PIC32 Configuration Bits

Configuration of Device Specific Features
• The PIC32MX795F512L microcontroller has many custom features.
  
  • These features can be either (1) enabled or disabled or (2) configured to suit particular customers.
  
  • For example:
    
    • Some companies would like to protect the code on the device when the device is deployed in the market.
    
    • This would prevent others (for example, competitors) from obtaining the source code by simply reading the machine code from the Flash memory and converting it to assembly language.
    
    • Other developers may not care if their code is readable.
As listed in [1], the PIC32MX795F512L supports the following special features (main features):

- Code Protection
- In-Circuit Debugger and Debugger Select
- Watchdog Timer (WDT)
- Oscillator Selection
- USB
- CAN
- Ethernet

Each of these features may be enabled/disabled or configured by writing to the respective bits of the configuration register (word) within the microcontroller.

In other words, the bits in the configuration word either enable or disable these features, or control how these feature operate (configured).

A possible advantage is that if a product does not use any of these special features, then they may be disabled to conserve energy.
The PIC32 microcontroller has an optional in-circuit debugger, which can be enabled or disabled.

The debugger circuit can be enabled by using C’s `#pragma` pre-processor directive:

```
#pragma config DEBUG = ON  // Debugger Enabled
```

This statement informs the compiler to enable the in-circuit debugger on the microcontroller.

The compiler enables the debugger by sending a configuration word to the microcontroller that has the Debug Configuration Bit set to ON.
MPLAB X communicates with the debugger through the USB interface to two alternate pairs of pins on the microcontroller.

- PGC1/PGD1 or PGC2/PGD2 (Default) Note: the pin names are mapped to pin numbers on the microcontroller (ProGram Code/ProGramDebug).
- The manufacturer of the MX7cK board choose to use the PGC1/PGD1 pair.
- Therefore, we need to inform the compiler to set the ICESEL Configuration Bits to PGC1/PGD1 pins for the debugging interface.

```
#pragma config ICESEL = ICS_PGX1  //In-Circuit Emulator SELect
```

- The debugger software in MPLAB X uses USB connector J15, labeled DEBUG to communicate with the in-circuit debugger within the microcontroller.
- Note: the PGC1/PGD1 pair of pins will also be used by MPLAB X to upload the machine code of a program to the microcontroller’s flash memory.
Clock Generation and Selection

- There are three main clocks in a PIC32 device
  - System Clock (SYSCLK), used by the CPU and some peripherals
  - Peripheral Bus Clock (PBCLK), used by most peripherals
  - USB Clock (USBCLK), used by the USB peripheral
- The PIC32 microcontroller supports numerous clock source options, from which the above three clocks will be generated:
  - EC External Clock
  - XT Crystal/Resonator
  - HS High-Speed Crystal/Resonator
  - HS, Xt, and EC with PLL
- The MX7cK board has implemented only two of these: EC and XT.
- Clock configuration for our board is an external 8 MHz silicon resonator:
  #pragma config POSCMOD=EC
Clock Speed Selection

• Once again, the three main clocks in a PIC32 device are:
  • System Clock (SYSCLK), used by the CPU and some peripherals
  • Peripheral Bus Clock (PBCLK), used by most peripherals
  • USB Clock (USBCLK), used by the USB peripheral

• The following example will set up the Cerebot MX7cK for operation with a SYSCLK frequency of 80Mhz and a PBCLK frequency of 10Mhz:

```c
#pragma config FNOSC = PRIPLL //Primary Oscillator with PLL
#pragma config POSCMOD = EC
#pragma config FPLLIDIV = DIV_2
#pragma config FPLLMUL = MUL_20
#pragma config FPLLODIV = DIV_1
#pragma config FPBDIV = DIV_8 //Peripheral clock speed
```
The PIC32MX795 microcontroller contains a 10/100 Ethernet Medium Access Controller (MAC), which is housed within the microcontroller chip.

External to the PIC32 microcontroller, the Cerebot MX7cK board provides an SMSC LAN8720 Ethernet Physical Layer Transceiver (PHY) chip.

Together, the MAC and PHY in combination with an appropriate coupling transformer and RJ45 jack provide a standard 10/100 hardware Ethernet interface.

The PIC32MX795 microcontroller provides two alternate sets of pins that can be used to connect the MAC to the external PHY, and two alternate standard MAC/PHY interface signaling conventions.

The Cerebot MX7cK is designed to use the standard (not the alternate) pins, and to use the RMII (not the MII) interface signaling convention.

```
#pragma config FETHIO=ON    //standard pins
#pragma config FMIIEN=OFF   //RMII signalling
```
Example Configuration (See [2])

/* Oscillator Settings*/
#pragma config FNOSC = PRIPLL // Oscillator selection
#pragma config POSCMOD = EC // Primary oscillator mode
#pragma config FPLLDIV = DIV_2 // PLL input divider
#pragma config FPLLMUL = MUL_20 // PLL multiplier
#pragma config FPLLODIV = DIV_1 // PLL output divider
#pragma config FPBDIV = DIV_8 // Peripheral bus clock divider
#pragma config FSOSCE = OFF // Secondary oscillator enable
/* Clock control settings*/
#pragma config IESO = OFF // Internal/external clock switchover
#pragma config FCKSM = CSDCMD // Clock switching (CSx)/Clock monitor (CMx)
#pragma config OSCIOFNC = OFF // Clock output on OSCO pin enable

/* USB Settings */
#pragma config UPLLEN = ON // USB PLL enable
#pragma config UPLLIDIV = DIV_2 // USB PLL input divider
#pragma config FVBUSONIO = OFF // VBUS pin control
#pragma config FUSBIDIO = OFF // USBID pin control
/* Other Peripheral Device settings*/
#pragma config FWDTEN = OFF // Watchdog timer enable
#pragma config WDTPS = PS1024 // Watchdog timer post-scalar
#pragma config FSRSEL = PRIORITY_7 // SRS interrupt priority
#pragma config FCANIO = OFF // Standard/alternate CAN pin select
#pragma config FETHIO = ON // Standard/alternate ETH pin select
#pragma config FMIEN = OFF // MII/RMII select (OFF=RMII)
/* Code Protection settings*/
#pragma config CP = OFF // Code protection
#pragma config BWP = OFF // Boot flash write protect
#pragma config PWP = OFF // Program flash write protect
/* Debug settings*/
#pragma config ICESEL = ICS_PGx1 // ICE pin selection
Config. Bits Implementation

- The configuration is written to a special memory location called the configuration word (CW) (Three registers, CW1, CW2, and CW3) (See [1])
  - The CW causes the hardware to configure according to the bits in the configuration word
  - Thus, these features may be enabled or disabled
We as programmers inform the compiler how we want certain custom features to operate by specifying the configuration bits settings in #pragma statements.

In turn, the compiler sends the configuration bits settings to the microcontroller, and they get written to the configuration word registers.

The bits of the configuration bits registers are connected to the associated custom hardware features to either enable or disable them, or to control how they operate.
Security

- Access to the configuration word registers must be protected

  - If access was not protected, then code may be written to change the settings of the CW and subvert the intentions of the original developers

- Access can be protected by:
  1. Placing the CWs in a memory location that is not accessible by any code while it is running on the device.
  2. Allowing the CW to be set only during when the code is initially written into the device (i.e., when the device is programmed).

Programs can only access this area in memory when they are running
CW Programming Methodology

• If access to CW is protected, then how do we specify and set the configuration word?

• Answer:
  • We specify the CW in our source code of our project, using a special compiler directive (#pragma for XC32)
  • The compiler takes our specification of the CW and encodes it into the stream of codes to be programmed into the device
  • When the IDE uploads the machine code onto the PIC32, the CW is written to the protected memory. This is the only time that the CW may be written.
The XC32 compiler supports the ANSI standard “pragma” directive to allow developers to specify the configuration bits settings.

- See “C:\Program Files (x86)\Microchip\xc32\v1.21\docs\MPLAB-XC32-Users-Guide.pdf”
- See “C:\Program Files (x86)\Microchip\xc32\v1.21\docs\PIC32ConfigSet.pdf”
References
