD & OSCIOFNC_OFF & POSCMD_NONE); //internal RC w/ PLL

int main()
{
    PLLFBD=18; //Configure multiplier
    CLKDIVbits.PLLPOST = 0; //Post-Scaler
    CLKDIVbits.PLLPRE = 0; //Pre-Scaler
    OSCTUN = 0; //Tune RC Oscillator

    while(OSCCONbits.LOCK!=1); //Wait for PLL to Lock

    while(1) //Run whatever it is I need
        run_camera();
}
```

Setup UART - comm.c

```
void init_uart()
{
    int uartConfig1 = UART_EN & UART_IDLE_CON & UART_IrDA_DISABLE & UART_MODE_SIMPLEX &
    UART_UEN_00 & UART_EN_WAKE & UART_DIS_LOOPBACK & UART_DIS_ABAUD & UART_UXRX_IDLE_ONE &
    UART_BRGH_FOUR & UART_NO_PAR_8BIT & UART_1STOPBIT;

    int uartConfig2 = UART_INT_TX & UART_SYNC_BREAK_DISABLED & UART_TX_ENABLE &
    UART_INT_RX_CHAR & UART_IrDA_POL_INV_ZERO;

    int uartBRG = 39;

    OpenUART1(uartConfig1, uartConfig2, uartBRG);
}

void write_packet(char* packet_p)
{
    int c;
    for(c=0; c<PACKET_SIZE; c++)
    {
        while(!U1STAbits.TRMT);
        U1TXREG = *packet_p++;
    }
}
```


Read Image - imager.c

```
#define LATBbits.LATB2 ROWCLK
#define LATBbits.LATB3 COLCLK
#define LATBbits.LATB4 RESET
unsigned char picture[XRES*YRES];

void image_out(unsigned char* pic)
{
    int y,x;
    for(y=0; y<YRES; y++)
    {
        for(x=0; x<XRES; x++)
        {
            AD1CON1bits.SAMP = 1;
            ROWCLK = 1;
            AD1CON1bits.SAMP = 0;
            while(!AD1CON1bits.DONE);
            *(pic++) = 255-ADCBUF0>>2;
            ROWCLK = 0;
        }
        RESET = 1;
        COLCLK = 1;
        delay_us(10);
        RESET = 0;
        COLCLK = 0;
    }
}
```

UART Interrupt

```
volatile int rcv_index=0;
unsigned char rcv[2];

void __attribute__((__interrupt__)) _U1RXInterrupt(void)
{
    unsigned char temp = U1RXREG;
    if(temp==0xFF)
    {
        config1 = rcv[0];
        config2 = rcv[1];
    }
    else
        rcv[index++] = temp;

    IFS0bits.U1RXIF = 0;
}
```

Some Advice

- Write to LAT registers not PORT
- Be careful with C optimization options
- Beware of I2C
- Using MPLAB C30?
 - 16-bit int, emulated floating point, limited dynamic memory allocation
- dsPIC33 peripheral libraries incomplete

Resources

- www.microchip.com
 - datasheets, C manual, tutorials, web seminars, MPLAB ASM, MPLAB C student edition
- www.parallax.com
 - free book on ASM programming, SX-Key assembler, virtual peripherals, SX BASIC (you should not be using this...)