Unit 1: Introduction

Introduction:-

In 1989, Microchip Technology Corporation introduced an 8-bit microcontroller called the PIC, which stands for Peripheral Interface Controller. This microcontroller had small amounts of data RAM, a few hundred bytes of on-chip ROM for the program, one timer, and a few pins for I/O ports, all on a single chip with only 8 pins. (See Figure 1-2.) It is amazing that a company that began with such a humble product became one of the leading suppliers of 8-bit microcontrollers in less than a decade. At the time of this writing, Microchip is the number-one supplier of 8-bit microcontrollers in the world. Since the introduction of the PIC16xxx, they have introduced an array of 8-bit microcontrollers too numerous to list here. They include the PIC families of 10xxx, 12xxx, 14xxx, 16xxx, 17xxx, and 18xxx. They are all 8-bit processors, meaning that the CPU can work on only 8 bits of data at a time. Data larger than 8 bits has to be broken into 8-bit pieces to be processed by the CPU. One of the problems with the PIC family is that they are not all 100% upwardly compatible in terms of software when going from one family to another family. For example, while the 12xxx/16xxx have 12-bit and 14-bit wide instructions, the PIC18xxx instruction is 16 bits wide with many new instructions. To run programs written for the PIC12xxx on a PIC18, we must recompile the program and possibly change some register locations before loading it into the PIC18. At the time of this writing, the PIC18xxx family has the highest performance of all the families of 8-bit PIC microcontrollers. The fact that PIC18xxx is available in 18- to 80-pin packages makes it an ideal choice for new designs because it allows an easy migration to more powerful versions of the chip without losing software compatibility. At this time, no 8-pin version of the PIC18xxx exists, and that is the main reason to choose other family members of the 10xxx–16xxx if your design calls for a small package. Because this book is about the PIC18 family, we describe some of the main features of this family and refer the reader to the Microchip web site for other families of PIC10xxx–16xxx. For those who have mastered the PIC18 family, understanding the other families is very easy and straightforward. The following is a brief description of the PIC18 series.

PIC18 features

The PIC18 has a RISC architecture that comes with some standard features such as on-chip program (code) ROM, data RAM, data EEPROM, timers, ADC, and USART and I/O ports. See Figure 1-2. Although the size of the program ROM, data RAM, data EEPROM, and I/O ports varies among the family members, they all have peripherals such as timers, ADC, and USART. See Figures 1-3 and 1-4. Due to the importance of these peripherals, we have dedicated an entire chapter to each one of them. The details of the RAM/ROM memory and I/O features of the PIC18 are given in the next few chapters.

Watchdog Timer (WDT)

The Watchdog Timer is a free running, on-chip RC oscillator which does not require any external components. This RC oscillator is separate from the RC oscillator of the OSC1/CLKI pin. That means that the WDT will run, even if the clock on the OSC1/CLKI and OSC2/CLKO/RA6 pins of the device has been stopped.

During normal operation, a WDT time-out generates a device Reset (Watchdog Timer Reset). If the device is in Sleep mode, a WDT time-out causes the device to wake-up and continue with normal operation (Watchdog Timer wake-up). The TO bit in the RCON register will be cleared upon a WDT time-out.

The Watchdog Timer is enabled/disabled by a device configuration bit. If the WDT is enabled, software execution may not disable this function. When the WDTEN configuration bit is cleared, the SWDTEN bit enables/disables the operation of the WDT.

WDTCON: WATCHDOG TIMER CONTROL REGISTER

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-0
—	—	—	—	—	—	—	SWDTEN
bit 7							bit 0

bit 7-1 **Unimplemented**: Read as ‘0’

bit 0 **SWDTEN:** Software Controlled Watchdog Timer Enable bit
1 = Watchdog Timer is on
0 = Watchdog Timer is turned off if the WDTEN configuration bit in the Configuration register = 0

Brown-out Reset (BOR):-

A configuration bit, BOREN, can disable (if clear/ programmed), or enable (if set), the Brown-out Reset circuitry. If VDD falls below parameter D005 for greater than parameter #35, the brown-out situation resets the chip. A Reset may not occur if VDD falls below parameter D005 for less than parameter #35. The chip will remain in Brown-out Reset until VDD rises above BVDD. The Power-up Timer will then be invoked and will keep the chip in Reset an additional time delay (parameter #33). If VDD drops below BVDD while the Power-up Timer is running, the chip will go back into a Brown-out Reset and the Power-up Timer will be initialized. Once VDD rises above BVDD, the Power-up Timer will execute the additional time delay.

In-Circuit Serial Programming (ISP):-

PIC18FXXX microcontrollers can be serially programmed while in the end application circuit. This is simply done with two lines for clock and data and three other lines for power, ground and the programming voltage. This allows customers to manufacture boards with unprogrammed devices and then program the microcontroller just before shipping the product. This also allows the most recent firmware or a custom firmware to be programmed.

I²C:-

I²C (pronounced I-squared-C) created by Philips Semiconductors and commonly written as 'I2C' stands for Inter-Integrated Circuit and allows communication of data between I2C devices over two wires. It sends information serially using one line for data (SDA) and one for clock (SCL).

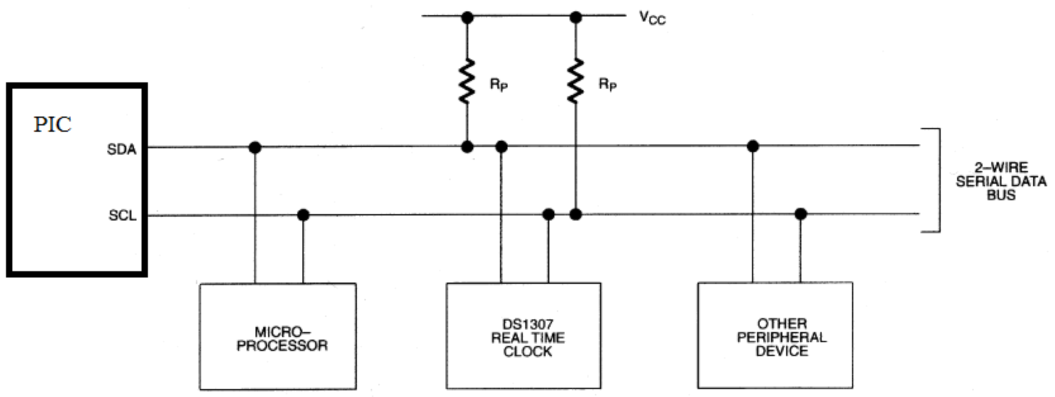
Master and slave

The phillips I2C protocol defines the concept of master and slave devices. A master device is simply the device that is in charge of the bus at the present time and this device controls the clock and generates START and STOP signals. Slaves simply listen to the bus and act on controls and data that they are sent.

The master can send data to a slave or receive data from a slave - slaves do not transfer data between themselves.

Data and Clock

The I2C interface uses two bi-directional lines meaning that any device could drive either line. In a single master system the master device drives the clock most of the time - the master is in charge of the clock but slaves can influence it to slow it down.



Speed

Standard clock speeds are 100kHz and 10kHz but the standard lets you use clock speeds from zero to 100kHz and a fast mode is also available (400kHz - Fast-mode).
Note that the low-speed mode has been omitted (10kHz) as the standard now specifies the basic system operating from 0 to 100kHz.

Device addresses

Each device you use on the I2C bus must have a unique address. For some devices e.g. serial memory you can set the lower address bits using input pins on the device others have a fixed internal address setting e.g. a real time clock DS1307. You can put several memory devices on the same IC bus by using a different address for each.

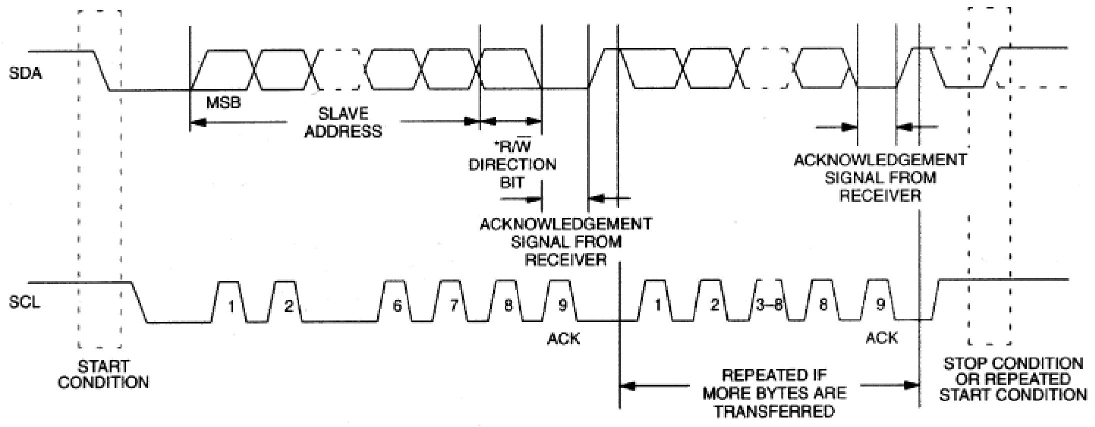
Start data transfer: A change in the state of the data line, from HIGH to LOW, while the clock is HIGH, defines a START condition.

Stop data transfer: A change in the state of the data line, from LOW to HIGH, while the clock line is HIGH, defines the STOP condition.

Data valid: The state of the data line represents valid data when, after a START condition, the data line is stable for the duration of the HIGH period of the clock signal. The data on the line must be changed during the LOW period of the clock signal. There is one clock pulse per bit of data.
Each data transfer is initiated with a START condition and terminated with a STOP condition. The number of data bytes transferred between START and STOP conditions is not limited, and is determined by the master device. The information is transferred byte-wise and each receiver acknowledges with a ninth bit.

Acknowledge: Each receiving device, when addressed, is obliged to generate an acknowledge after the reception of each byte. The master device must generate an extra clock pulse which is associated with this acknowledge bit. A device that acknowledges must pull down the SDA line during the acknowledge clock pulse in such a way that the SDA line is stable LOW during the HIGH period of the acknowledge related clock pulse. Of course, setup and hold times must be taken into account. A master must signal an end of data to the slave by not generating an acknowledge bit on the last byte that has been clocked out of the slave. In this case, the slave must leave the data line HIGH to enable the master to generate the STOP condition.

DATA TRANSFER ON 2-WIRE SERIAL BUS


SDA
MSB
SLAVE ADDRESS
*R/W DIRECTION BIT
ACKNOWLEDGEMENT SIGNAL FROM RECEIVER
ACKNOWLEDGEMENT SIGNAL FROM RECEIVER
SCL
1
2
6
7
8
9
ACK
1
2
3–8
8
9
ACK
START CONDITION
REPEATED IF MORE BYTES ARE TRANSFERRED
STOP CONDITION OR REPEATED START CONDITION

Depending upon the state of the R/W bit, two types of data transfer are possible:

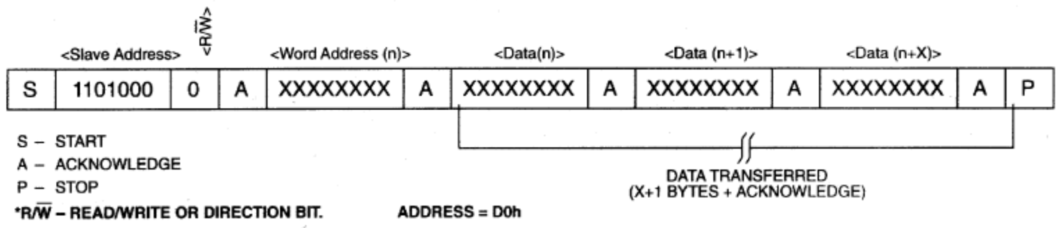
1. **Data transfer from a master transmitter to a slave receiver.** The first byte transmitted by the master is the slave address. Next follows a number of data bytes. The slave returns an acknowledge bit after each received byte. Data is transferred with the most significant bit (MSB) first.
2. **Data transfer from a slave transmitter to a master receiver.** The first byte (the slave address) is transmitted by the master. The slave then returns an acknowledge bit. This is followed by the slave transmitting a number of data bytes. The master returns an acknowledge bit after all received bytes other than the last byte. At the end of the last received byte, a 'not acknowledge' is returned.

The master device generates all of the serial clock pulses and the START and STOP conditions. A transfer is ended with a STOP condition or with a repeated START condition. Since a repeated START condition is also the beginning of the next serial transfer, the bus will not be released. Data is transferred with the most significant bit (MSB) first.

1. T**ransmitter mode:** Serial data and clock are received through SDA and SCL. After each byte is received an acknowledge bit is transmitted. START and STOP conditions are recognized as the beginning and end of a serial transfer. Address recognition is performed by hardware after reception of the slave address and direction bit. The address byte is the first byte received after the start condition is generated by the master. The address byte contains the 7 bit address of slave, followed by the direction bit (R/W) which, for a write, is a 0. After receiving and decoding the address byte the slave device outputs an acknowledge on the SDA

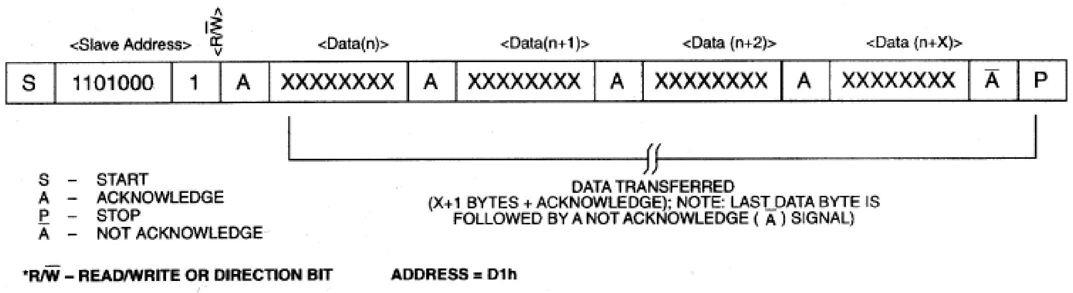
line. After the acknowledges the slave address + write bit, the master transmits a register address to the slave. This will set the register pointer on the slave. The master will then begin transmitting each byte of data with the slave acknowledging each byte received. The master will generate a stop condition to terminate the data write.

DATA WRITE - SLAVE RECEIVER MODE



2. **Receiver mode:** The first byte is received and handled as in the slave receiver mode. However, in this mode, the direction bit will indicate that the transfer direction is reversed. Serial data is transmitted on SDA by the slave while the serial clock is input on SCL. START and STOP conditions are recognized as the beginning and end of a serial transfer. The address byte is the first byte received after the start condition is generated by the master. The address byte contains the 7-bit slave address, followed by the direction bit (R/W) which, for a read, is a 1. After receiving and decoding the address byte the device inputs an acknowledge on the SDA line. The slave then begins to transmit data starting with the register address pointed to by the register pointer. If the register pointer is not written to before the initiation of a read mode the first address that is read is the last one stored in the register pointer. The slave must receive a Not Acknowledge to end a read.

DATA READ - SLAVE TRANSMITTER MODE



SPI:-

The SPI bus was originally started by Motorola Corp. (now Freescale), but in recent years has become a widely used standard adapted by many semiconductor chip companies. SPI devices use only 2 pins for data transfer, called SDI (Din) and SDO (Dout), instead of the 8 or more pins used in traditional buses. This reduction of data pins reduces the package size and power consumption drastically, making them ideal for many applications in which space is a major concern. The SPI bus has the SCLK (shift clock) pin to synchronize the data transfer between two chips. The last pin of the SPI bus is CE (chip enable), which is used to initiate and terminate the data transfer. These four pins, SDI, SDO, SCLK, and CE, make the SPI a 4-wire interface. See Figure 16-1. There is also a widely used standard called a 3-wire interface bus. In a 3-wire interface bus, we have SCLK and CE, and only a single pin for data transfer. The SPI 4-wire bus can become a 3-wire interface when the SDI and SDO data pins are tied together. However, there are some major differences between the SPI and 3-wire devices in the data transfer protocol. For that reason, a device must support the 3-wire protocol internally in order to be used as a 3-wire device. Many devices such as the DS1306 RTC (real-time clock) support both SPI and 3-wire protocols.

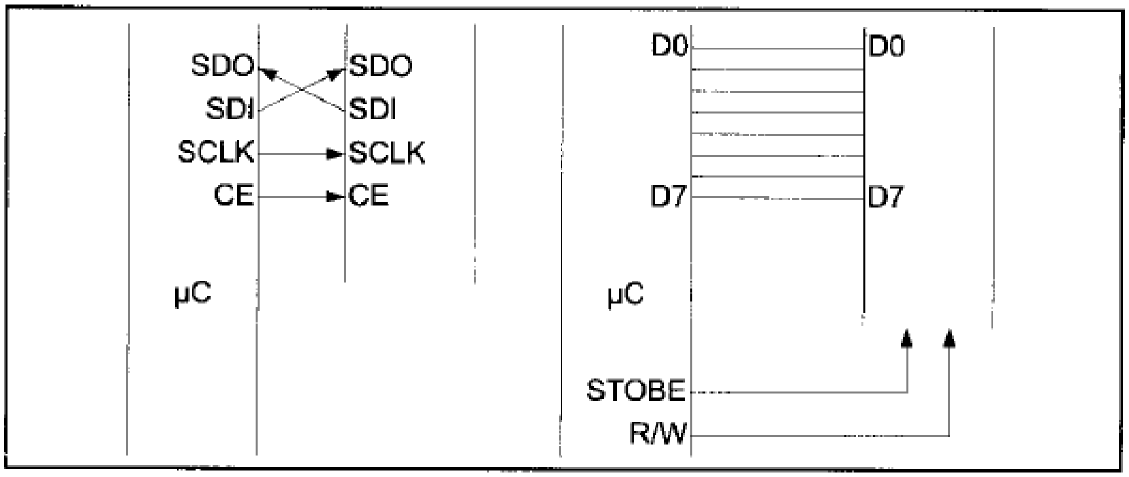


Figure 16-1. SPI Bus vs. Traditional Parallel Bus Connection to Microcontroller

Harvard vs. Von-Neumann Architecture

Every microprocessor must have memory space to store program (code) and data. As we have seen so far, the PIC is no exception with its code ROM space and data RAM (file register) space. While code provides instructions to the CPU, the data provides the information to be processed. The CPU uses buses (wire traces) to access the code ROM and data RAM memory spaces. The early computers used the same bus for accessing both the code and data. Such an architecture is commonly referred to as von Neumann (Princeton) architecture. That means for von Neumann computers, the process of accessing the code or data could cause them to get in each other's way and slow down the processing speed of the CPU, because each had to wait for the other to finish fetching. To speed up the process of program execution, some CPUs use what is called Harvard architecture. In Harvard architecture, we have separate buses for the code and data memory. That means that we need four sets of buses: (1) a set of data buses for carrying data into and out of the CPU, (2) a set of address buses for accessing the data, (3) a set of data buses for carrying code into the CPU, and (4) an address bus for accessing the code.

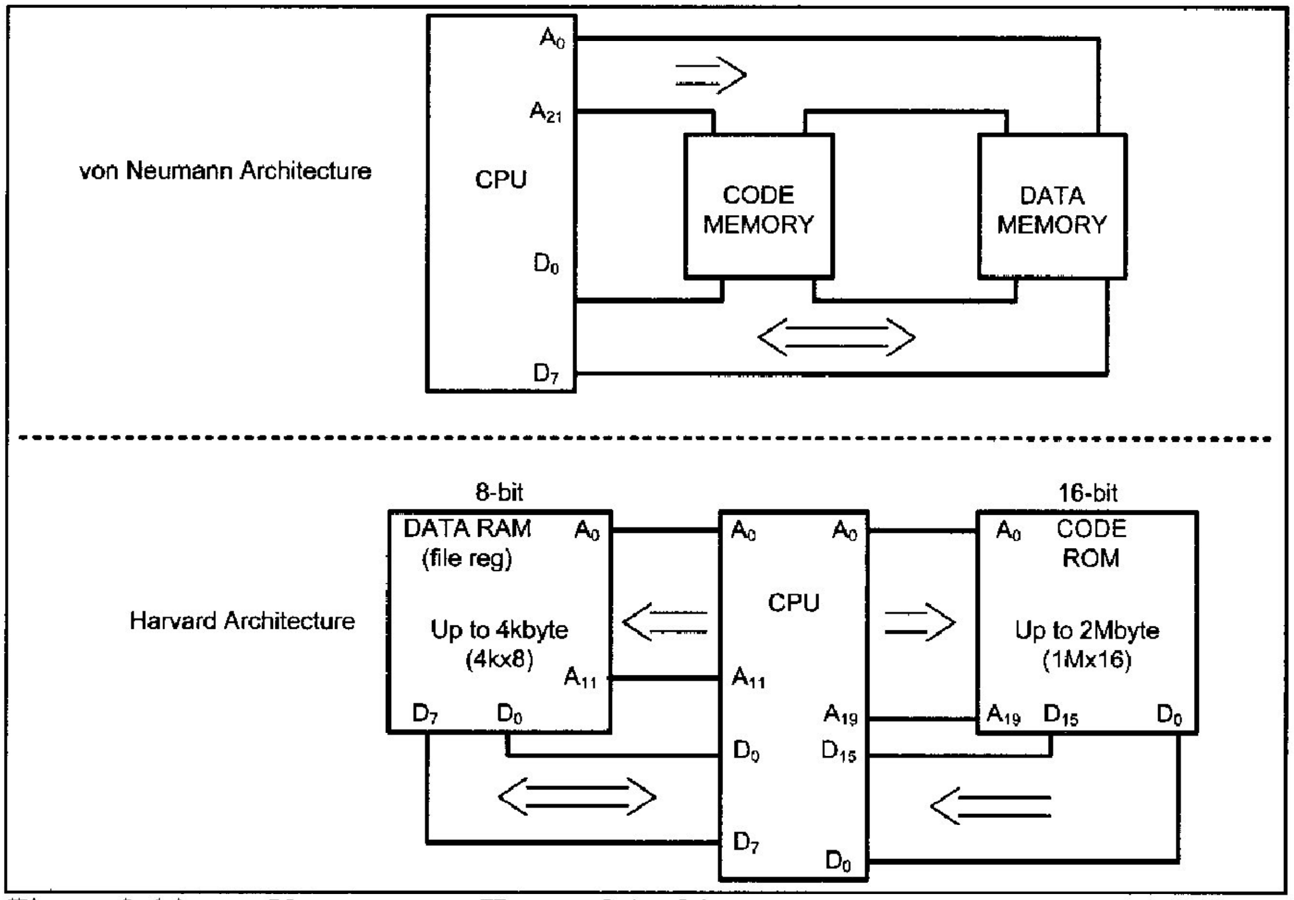


Figure 2-14. von Neumann vs. Harvard Architecture

CISC and RISC Architecture

In the early 1980s, a controversy broke out in the computer design community, but unlike most controversies, it did not go away. Since the 1960s, in all mainframes and minicomputers, designers put as many instructions as they could think of into the CPU. Some of these instructions performed complex tasks. An example is adding data memory locations and storing the sum into memory. Naturally, microprocessor designers followed the lead of minicomputer and mainframe designers. Because these microprocessors used such a large number of instructions and many of them performed highly complex activities, they came to be known as CISC (complex instruction set computer). According to several studies in the 1970s, many of these complex instructions etched into the brain of the CPU were never used by programmers and compilers. The huge cost of implementing a large number of instructions (some of them complex) into the microprocessor, plus the fact that a good portion of the transistors on the chip are used by the instruction decoder, made some designers think of simplifying and reducing the number of instructions. As this concept developed, the resulting processors came to be known as RISC (reduced instruction set computer).

RISC processors have a fixed instruction size. In a CISC microcontroller such as the 8051, instructions can be 1, 2, or even 3 bytes.

This variable instruction size makes the task of the instruction decoder very difficult because the size of the incoming instruction is never known. In a RISC architecture, the size of all instructions is fixed. Therefore, the CPU can decode the instructions quickly.

One of the major characteristics of RISC architecture is a large number of registers. All RISC architectures have at least 32 registers. Of these 32 registers, only a few are assigned to a dedicated function. One advantage of a large number of registers is that it avoids the need for a large stack to store parameters. Although a stack can be implemented on a RISC processor, it is not as essential as in CISC because so many registers are available. In the PIC microcontrollers the use of a 256-byte bank for the file register satisfies this RISC feature.

WREG Register

In the CPU, registers are used to store information temporarily. That information could be a byte of data to be processed, or an address pointing to the data to be fetched. The vast majority of PIC registers are 8-bit registers. In the PIC there is only one data type: 8-bit. The 8 bits of a register are shown in the diagram below. These range from the MSB (most-significant bit) D7 to the LSB (least-significant bit) D0. With an 8-bit data type, any data larger than 8 bits must be broken into 8-bit chunks before it is processed.

D7	D6	D5	D4	D3	D2	D1	D0

The 8-bit WREG register is the most widely used register in the PIC microcontroller. WREG stands for working register, as there is only one. The WREG register is the same as the accumulator in other microprocessors. The WREG register is used for all arithmetic and logic instructions. To understand the use of the WREG register, we will show it in the context of two simple instructions: MOVE and ADD.

THE PIC FILE REGISTER

The PIC microcontroller has many other registers in addition to the WREG register. They are called data memory space to distinguish them from program (code) memory space. The data memory space in PIC is a read/write (static RAM) memory. In the PIC microcontroller literature, the data memory is also called the *file register.*

File register (data RAM) space allocation in PIC

The file register is read/write memory used by the CPU for data storage, scratch pad, and registers for internal use and functions. As with WREG, we can perform arithmetic and logic operations on many locations of the file register data RAM. The PIC microcontrollers' file register size ranges from 32 bytes to several thousand bytes depending on the chip. Even within the same family, the size of the file register data RAM varies from chip to chip. Notice that the file register data RAM has a byte-size width, just like WREG. The file register data RAM in PIC is divided into two sections: (a) Special Function Registers (SFR), and (b) General-Purpose Registers (GPR). The general-purpose register section is also referred to as General-Purpose RAM (GP RAM). We examine each section separately.

SFRs (Special Function Registers)

The Special Function Registers (SFRs) are dedicated to specific functions such as ALU status, timers, serial communication, I/O ports, ADC, and so on. The function of each SFR is fixed by the CPU designer at the time of design because it is used for control of the microcontroller or peripheral. The PIC SFRs are 8-bit registers. The number of locations in the file register set aside for SFR depends on the pin numbers and peripheral functions supported by that chip. That number can vary from chip to chip even among members of the same family. Some have as few as 7 (8-pin PIC12C508 with no on-chip analog-to-digital converter) and some have over a hundred (40-pin PIC18F458 with on-chip analog-to digital converter). For example, the more timers we have in a PIC chip, the more SFR registers we will have. We will study and use many SFRs in future chapters.

GPR (General-Purpose Registers or RAM)

The general-purpose registers are a group of RAM locations in the file register that are used for data storage and scratch pad. Each location is 8 bits wide and can be used to store any data we want as long as it is 8-bit. Again, the number of RAM locations in the file register that are set aside for general-purpose registers can vary from chip to chip, even among members of the same family. In the PIC controllers, the space that is not allocated to the SFRs typically is used for general-purpose registers. That means in a PIC chip with a thousand-byte file register, no more than 100 bytes are used for SFRs and the rest are used for general-purpose registers. A larger GPR size means more difficulties in managing these registers if you use Assembly language programming. In today's high-performance microcontroller, however, with over a thousand bytes of GPR, the job of managing them is handled by the C compilers. Indeed, the C compilers are the very reason we need a large GPR since it makes it easier for C compilers to store parameters and perform their jobs much faster. See Table 2-1 for a comparison of file registers among various PIC chips. Also see Figure 2-2.

Table 2-1: File Register Size for PIC Chips

	File Register (Bytes)	=	SFR (Bytes)	+	Available space for GPR (Bytes)
PIC12F508	32		7		25
PIC16F84	80		12		68
PIC18F1220	512		256		256
PIC18F452	1792		256		1536
PIC18F2220	768		256		512
PIC18F458	1792		256		1536
PIC18F8722	4096		158		3938

GP RAM vs. EEPROM in PIC chips

Note that there are two RAM columns in the chip information section of the Microchip web site. One refers to the general-purpose registers' (GP RAM) size, and the other is the EEPROM size. GP RAM (which constitutes most of the file register) must not be confused with the EEPROM data memory. The GPRs are used by the CPU for internal data storage, whereas the EEPROMs are considered as an add-on memory that one can also add externally to the chip. In other words, while many PIC chips have zero bytes of EEPROM data memory, it is impossible for a microcontroller to have zero size for the file register. The EEPROM memory of PIC chips is covered in Chapter 14.

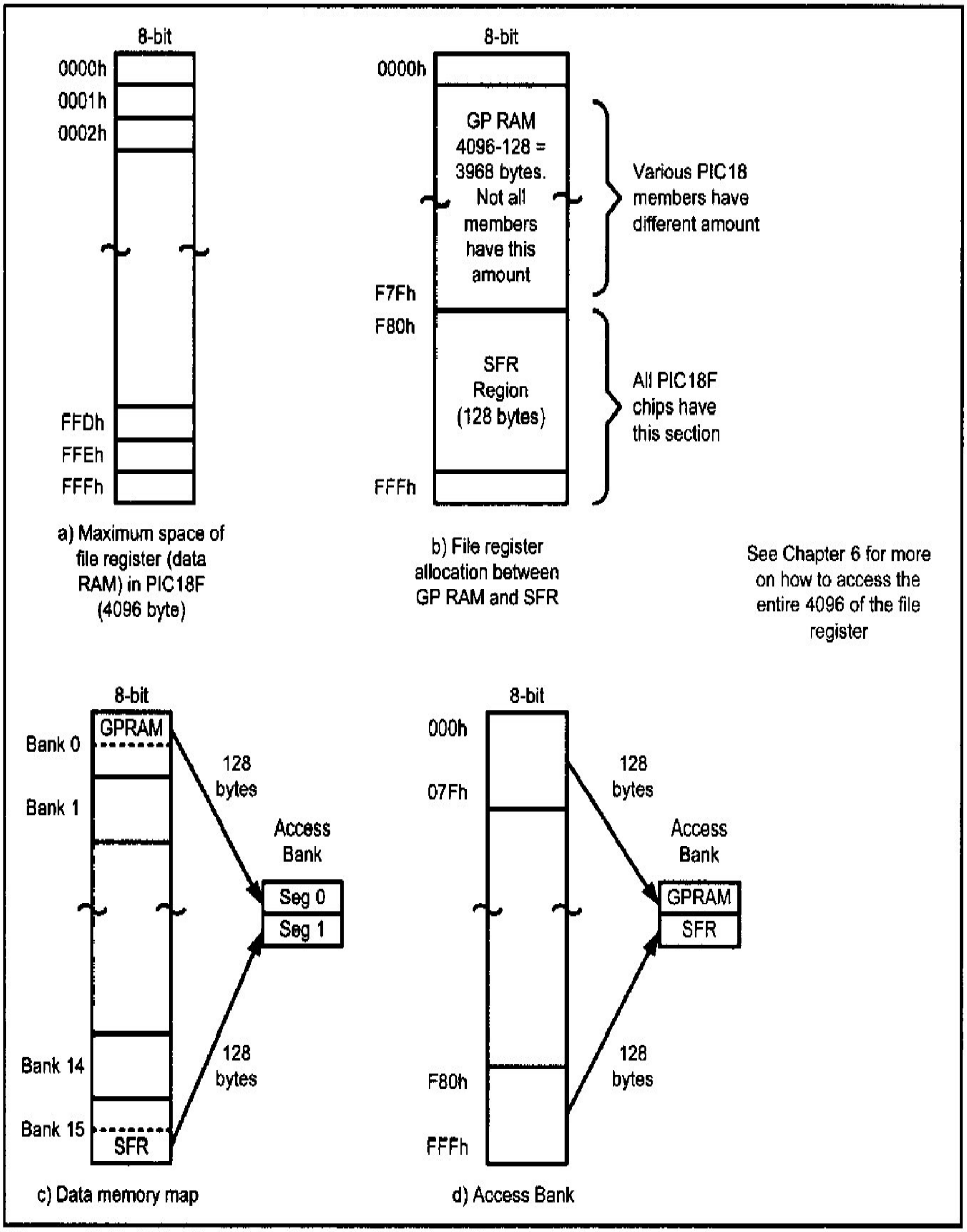

8-bit
0000h
0001h
0002h
FFDh
FFEh
FFFh
a) Maximum space of file register (data RAM) in PIC18F (4096 byte)
8-bit
0000h
GP RAM 4096-128 = 3968 bytes. Not all members have this amount
F7Fh
F80h
SFR Region (128 bytes)
FFFh
Various PIC18 members have different amount
All PIC18F chips have this section
b) File register allocation between GP RAM and SFR
See Chapter 6 for more on how to access the entire 4096 of the file register
8-bit
GPRAM
Bank 0
Bank 1
Bank 14
Bank 15
SFR
128 bytes
128 bytes
Access Bank
Seg 0
Seg 1
c) Data memory map
8-bit
000h
07Fh
F80h
FFFh
128 bytes
128 bytes
Access Bank
GPRAM
SFR
d) Access Bank

Figure 2-3. File Register for PIC18 Family

File register and access bank in the PIC18

The file register of the PIC18 family can have a maximum of 4096 (4K) bytes. With 4096 bytes, the file register has addresses of 000–FFFH. The file register in the PIC18 is divided into 256-byte banks. Therefore, we can have up to a maximum of 16 banks ($16 \times 256 = 4096$). Although not all members of the PIC18 family have that many banks, every PIC18 family member has at least one bank for the file register. This bank is called the *access bank* and is the default bank when we power up the PIC18 chip. To simplify the discussion of how to use the file register in the PIC family, we focus on this single bank that is found in every member of the PIC18 family. You can examine the file registers in other PIC families such as PIC12 and PIC16 at the Microchip website. In this book we concentrate on the PIC18 series with their large file register, although the insight gained in the process can be applied to the PIC16 and PIC12 series.

Examine the access bank for the PIC18 in Figure 2-3. The 256-byte access bank is divided into two equal sections of 128 bytes. These 128-byte sections are given to the general-purpose registers and special function registers. The 128 bytes from locations 00H to 7FH are set aside for general-purpose registers and are used for read/write storage, or what is normally called a *scratch pad.* These 128 locations of RAM are widely used for storing data and parameters by PIC18 programmers and C compilers. Each location of this 128-byte RAM of general-purpose registers can be accessed directly by its address. We will use these locations in future chapters to store data brought into the CPU via I/O and serial ports. We will also use them to define counters for time delay in Chapter 3. The other 128 bytes of the access bank is used for SFRs. It has addresses of F80H to FFFH, as shown in Figure 2-4. One might wonder why the memory space of the SFRs and GPRs in the access bank is not contiguous. The reason is to allow the RAM space between 080H and F7FH to be used for the general-purpose registers by various members of the PIC18 if they implement a larger data RAM size for the file register. A file register of more than 256 bytes will necessitate bank switching. *Bank switching* is a method used to access all the banks of the file register for PIC18 family members that have more than the minimum access bank. PIC18 members with a file register of more than 256 bytes will be discussed in more detail in Chapter 6 when we discuss bank switching.

PIC18 status register

The status register is an 8-bit register. It is also referred to as the *flag register*. Although the status register is 8 bits wide, only 5 bits of it are used by the PIC18. The three unused bits are unimplemented and read as 0. The five flags are called *conditional flags*, meaning that they indicate some conditions that result after an instruction is executed. These five flags are C (carry), DC (digital carry), Z (zero), OV (overflow), and N (negative). See Figure 2-7 for the bits of the status register. Each of the conditional flags can be used to perform a conditional branch (jump), as we will see in Chapter 3.

D7							D0
X	X	X	N	OV	Z	DC	C

C – Carry flag
DC – Digital Carry flag
Z – Zero flag
OV – Overflow flag
N – Negative flag
X – D5, D6, and D7 are not implemented, and reserved for future use.

Figure 2-7. Bits of Status Register

C, the carry flag

This flag is set whenever there is a carry out from the D7 bit. This flag bit is affected after an 8-bit addition or subtraction. Chapter 5 shows how the carry flag is used.

DC, the digital carry flag

If there is a carry from D3 to D4 during an ADD or SUB operation, this bit is set; otherwise, it is cleared. This flag bit is used by instructions that perform BCD (binary coded decimal) arithmetic. In some microprocessors this is called the AC flag (Auxiliary Carry flag). See Chapter 5 for more information.

Z, the zero flag

The zero flag reflects the result of an arithmetic or logic operation. If the result is zero, then Z = 1. Therefore, Z = 0 if the result is not zero. See Chapter 3 to see how we use the Z flag for looping.

OV, the overflow flag

This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit. In general, the carry flag is used to detect errors in unsigned arithmetic operations while the overflow flag is used to detect errors in signed arithmetic operations. The OV and N flag bits are used for the signed number arithmetic operations and are discussed in Chapter 5.

N, the negative flag

Binary representation of signed numbers uses D7 as the sign bit. The negative flag reflects the result of an arithmetic operation. If the D7 bit of the result is zero, then N = 0 and the result is positive. If the D7 bit is one, then N = 1 and the result is negative. The negative and OV flag bits are used for the signed number arithmetic operations and are discussed in Chapter 5.

PIC18F458/452 PIN CONNECTION

The PIC18F458 family members come in different packages, such as DIP (dual in-line package), QFP (quad flat package), and LLC (leadless chip carrier). They all have many pins that are dedicated to various functions such as I/O, ADC, timer, and interrupts. Note that Microchip provides an 18-pin version of the PIC18 family with a reduced number of I/O ports for less demanding applications. Because the vast majority of developers use the 40-pin chip, however, we will concentrate on that. Figure 8-1 shows the pins for the PIC18F458.

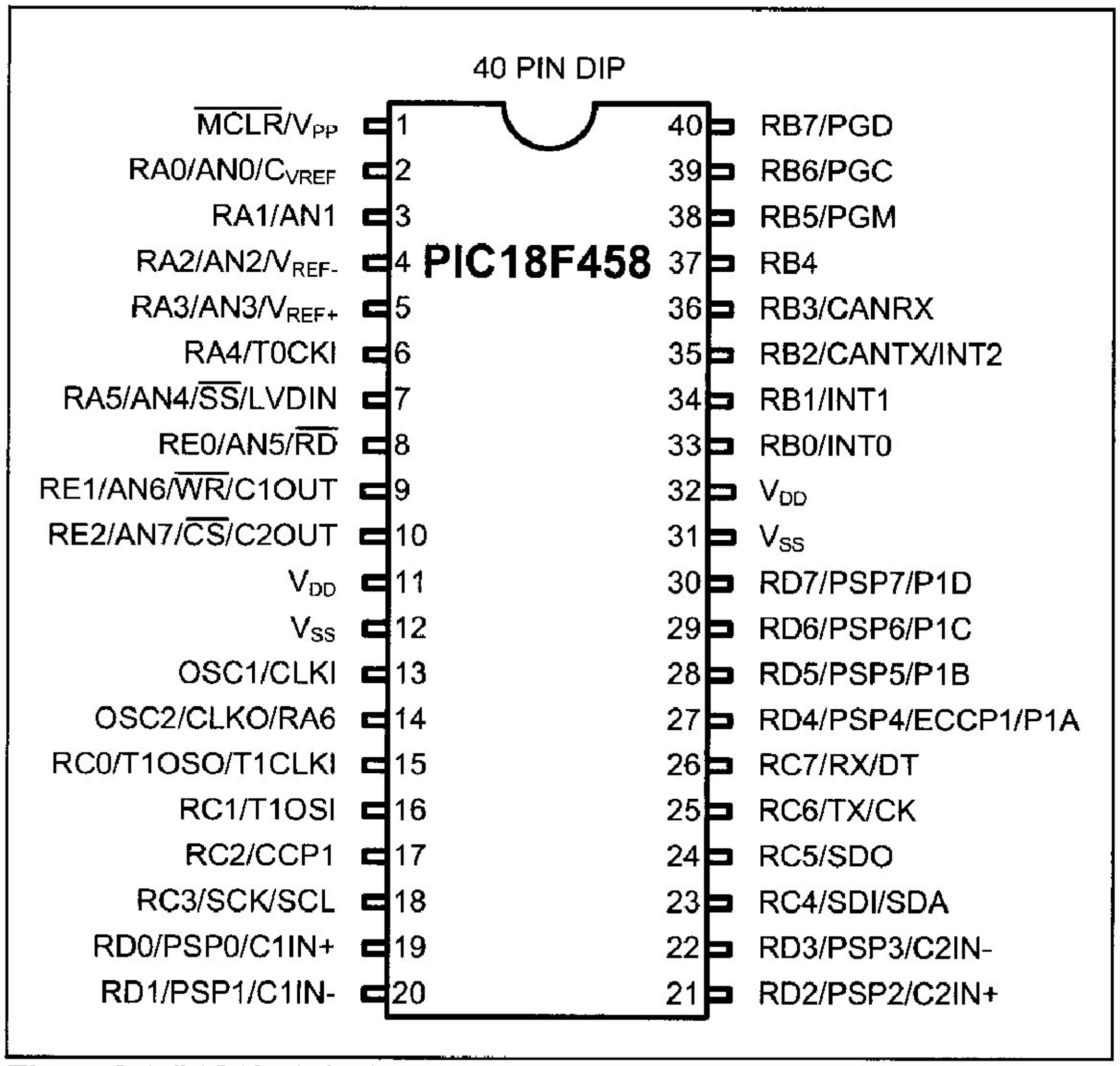


Figure 8-1. PIC18F458 Pin Diagram

Examining Figure 8-1, note that of the 40 pins, a total of 33 are set aside for the five ports A, B, C, D, and E, with their alternate functions. The rest of the pins are designated as Vdd, GND (Vss), OSC1, OSC2, and MCLR (master clear reset). Next, we describe the function of each pin.

Vdd (Vcc)

Two pins are used to provide supply voltage to the chip. The typical voltage source is +5V. Some PIC18F family members have lower voltage for Vdd pins in order to reduce the noise and power dissipation of the PIC system. We can choose other options for the Vdd voltage level by setting the bits in the configuration register. The configuration register for Vdd is discussed in the next section.

Vss (GND)

Two pins are also used for ground. In chips with 40 pins and more, it is common to have multiple pins for VCC and GND. This will help reduce the noise (ground bounce) in high-frequency systems, as discussed in Appendix C.

OSC1 and OSC2

The PIC18F has many options for the clock source. Most often a quartz crystal oscillator is connected to input pins OSC1 and OSC2. The quartz crystal oscillator connected to the OSC1 and OSC2 pins also needs two capacitors. One side of each capacitor is connected to the ground as shown in Figure 8-3. Note that PIC18F microcontrollers can have speeds of 0 Hz to 40 MHz.

MCLR

Pin 1 (in the PIC18F458 40-pin DIP) is the MCLR (master clear reset) pin. It is an input and is active-LOW (normally HIGH). When a LOW pulse is applied to this pin, the microcontroller will reset and terminate all activities. This is often referred to as a *power-on reset (POR).*

Program counter value upon reset

Activating a MCLR reset will cause all values in the registers to be lost. Table 8-1 provides a partial list of PIC18F registers and their values after power-on reset. From Table 8-1 we note that the value of the PC (program counter) is 0 upon reset, forcing the CPU to fetch the first opcode from ROM memory location 00000. This means that we must place the first byte of opcode in ROM location 0 because that is where the CPU expects to find the first instruction.

Table 8-1: RESET Values for Some PIC18 Registers

Register	Reset Value (hex)
PC	000000
WREG	00
SP	00
TRISA–TRISE	FF

Minimum connections:-

Figures 8-2a and 8-2b show two ways of connecting the MCLR pin to the power-on reset circuitry. Figure 8-2b uses a momentary switch for reset circuitry. The most difficult time for any system is during the power-up. The CPU needs both a stable clock source and a stable voltage level to function properly. The PIC18 chips come with some features that help the reset process. We can choose these features by setting the bits in the configuration register. The configuration register for the reset pin is discussed in the next section. There are other sources of reset in the PIC18 family, and they are discussed in future chapters.

The pins discussed so far must be connected no matter which family member is used. They are the minimum pin connections that every PIC18 must have. See Figure 8-3.

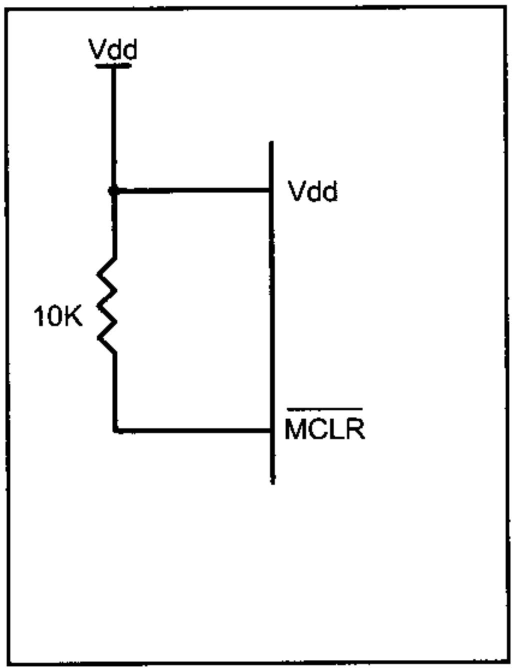


Figure 8-2a. PIC18F458 Power-On Reset Circuit

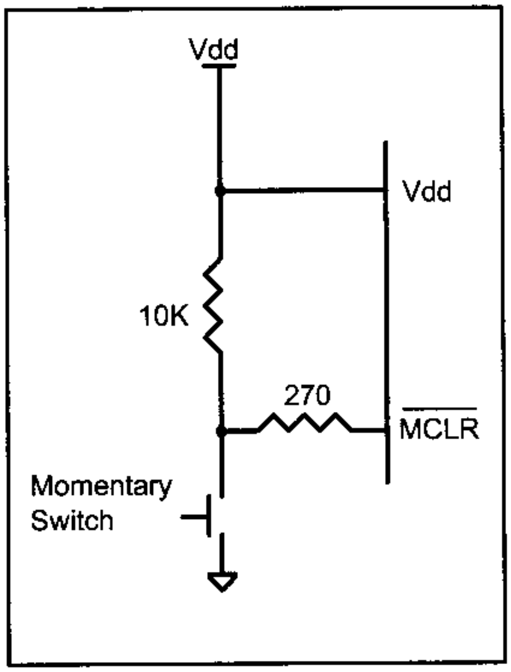


Figure 8-2b. PIC18F458 Power-On Reset with a Momentary Switch

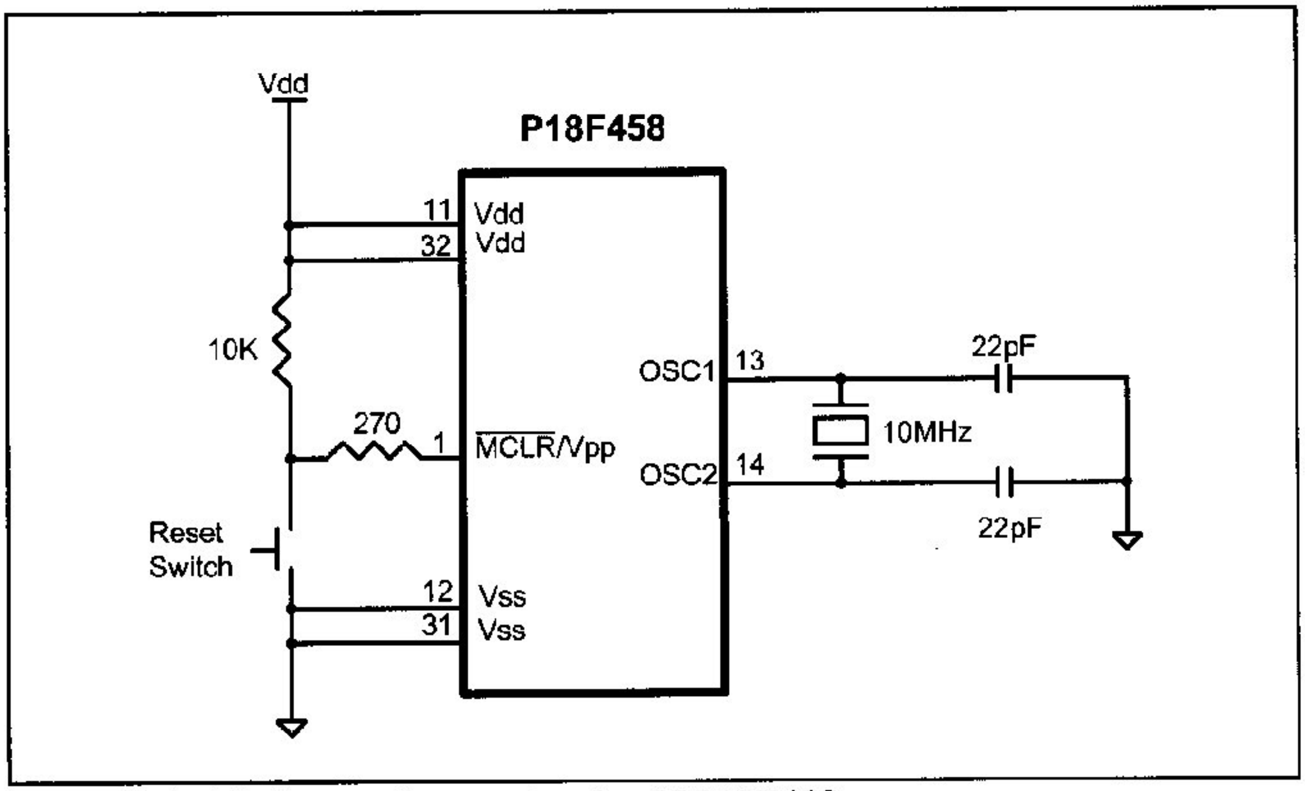


Figure 8-3. Minimum Connection for PIC18F458