Chapter 8
External World Interfaces

Once the basic interface of a microcontroller-based system is established, the next natural step is to use general purpose I/O (GPIO) pins to connect the MCU to the external world. This Chapter focuses on the tasks related to interfacing external devices to microcontrollers using general purpose I/O lines.

The first section provides a thorough discussion of GPIOs covering their structure, capabilities, configuration, and electrical characteristics. The section is crowned by a detailed coverage of GPIO characteristics in MSP430 generations.

The rest of the chapter deals with the interface of real world devices via GPIO. A thorough discussion of how to interface switches, displays, large DC and AC loads, and motors to microcontrollers is included, highlighting MSP430 supporting functions and capabilities, enhanced by hardware and software examples that illustrate their applicability.

8.1 General Purpose Input-Output Ports

One of the most valuable resources in microcontrollers are its general purpose Input/Output (GPIO) ports. GPIO ports or just I/Os provide MCUs with the ability of communicating with the external world via digital lines that can be configured to operate as either inputs or outputs. A GPIO port consists of a group of I/O lines that can be, independently or as a group, configured as inputs or as outputs. Most MCUs associate eight I/O lines to a port, although it is possible to find MCUs with ports of only four, six or other number of I/O lines in a port. GPIO lines become available to the external world through either dedicated or shared package pins. Figure 8.1 shows the package outline of an MSP430G2452IN20, denoting the pins where I/O lines become available. This particular MCU provides 16 I/O lines available as two 8-bit ports: Port 1 (P1) and Port 2 (P2). The individual lines in a port are denoted P*i*.*j*, where *i* denotes the port number and *j* represents the line number in the port,

DOI: 10.1007/978-1-4614-3143-5_8,

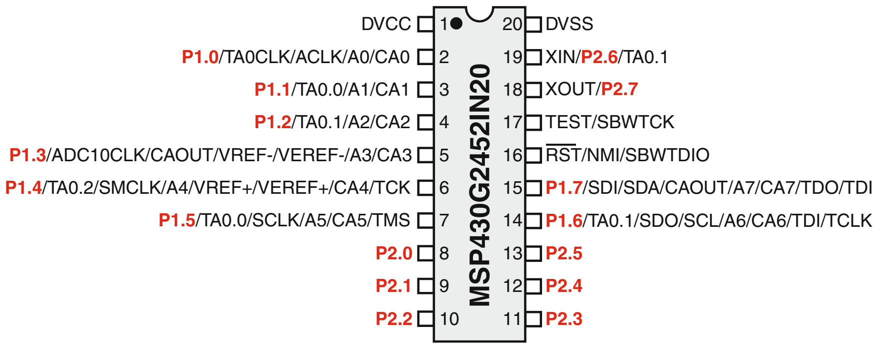

DVCC
P1.0/TA0CLK/ACLK/A0/CA0
P1.1/TA0.0/A1/CA1
P1.2/TA0.1/A2/CA2
P1.3/ADC10CLK/CAOUT/VREF-/VEREF-/A3/CA3
P1.4/TA0.2/SMCLK/A4/VREF+/VEREF+/CA4/TCK
P1.5/TA0.0/SCLK/A5/CA5/TMS
P2.0
P2.1
P2.2
MSP430G2452IN20
DVSS
XIN/P2.6/TA0.1
XOUT/P2.7
TEST/SBWTCK
RST/NMI/SBWTDIO
P1.7/SDI/SDA/CAOUT/A7/CA7/TDO/TDI
P1.6/TA0.1/SDO/SCL/A6/CA6/TDI/TCLK
P2.5
P2.4
P2.3

Fig. 8.1 Package outline of an MSP430g2452 denoting I/O pins

labeled from 0 to 7. This package also show how the I/O lines in P1 are all shared with other functions, while those in P2 have dedicated lines.

An I/O line configured as input allows the MCU to accept incoming information about the binary status of an external device connected to it. Whenever we need to learn about the status of an external event, device status, or any binary signal whose value can be represented through either a logic high or logic low level, input ports are the hardware resource of choice. Examples of such conditions include determining if a device is on or off, open or closed, up or down, left or right, set or clear, etc.

When configured as output, an I/O line provides the MCU with the ability of controlling the binary status of external devices or signals. Actions such as turning on or off an LED indicator, a buzzer, or anything that can be managed by setting an external voltage level to either a logic high or logic low can be done with an output I/O line.

8.1.1 Structure of an Input-Output Pin

To better understand how to configure an I/O port, it deserves to understand the structure of an I/O interface. Figure 8.2 shows a generic structure of an I/O pin driver.

To communicate the CPU with the external world using a digital I/O line, it is necessary an interface that enables connecting a physical I/O pin to the processor buses, with the ability of operating as either input or output. To perform an Input operation (In), the minimum requirement calls for an input buffer. Such a requirement is provided in Fig. 8.2 by the buffer labeled "*In*". The input buffer is typically a standard logic buffer with tri-state control. The tri-state capability allows for configuring the pin as either input or output while avoiding signal contention. Note the a logic "0" in the port direction latch will enable the *In* buffer, while disabling the "*Out*" buffer, making the pin operate as an input. Under this mode, the input data flows through the input buffer to the $P_{i,j}\_In$ line accessing the data bus.

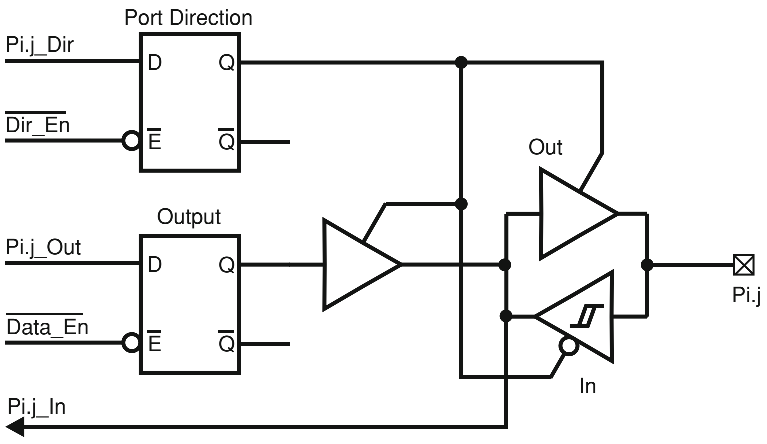


Fig. 8.2 Basic structure of an Input/Output pin driver

Sometimes, the signal being fed to an input port might have a slow changing behavior or noisy contents around the nominal logic levels. Either of these conditions will cause an undesired behavior in the digital side of the circuit. In such cases, a Schmitt-trigger input buffer becomes convenient. Section 8.2.2 on page 395 discusses the advantages of Schmitt-triger inputs. At this point it is enough to point out that this type of input buffer has the ability of speeding-up slow transitions and eliminating most noise from a digital input. The *In* buffer in Fig. 8.2 is of this type. The hysteresis loop inside the buffer symbol indicates it is a Schmitt-trigger capable input pin.

Contemporary microcontrollers such as the MSP430 typically provide Schmitt-trigger capable GPIO inputs in their ports, saving the designer the burden of externally providing them when working with slow or noisy signals.

An output transaction (Out) in the I/O pin would require the value written by the CPU through the $P_{i,j}\_Out$ data line to stay on the line until a later CPU write access changes the pin to a different value. This calls for a latched output. The latch labeled *Output* performs this function in Fig. 8.2. The output latch is double buffered to allow for being configured as either input or output. In particular, the I/O pin ability to source or sink current will be determined by the characteristics of the "Out" buffer.

The direction in which the pin is used, either as input or as output, is achieved by using tri-state buffers for the input and output functions, and connecting them to form a transceiver. The transceiver control is exerted through line $P_{i,j}\_Dir$, via a latch to allow for retaining the desired In or Out functionality. Additional logic not shown in Fig. 8.2 takes care of interfacing lines $P_{i,j}\_In$, $P_{i,j}\_Out$, and $P_{i,j}\_Dir$ to a single bidirectional data bus line. This logic would also provide for decoding the address, timing, and control lines to drive signals $\overline{Dir\_En}$ and $\overline{Data\_En}$ allowing for a physical connection to the processor buses. The configuration of such an interface logic will depend on the signal architecture of the CPU buses. Figure 8.3 shows a possible configuration assuming a Motorola style bus timing.

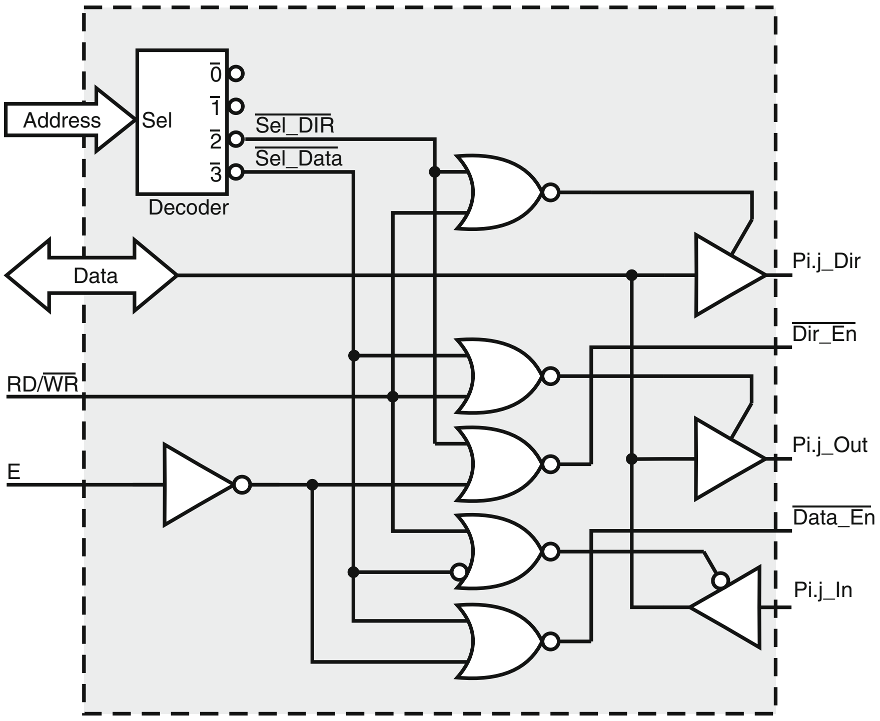


Fig. 8.3 Interface logic for an Input/Output pin assuming a Motorola style bus timing

The three tri-state buffers in the pin interface side allow accessing a bidirectional data bus line. The NOR gates decode control signals $RD/\overline{WR}$, E, and the decoder outputs provide unique access for the corresponding data lines. The decoder assigns unique addresses to the direction and data latches. Note that $P_{i,j}\_In$ and $P_{i,j}\_Out$ share the same address, the first as input to the CPU and the other as output, while $P_{i,j}\_Dir$ is a write-only location.

8.1.2 Configuring GPIO Ports

The insight gained into the structure of a bidirectional I/O pin in the previous section denotes that for its usage, we simply need to configure the pin direction (as either In or Out), and then access (read or write) its data contents. As MCUs group I/O pins into ports, all I/O lines in a given port are accessed at the same address location. The same rule applies for all direction control bits for the port, making it possible to use simple instructions to configure and access a port. Figure 8.4 illustrates a simple port model (Port.x) containing one data and one direction register.

Let’s use an example to illustrate how to read and write I/O pins in a GPIO port.

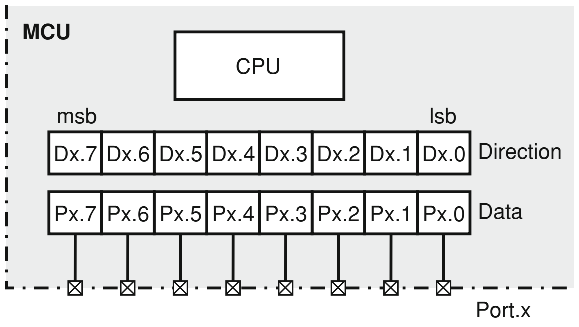

MCU
CPU
msb
lsb
Dx.7 Dx.6 Dx.5 Dx.4 Dx.3 Dx.2 Dx.1 Dx.0
Direction
Px.7 Px.6 Px.5 Px.4 Px.3 Px.2 Px.1 Px.0
Data
Port.x

Fig. 8.4 Programmer's model for a simple GPIO port containing only data and direction registers

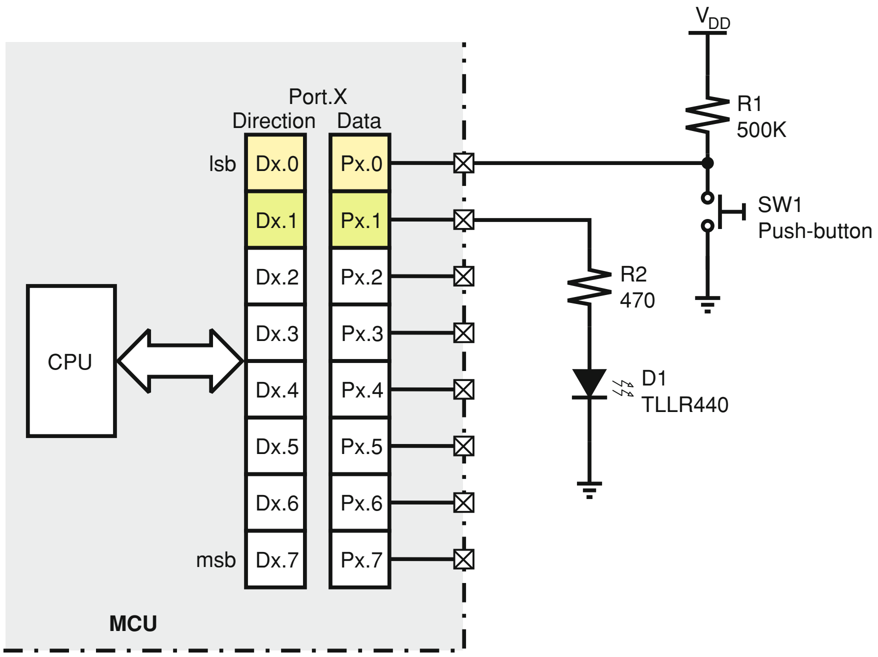


Fig. 8.5 Hardware setup for Example 8.1

Example 8.1 (Push-button/LED Interface) *Consider an 8-bit GPIO port, denoted as Port X* (Px)*, with a momentary push-button SW1 connected to pin 0* (Px.0) *and a light-emitting diode (LED) D1 connected to pin 1* (Px.1)*. A hardware setup for this example is illustrated in Fig. 8.5. Assume* Px *pins are by default GPIOs with direction and data registers at addresses* PxDir *and* PxDat*, respectively. Also assume that a direction bit set to "1" makes its corresponding port bit work as output, otherwise it will work as input. Provide a code fragment to properly configure the port and to toggle the LED whenever SW1 is depressed.*

Solution: *In this setup,* Px.0 *needs to be configured as input and* Px.1 *as output. SW1 has been wired to provide a logic low level when SW1 is depressed, and a "*1*" otherwise. D1 is wired to turn-on when* Px.1 *is set to high. The software solution needs to configure pin* Px.0 *as in and* Px.1 *as output. It will then enter an infinite loop where we poll the switch until detected pressed.*

When SW1 is depressed we'll toggle Px.1 *and proceed to read again the switch status. To avoid LED flickering due to the time SW1 remains depressed with respect to the MCU speed, we'll also detect when the switch is released before returning to poll it for low. The code assumes SW1 is debounced and a header file provides declarations for port registers* PxDir, PxDat. *The pseudo-assembly code fragment below shows a solution for this problem.*

```
;=========================================================================
; Addresses PxDir and PxDat are assumed declared in header file
; Assumes Dir = 1 for In
;-------------------------------------------------------------------------
#include "headers.h"
;-------------------------------------------------------------------------
            ...
            AND #0FEh,&PxDir     ; Set Px.0 bit 0 as input (FEh=11111110b)
            OR #002h,&PxDir      ; Set Px.1 as output (02h=00000010b)
            AND #0FDh,&PxDat     ; Begin with D1 off
            ...
loop_low    TEST #001h,&PxDat    ; Check the value of Px.0
            JNZ loop_low         ; If set, continue to poll Px.0
            XOR #002,&PxDat      ; Otherwise, toggle Px.1
loop_high   TEST #001,&PxDat     ; Check if SW1 was released
            JZ loop_high
            JMP loop_low         ; When released go back to poll depress
            END
;=========================================================================
```

8.1.3 GPIO Ports in MSP430 Devices

MSP430 microcontrollers provide a rich assortment of GPIO pins in its devices. Depending on the particular family member, up to eleven GPIO ports can be found in an MSP430 MCU, enumerated from P1 through P11. The first generation of MSP430 MCUs, series x3xx featured a port P0, but none of the succeeding generations have featured it. Series x5xx/x6xx devices, being the largest devices in pin count per package, feature ports P1 through P11.

Each MSP430 GPIO port has up to eight pins. Every I/O pin can be individually configured to operate as input or output, and each I/O line can be individually read or written to. When configured as input, each I/O line has individually configurable, internal, pull-up or pull-down resistors.

Ports P1 and P2 have interrupt capability, each with its own interrupt vector. The service routine of each port must poll the corresponding interrupt flag register (PxIFG) to determine which pin triggered the interrupt. Interrupts from individual

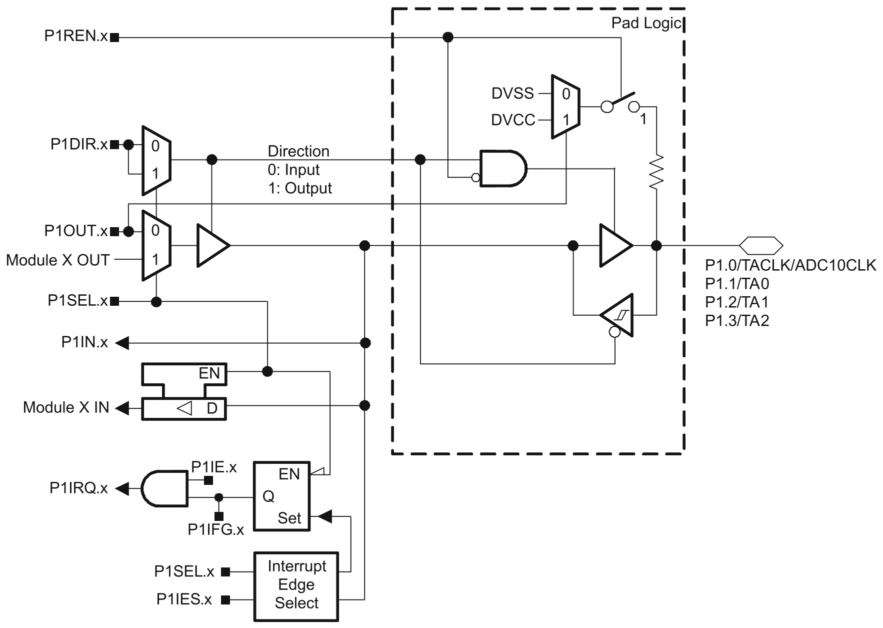


Fig. 8.6 Structure of pin P1.0 in an MSP430F2274 MCU (*Courtesy of Texas Instruments, Inc.*)

I/O lines of each port can be independently enabled/disabled or configured to be triggered with the rising or falling edge of an input signal. Figure 8.6 shows the structure of P1.0 in an MSP430F2274. The particular topology of an I/O pin in the MSP430 will depend on the device itself and the functions shared in the pin. Despite the specific details, it is possible to identify the I/O logic in the pad, the pull-up/pull-down resistor, and the interrupt logic shared by all P1 and P2 pins.

A set of up to ten registers, depending on the device series, are used to configure and operate each I/O port in the MSP430. These registers include:

PxSEL —(Port x Function select) This register selects the functionality to be adopted by a particular pin. When cleared, this bit makes the corresponding pin a GPIO.

PxDIR —(Port x data direction) If a pin is set as a GPIO pin with PxSEL, the corresponding bit in this register sets the pin direction: ’0’ = Input or ’1’ = Output.

PxOUT —(Port x Out) If a pin is set as output, writing PxOUT will cause the corresponding port pin to be driven to the logic level written to the pin. PxOUT is a R/W register and when read will return that last value written to the pin. In MSP430x5xx/x6xx devices with optional pull-up/pull-down resistors in their input mode, the bits of PxOUT are used to select the pulling action of the embedded resistors (when enabled). Setting a PxOUT bit will select the corresponding pin resistor as pull-up, otherwise, when cleared, will set it as pull-down.

PxIN —(Port x Input) If a pin is set as input with PxDIR, this register will reflect the logic level present at the corresponding GPIO pin. PxIn is a read-only register.

PxIFG —(Port x interrupt flag) This flag is set when the corresponding pin experiences the appropriate signal edge transition(either high to low or low to high).

PxIES —(Port x Interrupt edge selection) Establishes the type of transition setting PxIFG. When clear, a low-to-high transition in the corresponding input pin will set PxIFG. Otherwise, a high-to-low transition will set PxIFG.

PxIE —(Port x Interrupt enable) When this bit is set, an event setting the corresponding PXIFG will generate an interrupt.

PxREN —(Port x resistor enable) This register is present only in x2xx and x5xx/x6xx devices. When present, setting this bit will enable the corresponding pull-up/pull-down resistor in the input port.

PxDS —(Port x Drive strength) When present, setting this bit makes the corresponding pin a high-current driver, allowing up to 30 mA while keeping the output voltage within the nominal V_{IL}/V_{IH} value. This increased strength is gained at the expense of a higher power dissipation and increased electromagnetic interference (EMI). This feature is exclusive of x5xx/x6xx devices.

PxIV —(Port x interrupt vector) Provides a priority encoding of the interrupts within a port. When the n lowest interrupts of a port are set, PxIV reads $2n + 2$ allowing this number to be used as a displacement to jump to the corresponding ISR segment where the pin is served. This feature is exclusive to x5xx/x6xx MCUs.

Some MSP430 MCUs have a built-in pin oscillator function in some of their output pins that can be activated for facilitating external sensor interfacing. This feature is particularly useful to interface capacitive touch sensors, as no external components other than a conductive pad is needed in the interface. The conductive pad forms a capacitor that is part of the pin oscillator itself, making the resulting oscillation frequency a function of the loading state of the external pad. Devices featuring such a function, like the MSP430G2553, use internal timer channels in conjunction with pin oscillator inputs for determining the loading state of the external pad. The pin oscillator configuration is device dependent, thus the data sheet of the particular device being used needs to be consulted for specific information. Application note "MSP430 Low-cost PinOsc Capacitive Touch overview" provides a detailed explanation of how to use this feature [45].

Additional details related to the configuration and operation of particular I/O pins in MSP430 devices can be found in the corresponding Family user's manual and in the particular device data sheets. Since topologies and features change from one family to another and within a family from one particular device to another, it becomes necessary to have both pieces of documentation when interfacing or programming a particular chip.

8.2 Interfacing External Devices to GPIO Ports

When interfacing embedded systems to the real world, there will be many instances when the devices to be driven by the MCU GPIO pins, or the signals we need to feed into an I/O will not meet the electrical requirements of the port. When that happens, it becomes necessary to provide a feasible interface to make possible connecting the MCU to the real world.

To determine whether or not an interface will be needed between the MCU I/Os and a particular external device, the first step we need to perform is to revise and understand the electrical characteristics of both, the MCU GPIO's and the external device's. This will unavoidably lead us to the device's data sheets.

To understand the capabilities and limitations of an input/output port, the first thing we need to realize is that like any electronic circuit, GPIOs have voltage and current limits that we need to adhere to when interfacing to them. There are both, electrical and switching characteristics associated to an I/O pin that we need to become familiar with.

In this section we review the fundamental electrical characteristics of GPIOs to help understand their strengths and limitations, and then provide several typical cases when interfaces become necessary for their connection.

8.2.1 Electrical Characteristics in I/O Pins

Like any digital circuit, the electrical characteristics of an I/O pin are dominated by the specifications of the currents and voltages that can be safely managed by the pin. In addition, these also include the chip power supply requirements and thermal limits of the chip itself.

The voltage specifications of an I/O pin are described by parameters of the voltage-transfer characteristics (VTC) of its input and output buffers. Specifically, an input GPIO pin will have the thresholds V_{IL} and V_{IH} of its input buffer. An output GPIO pin will exhibit the voltage limits V_{OL} and V_{OH} of its output driver. These four voltage parameters are described as follows:

V_{IL} is the pin input-low voltage. It establishes the maximum voltage level that will be interpreted as a low by the pin input buffer. The value of V_{IL} is lower bounded by the reference level (V_{SS}) used by the chip, typically ground level or 0 V.

V_{IH} is the pin input-high voltage. It represents the minimum voltage level that will be interpreted as a high by the I/O pin. The value of V_{IH} is upper bounded by the supply level in the MCU (V_{DD}).

V_{OH} is the maximum voltage level exhibited by a digital output pin. The nominal value of V_{OH} is specified for a no-load condition in the pin. In the practice, the value of V_{OH} decreases as the current driven by the pin in its high level increases.

V_{OL} Represents the minimum level observed in a digital output pin. This voltage is also specified for a no-load condition, but in the practice its level increases with the current driven by the pin when in its low state.

A fifth voltage parameter, rarely specified for GPIOs is the pin threshold voltage V_M. By definition, this is a voltage level that when fed to a buffer input will produce exactly the same voltage level in its output. In symmetric CMOS logic $V_M = V_{DD}/2$.

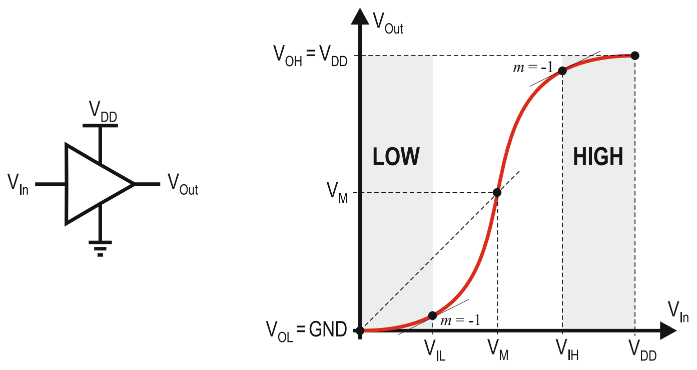


Fig. 8.7 Symbol and voltage transfer characteristic of a conventional buffer

Figure 8.7 shows the voltage transfer characteristic of a conventional non-inverting buffer, denoting all its voltage parameters, including V_M.

The nominal regions of low- and high-level input voltages are clearly marked in the figure. They are lower-bounded by $V_{SS} = 0V$ and upper bounded by V_{DD} (or V_{CC}), meaning that no input shall be applied outside this range. The portion of the input ranging between V_{IL} and V_{IH} is denominated the *transition region* of the pin. Valid digital voltages are expected to avoid this region as they might lead to degraded voltage levels within the input port logic.

An input level falling in the transition region of a pin might actually be internally regenerated to a valid logic level after it passes through a few internal gates. The regenerated level will end up being low or high depending on wether the applied input level is correspondingly below or above V_M. This is why regular logic is said to have a single threshold level V_M.

Despite the regeneration possibility, a good interface is expected to provide valid logic levels, with logic high levels greater than or equal to V_{IH} and logic low levels lower than or equal to V_{IL} to GPIO inputs. If at some point in a design, the voltage fed from a peripheral to a GPIO or viceversa does not meet the input threshold requirements, some form of level-shifting interface becomes necessary. This situation frequently arises when the supply levels of the MCU and one or more of its external peripherals are different. We need to ensure voltage compatibility in the input side. The basic rule here is: if the output levels of the driver (V_{OL} or V_{OH}) fall outside the valid regions of low or high for the load, then a voltage level-shifter must be provided between both sides regardless of who is the driver and who is the load. The rule can be summarized as established in Eqs. 8.1 and 8.2.

$$\text{If } V_{OH_{\text{driver}}} \left\{ \begin{array}{l} < V_{IH_{\text{load}}} \\ \text{or} \\ > V_{DD_{\text{load}}} \end{array} \right\} \Rightarrow \text{lever-shifter needed} \tag{8.1}$$

$$\text{If } V_{OL_{\text{driver}}} \left\{ \begin{array}{l} > V_{IL_{\text{load}}} \\ \text{or} \\ < V_{SS_{\text{load}}} \end{array} \right\} \Rightarrow \text{lever-shifter needed} \tag{8.2}$$

Nowadays many MCUs and peripherals are designed to operate from a 3.3 V supply, but a considerable number of circuits still work at 5.0 V, a voltage level inherited from the TTL era. When both technologies encounter each other in the same application, it becomes necessary to verify compatibility and in most cases taking some measures to make signals compatible.

The solution to the compatibility problem will depend on the direction of the signals. When the driver works at 3.3 V and the load at 5.0 V, in most cases the connection can be directly made. A standard TTL gate with $V_{IHmin} = 2.0\,\text{V}$ and $V_{ILmax} = 0.8\,\text{V}$ will generally accept a 3.3 V CMOS driver with $V_{OH} = 2.7\,\text{V}$ and $V_{OL} = 0.6\,\text{V}$. However, when the connection is in the opposite direction, a level-shifter becomes unavoidable. The simplest solution is usually obtained with a resistor-based voltage divider. This solution works in situations where a high-impedance constant load is to be driven. Low impedance or high-speed loads should not be interfaced with resistor-based dividers as the integrity of signals and edge speed could be compromised. In such cases, active shifters and dedicated buffers offer a better solution. Example 8.2 illustrates a typical situation and a feasible solution.

Example 8.2 (Mixed $\mathbf{V_{DD}}$ MCU Interface) *Consider the electrical interface of a 5 V, HD44780-based LCD module to an MSP430F169 GPIO fed from a 3.3 V supply. Provide an interface such that the LCD memory can be read back from the MCU.*

Solution: *In most cases, LCD modules can be connected in write-only mode by hardwiring their $R/\overline{W}$ input to GND. Under this condition the MCU acts as driver ($V_{OH} \simeq 3.3\,V$, $V_{OL} \simeq 0\,V$) and the LCD as load ($V_{ILmax} = 0.6\,V$ and $V_{IHmin} = 2.2\,V$). In such a case we can get away with a direct connection.*

However, the case here calls for an ability to read back the LCD memory. Such a condition makes the LCD act as a driver with V_{OH} anywhere between 2.4 and 5.0 V, and V_{OL} between 0 and 0.4 V. The MSP430F169 GPIO becomes a load with requirements of $V_{IL} \leq 1.09\,V$ and $V_{IH} \geq 1.65\,V$, upper bounded by $V_{DD} = 3.3\,V$. These specs present no problem at all for the low-level voltages. However, the LCD output-high level voltage would result harmful to the MCU as it exceeds the absolute maximum rated input voltage of any MSP430F169 input, specified at $V_{CC} + 0.3V = 3.6\,V$. In this case a level shifter becomes mandatory.

A feasible solution in this case could be provided with a voltage divider [46]. Note that the MSP input leakage in any GPIO pin is only 50 nA, making them indeed, high impedance loads. Moreover, signal speed are limited to around 2 MHz by the 250 nS cycle width spec in the LCD driver. The interface for this case is illustrated below, in Fig. 8.8.

The values of R_1 and R_2 must be chosen to produce satisfactory low- and high-level voltages in the load even under the worst case voltage conditions. Moreover, they must be appropriately sized to let the gate leakage current (I_Z) circulate without

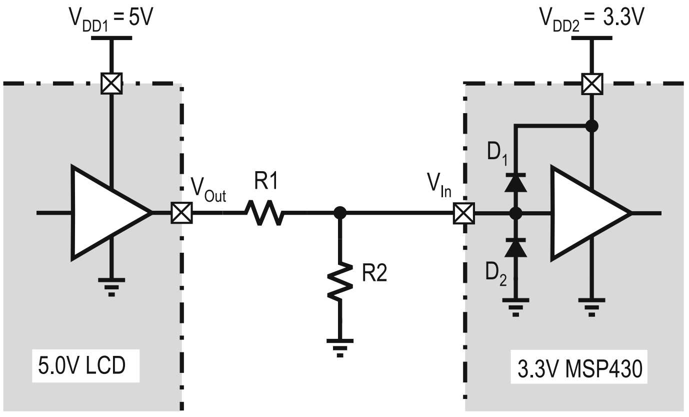
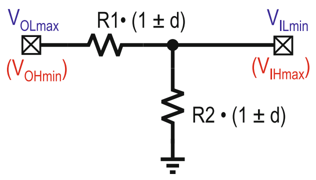


Fig. 8.8 A practical 5.0–3.3 V level shifter

Fig. 8.9 Worst case voltages and resistances in divider network



perturbing the load input logic levels. These requirements are satisfied under the following three conditions:

1. *A worst case, low-level, driver output V_{OLmax} shall produce a voltage equal or less than the nominal low level input in the load. As V_{IL} has maximum and minimum limits, the worst case must satisfy the minimum low-level input V_{ILmin}*
2. *A worst case, high-level, driver output V_{OHmin} shall produce a voltage equal or higher than the worse, high-level input in the load V_{IHmax}.*
3. *The voltage drop caused by the GPIO input leakage (I_Z) on the equivalent voltage divider impedance shall be negligible when compared to the voltage levels applied to the load input.*

We also need to consider the deviations caused by the resistors' tolerances from their nominal value. Assuming both resistor have the same tolerance $\pm d$ %, then their nominal values are affected by a factor $(1 \pm d)$, where $d \ll 1$. The voltage and resistor conditions in the divider are illustrated in Fig. 8.9.

Considering the worse case conditions of R_1, R_2, and the driver-load voltages limits as discussed above, a simple circuit analysis allows for obtaining limiting expressions for the resistor values. Arriving to Eqs. 8.3 through 8.5 requires introducing approximations $(1+d)/(1-d) \simeq (1+2d)$ and $(1-d)/(1+d) \simeq (1-2d)$, as $d \ll 1$.

$$\frac{R_1}{R_2} \geq \frac{(V_{OHmin} - V_{IHmax})}{V_{IHmax} \cdot (1 - 2d)} \tag{8.3}$$

$$\frac{R_1}{R_2} \leq \frac{(V_{OLmax} - V_{ILmin})}{V_{ILmin} \cdot (1 + 2d)} \tag{8.4}$$

$$R_1 \parallel R_2 \ll \frac{V_{DD2}}{I_Z} \tag{8.5}$$

To finally compute the resistances we plug-in the voltage values in 8.3 through 8.5. The LCD data sheet provides values $V_{OHmin} = 2.4\,V$, $V_{OLmax} = 1.5\,V$. The MSP430169 values were obtained by extrapolating those in the data sheet as specific input thresholds for $V_{DD} = 3.3\,V$ are not listed. A quick linear extrapolation on the Schmitt-trigger thresholds yields $V_{IHmax} = V^{+}_{Tmax} = 2.16\,V$, and $V_{ILmin} = V^{-}_{Tmin} = 1.09\,V$, and assuming resistors with 5 % tolerance, Eqs. 8.3 to 8.5 result $\frac{R_1}{R_2} \leq 1.378$, $\frac{R_1}{R_2} \geq -0.575$, and $R_1 \parallel R_2 \ll 66\,M\Omega$.

The negative value in the lower bound for R_1/R_2 only means that the low-level output voltage is already below the minimum required input, which makes it compliant regardless of the resistor ratio. So we can assign $R_1/R_2 = 1$, which would satisfy both conditions. The third condition is also relaxed so we could assign both $R_1 = R_2 = 100\,K\Omega$, which satisfies all conditions.

Note that in this particular case the LCD can still be written back by the MCU through this level shifter. The LCD input leakage I_{Zlcd} is specified at 1 μA for its entire voltage range. The voltage drop caused by the level shifter when driven by the MCU would be $V_{drop} = I_{Zlcd} \cdot R_1 = 1\mu A \cdot 100\,K\Omega = 0.1\,V$, thus the LCD input would still have an input voltage high enough to qualify as a logic-high level. The MCU would not see a significant load: $I_{MCUout} = I_{Zlcd} + I_{R2} = 1\mu A + 3.3\,V/100\,K\Omega = 34\,\mu A$, which is indeed a light load for the GPIO pins.

One last observation about the solution in Example 8.2: the assignment of R_1 and R_2 has tradeoffs. The higher their value, the lower the power dissipation as the current through the divider will be lower. However, the time constant resulting from the equivalent resistance and the load input capacitance will go up making the interconnect slower.

A better solution in terms of switching speed is offered by a pass-transistor operating as a bidirectional level shifter like that illustrated in Fig. 8.10. This solution effectively shifts voltages in both directions and the switching times are improved as the on-resistance of a MOSFET is typically less than 200 Ω. The circuit analysis is left as an exercise to the reader.

8.2.2 Schmitt-Trigger Inputs

GPIO input ports with Schmitt-trigger capability behave differently from standard logic inputs. For such inputs, instead of V_{IL} and V_{IH}, the parameters of interest are the positive- and negative-going input thresholds V^{+}_{T} and V^{-}_{T}. A Schmitt-trigger capable

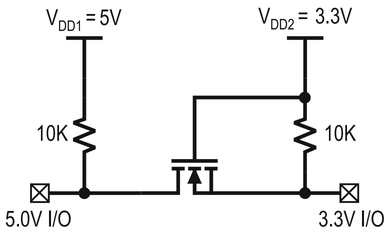


Fig. 8.10 A bidirectional level-shifter

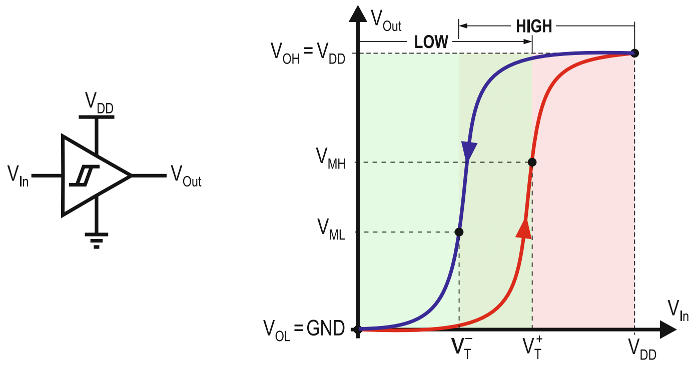


Fig. 8.11 Symbol and VTC of a Schmitt-trigger buffer

input is necessary when interfacing slow rising or noisy signals to a digital circuit.. It is common in contemporary MCUs to feature Schmitt-trigger capable inputs in part or all its GPIO inputs.

A Schmitt-trigger buffer adds hysteresis to its behavior by establishing two different threshold voltages, V_T^+ and V_T^-. A rising input signal would need to reach V_T^+ to make the output change from low-to-high. Similarly, a falling input signal would need to drop at least to V_T^- to cause a high-to-low output transition. This behavior can be observed in Fig. 8.11, where the symbol and voltage transfer characteristic of a typical Schmitt-trigger buffer are illustrated.

The slope in the transition region in a Schmitt-trigger buffer VTC is much steeper than that of a conventional buffer. This is an effect of the internal positive feedback used in the buffer to have hysteresis. This instability makes the buffer exhibit abrupt output transitions when the input voltage reaches the low- or high-going thresholds. This property is exploited to sharpen the edges of slow changing input signals. This behavior is illustrated in Fig. 8.12a where a slow rising input is illustrated feeding Schmitt-trigger buffer. Also, when compared to a conventional, single threshold (V_M) buffer, a Schmitt-trigger, when fed in its input a noisy signal, provides a cleaner output than that of a conventional buffer. Figure 8.12b compares the outputs from a both types of buffers in front of a noisy input. Note how the undesirable output

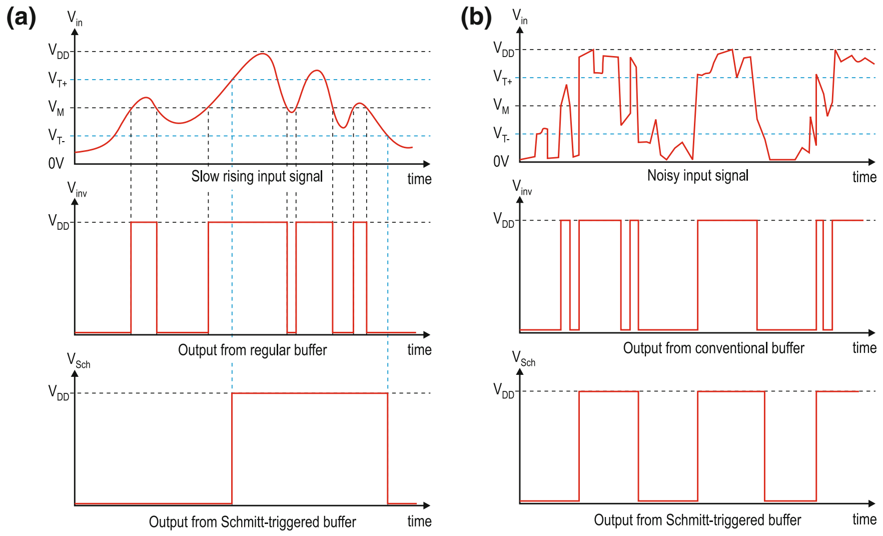


Fig. 8.12 Schmitt-trigger effect on noisy and slow rising inputs. **a** Response to slowly changing input, **b** Response to noisy input

transitions caused by noise in the conventional buffer output are eliminated from the Schmitt-triggered output.

8.2.3 Output Limits in I/O Pins

The practical values of the output levels V_{OL} and V_{OH} in a GPIO change with the current sank or sourced by the pin. This bring us to the definitions of the current parameters of an I/O pin.

In standard logic, five different current values might be specified associated to a GPIO pin. These include the high- and low-level input currents I_{IH} and I_{IL}, the high- and low-level output currents I_{OH} and I_{OL}, and for tri-state buffers the fifth specified values is the high-impedance leakage current I_Z, which quantifies the current leaking through the pin when in a high-impedance state.

These current values in well designed devices reduce to three: the output strength in either low- or high-level I_{OL} and I_{OH}, which are dependent on the load, and the input leakage I_Z. The input leakage describes in a single value the former I_{IL}, I_{IH}, and I_Z when inputs have been designed to have a high impedance input regardless of their state.

By convention, manufacturer's data sheets assign to each of these currents positive signs when they flow into a digital pin and negative otherwise. Despite their sign the current magnitude is what determines the internal voltage drop caused in a driver,

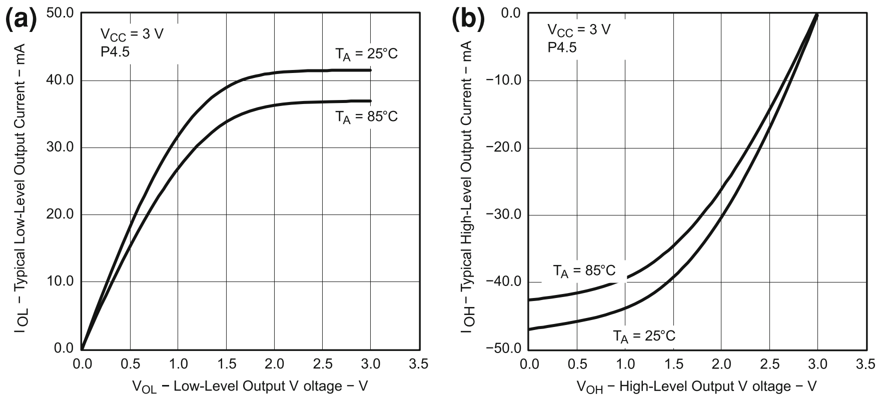


Fig. 8.13 Driving strength as a function of the output voltage in an MSP430F2274 (*Courtesy of Texas Instruments, Inc.*) **a** $I_{OL} = f(V_{OL})$, **b** $I_{OH} = f(V_{OH})$

defining the net output voltage seen at its output. This explains why we have varying specifications for V_{OL} and V_{OH} in function of the load current.

To exemplify this dependence, let's consider the case of an MSP430F2274. Its leakage current I_Z is specified at ± 50 nA, while its driving capability is given as a function of the pin current for each logic level. Figure 8.13 reproduces the manufacturer's specification in GPIO P4.5 when only one output is loaded at a time and the supply voltage is 3.0 V.

Both, Fig. 8.13a, b show how the output voltages change with the load current in both the low and high states. For an output voltage $V_{OH} = 80\,\%V_{DD}$ we see that the maximum level-high current source capability would be around 25 mA. Similarly for a nominal $V_{OL} = 0.6$ V, the maximum I_{OL} results approximately 22 mA. Both values were estimated at 25 °C. The curves denote that the current capability at any given voltage is lower at a higher temperature. Although not shown in the figure, the output current capability also depends on the supply voltage level. According to the chip data sheets, if the supply voltage were reduced to 2.2 V, the maximum currents for $V_{OH} = 80\,\%V_{DD}$ and $V_{OL} = 0.6$ V would have resulted in approximately 12 and 17 mA, respectively [48].

The limiting current values of an output pin must be observed when it is being loaded. A design shall not cause an I/O pin to degrade its output voltage at any logic level, as this might take the voltage levels to the point where the high and logic levels cannot be differentiated. Even worse, making the pin current reach or exceed the device's absolute maximum current ratings would cause irreversible damage on the chip.

As a general rule, a designer needs to ensure that the total load current connected to an output pin never exceeds the pin rated current for the nominal load voltage. A simple KCL (Kirchhoff's Current Law) verification shall suffice to determine the loading conditions. Given an output k driving n loads, where each load i, $i = 1, 2, \ldots, n$, requires current $I_{IL(i)}$ when driven low and $I_{IH(i)}$ when driven high, as illustrated in Fig. 8.14, we must make certain that the rules in (8.6) and (8.7) are both

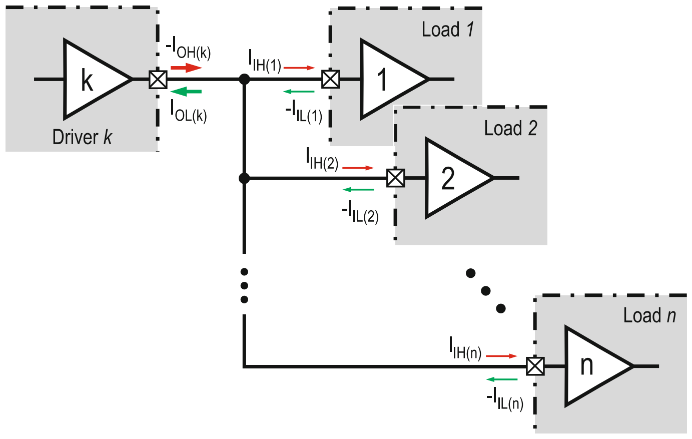


Fig. 8.14 Currents in a loaded driver

satisfied.

$$\sum_{i=1}^{n} |I_{IL(i)}| \leq |I_{OL(k)}| \tag{8.6}$$

$$\sum_{i=1}^{n} |I_{IH(i)}| \leq |I_{OH(k)}| \tag{8.7}$$

If we need to drive a total load that exceeds the current limit of the pin in either state, a buffer providing sufficient current driving capability is needed. Section 8.8 discusses several buffering options when driving large loads.

8.2.4 Maximum Power and Thermal Dissipation

Besides the total current per pin, the total power dissipation of the chip must also be kept within bounds. An MCU might have all its I/O pins driving currents below their individual limits, but that does not mean the chip is being safely operated. There is a limit in the maximum power dissipation that a chip can safely withstand. This limit is determined by the maximum temperature the die inside the chip case can withstand. Some manufacturers explicitly provide in their data sheets a limit for the total power dissipation of their chips, but this specification is more commonly given in the form of the thermal limits of the packaged device, due to its dependence in the ambient temperature.

A packaged MCU, when operated at a certain supply voltage and clock frequency, will generate heat from its die. The total temperature accumulated in a die

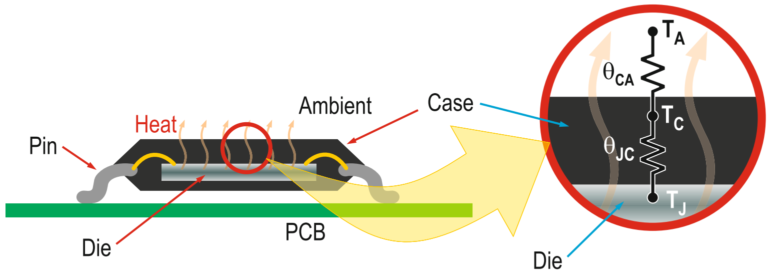


Fig. 8.15 MCU heat dissipation denoting thermal resistances θ_{JC} and θ_{CA}

(junction temperature Tj will result from the addition of the heat caused by the chip power dissipation and the ambient temperature (T_A). The chip ability to release this heat accumulation will depend on the thermal conductance of the die surrounding materials. The lower the heat conductivity, the higher the thermal resistance of a material. The thermal resistance is measured in centigrade degrees per watt (°C/W). Figure 8.15 shows a sectional view of a packaged IC. There we can observe that heat would need to pass from the die to the case (junction to case thermal resistance θ_{JC}) and from the case to the surrounding air (case to ambient thermal resistance θ_{CA}. The values for these thermal resistances for a chip in a given case are provided in the device data sheets or in the package specifications.

The total thermal resistance from junction (die) to ambient when no heat dissipation mechanism has been added to the chip can be written as:

$$\theta_{JA} = \theta_{JC} + \theta_{CA} \tag{8.8}$$

The die temperature can be estimated as the sum of the external temperature and that generated by the power dissipation as:

$$T_J = T_A + \theta_{JA} \cdot P_{\text{diss}}, \tag{8.9}$$

where P_{diss} is the total chip power dissipation. P_{diss} can be obtained from the supply voltage V_{DD} and the sum of all I/O currents handled by chip the as:

$$P_{\text{diss}} = V_{DD} \cdot \left(I_{DD(avg)} + \sum_{allpins} |I_{IO(avg)}| \right) \tag{8.10}$$

Note that making a precise estimate of the total power dissipation of an embedded design is a complex problem. Power in digital circuits is a function of the operating voltage, driven load, clock frequency, and processor activity. From this list of parameters, perhaps the most difficult to obtain is the processor activity since it depends on the information being processed by the CPU. The expression in Eq. 8.10 is a rough estimate assuming a constant supply voltage and the average currents have been adjusted to account for low power modes in use, use duty cycle, supply voltage, and clock frequency. For a deeper insight into the problem of estimating power in embedded systems, the reader is referred to [49].

If we were interested in the chip maximum power dissipation, it can be obtained from (8.9) using the ambient temperature T_A, and the manufacturer's provided maximum junction temperature $T_{J(max)}$ and the junction-to-ambient thermal resistance θ_{JA} as:

$$P_{\text{diss}(max)} = \frac{(T_{J(max)} - T_A)}{\theta_{JA}} \tag{8.11}$$

If the load on a chip makes the IC to exceed its maximum power dissipation, its junction temperature will reach values where early thermal failure may occur. To reduce the chip junction temperature, possible alternatives include the following approaches (or their combination):

1. Reduce the chip power dissipation. This can be achieved by placing external buffers to reduce the current demand on the IC or by lowering the chip supply voltage level, if that were an option. Modern MCUs, including the MSP430, provide a valid range of supply voltage options. Also, improving the power consumed by the application by exploiting low power modes and good hardware and software design practices can be very effective in reducing the MCU power consumption.
2. Reduce the total thermal resistance of the chip. A reduction on θ_{JA} can be obtained by two means: attaching a heat sink to the IC package and/or forcing the air circulation around the chip with a fan. The thermal resistance reduction induced by a heat sink will be a function of the heat sink area. When a heat sink is attached to an IC, the junction to ambient resistance formula is modified to replace θ_{CA} with the heat sink thermal resistances from case to sink θ_{CS} and form sink to ambient θ_{SA}. In this case the total thermal resistance is:

$$\theta_{JA} = \theta_{JC} + \theta_{CS} + \theta_{SA} \tag{8.12}$$

   If a fan were included, the air volume per unit time in cubic-feet per minute (CFM) moved by the fan would determine an adjustment factor $\gamma < 1$ that multiples θ_{SA}, lowering the net thermal resistance.
3. Reduce the ambient temperature T_A. As the net junction temperature is a direct function of the ambient temperature, a reduction of the surrounding chip temperature will help.

Figure 8.16 shows a solution combining heat sink and fan on a microprocessor to keep the die junction temperature cool.

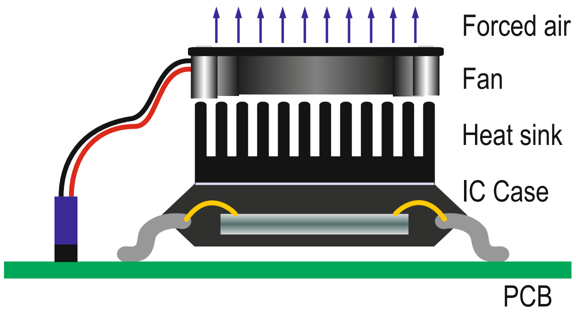


Fig. 8.16 IC cooling approach combining *heat sink* and *fan*

8.2.5 Switching Characteristics of I/O Pins

The second important set of specifications in GPIO regards their timing specifications. These define how fast a GPIO pin can be switched.

The maximum switching frequency of an I/O pin is generally defined by the voltage supply level, load capacitance, and the main clock frequency. Input pins have a minimum pulse width they can detect. For example, an MSP430F5438 specifies the minimum detectable pulse width in a GPIO input enabled to trigger interrupts at 20 nS.

Output pins would generally be limited by the maximum system clock and the external capacitance being switched. In the MSP430F5438 the maximum output frequency in an 18 MHz part is 25 MHz when the load capacitance does not exceed 20 pF. As the load capacitance or output resistance increase, the load time constant increases, softening signal edges and making transitions slower. This is manifested by an increase in the rising and falling times of the signals being propagated. Moreover, I/O pins associated to specific internal peripherals have their own timing constraints, the data sheets specific to a given part number or their particular peripheral specification need to be consulted to learn about the specific limitations that each case might have.

8.3 Interfacing Switches and Switch Arrays

One of the components most commonly interfaced to MCUs GPIO ports are mechanical contact switches. Switches provide for intuitive user interfaces and practical detection mechanisms for a vast number of applications. The states of a switch, either open or closed, perfectly match with the binary detection ability of an input GPIO line.

Depending on the application, switches might be used individually, like the case illustrated in Example 8.1. Other applications use arrays of switches, such as the

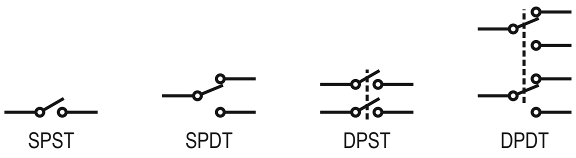


Fig. 8.17 Symbols of four common types of switches

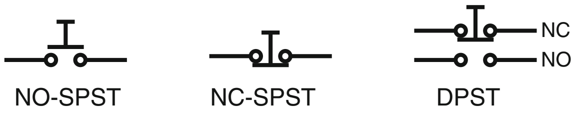


Fig. 8.18 Common types of momentary push-buttons

case of conventional keypads or keyboards. Depending on the case, particular switch interfaces can be used to interface them to GPIO ports.

8.3.1 Switch Types

Many variants of switches can be found in electronic circuit applications. A common classification is based on the number of contacts (poles) operated an the positions (throws) adopted by the contacts. Under such a classification, a *single-pole, single-throw* (SPST) switch has only one contact that is either open or closed, while a double-pole, single-throw (DPST) has two contacts that are either open or closed and moved by a single throw mechanism. Figure 8.17 illustrates a few typical switch configurations.

One particular type of switch widely used in MCU interfaces is the momentary push-button (MPB). A normally-open (NO) momentary push-button is a SPST switch that remains open at all times, except when it is being depressed. A normally closed (NC) MPB will remain closed at all times except when actuated. Figure 8.18 shows three types of push-button switches.

8.3.2 Working with Switches and Push-Buttons

The interface of a single switch to a GPIO input pin is quite simple. It only needs to provide a means of ensuring that the two possible switch states, OPEN or CLOSED, are represented with appropriate logic levels. Figure 8.19 shows two simple connections for a momentary push-button (SW). In the first case, node (x) provides a logic low level when the switch is closed, while in the second, it provides a logic high for the same switch condition. It results straightforward to see why node x in Fig. 8.19a goes low when SW is closed: it gets directly connected to GND. When SW is open, we need to ensure that node x goes high. To this end we attach pull-up resistor $R_{pull-up}$ to the node. Note that without this resistor, when SW opens, node

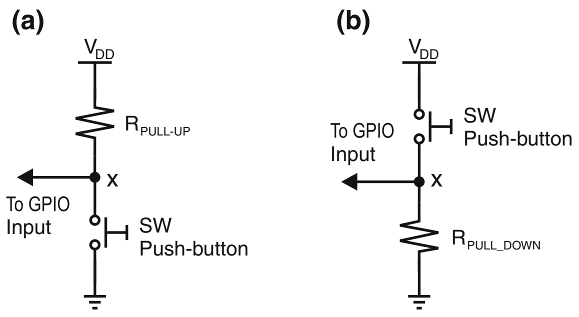


Fig. 8.19 Common interfaces for momentary push-buttons. **a** MPB with pull-up resistor, **b** MPB with pull-down resistor

x would simply float to a high impedance state, which would not necessarily be interpreted as a valid high level.

The case illustrated in Fig. 8.19b is similar to the above in terms of providing a reliable logic level when the switch is open. Except that in this last case the OPEN state is encoded a logic low. The pull-down resistor $R_{pull-down}$ does the job in this case.

In either case the value of the resistor is chosen to satisfy two criteria: first, to allow for having a valid logic level when SW is open and second for limiting the current circulating to ground when SW is closed. In the case of a pull-up resistor, the voltage at x is expected to be within the valid range of a logic high for the GPIO, i.e. $V_x \geq V_{IH}$. Thus, if the amount of current leaking into the GPIO pin in high were I_{IH} then the value of $R_{pull-up}$ would need to satisfy (8.13).

$$V_{DD} - R_{pull-up} \cdot I_{IH} \geq V_{IH} \tag{8.13}$$

Thus, an upper bound for the pull-up resistor would be:

$$R_{pull-up} \leq \frac{V_{DD} - V_{IH}}{I_{IH}} \tag{8.14}$$

If we establish a limit to the current to ground when the switch is closed, to be for example $I_{SWmax} = 10I_{IH}$, then we can also set a lower bound to the resistor. This however might be unnecessary since in most cases the limit obtained from V_{IH} would yield a satisfactory value. In this case we would say

$$R_{pull-up} \geq \frac{V_{DD}}{I_{SWmax}} \tag{8.15}$$

If a V_{IH} specification were not readily available we can safely assume $V_{IH} = 0.9V_{DD}$. Some MCUs have a very small input current in either state, reason for which this current is listed as an input leakage, denoted by I_Z.

Example 8.3 *Consider an MCU operating with $V_{DD} = 5.0\,V$, with an input current of $1\,\mu A$ in its GPIO pins for either low or high levels and $V_{IHmin} = 0.7V_{DD}$. Find a suitable value for a pull-up resistor to interface a NO-MPB to one of its input pins.*

Solution:*In this case we will assume $V_{IH} = 0.9V_{DD}$ which satisfies the minimum V_{IH}. Applying Eq. (8.14) we'd obtain*

$$R_{pull-up} \leq \frac{0.1V_{DD}}{1\,\mu A} = 500\,K\Omega \tag{8.16}$$

If we assign $I_{SWmax} = 20I_{IH} = 20\,\mu A$ then an upper bound for the resistor would be

$$R_{pull-up} \geq \frac{V_{DD}}{20I_{IH}} = \frac{5.0\,V}{20\,\mu A} = 250\,K\Omega \tag{8.17}$$

Thus, any value in the range $250\,K\Omega \leq R_{pull-up} \leq 500\,K\Omega$ would satisfy these specifications. For the sake of low-power, choosing the largest value would yield the lowest current when the switch is closed.

When using a pull-down resistor, as illustrated in Fig. 8.19b the reasoning would be similar to the previous case, except that we'd need to look into providing $V_x \leq V_{IL}$ when the switch is open. Thus we'd use the specification for I_{IL} to find the upper limit of $R_{pull-down}$, and, if necessary, a I_{SWmax} criteria for a lower bound.

8.3.3 Dealing with Real Switches

So far we have assumed that switches, when operated will cleanly close or open such that their interfaces produce one sharp high-to-low or low-to-high when opened or closed. That would be true only under ideal conditions. Real switches have multiple non-ideal characteristics that need to be considered when used in the real world. Among such non-idealities we can list the following:

- Non-zero "ON" Resistance: although small, the resistance between switch terminals, when closed is never zero.
- Limited contact current capability: there is a maximum current the switch can safely handle when closed. Exceeding this value accelerates contact wear and leads to early failure.
- Limited Dielectric Strength: This is the maximum voltage the open contacts can withstand before arcing.
- Limited Mechanical Life: Number of times the switch is rated to be operated.
- Finite Contact Bounce Time: Time required to contacts to settle after operated.

From this list, a non-ideality of concern in embedded systems design is contact bounce. When two mechanical contacts are operated, they do not open or close cleanly. Instead, they rebound a certain number of times, sometimes over one hundred times, before steadying at their final state. The maximum time taken by the contacts

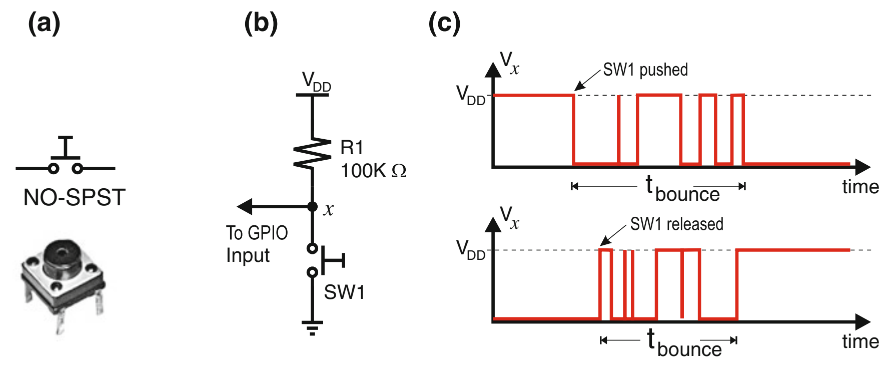


Fig. 8.20 Bouncy behavior of mechanical NO-MPB. **a** Actual Switch, **b** Interface Circuit, **c** Output waveform. The *top* and *bottom* waveforms correspond to the switch closing and opening, respectively

in a switch to reach steady state after being operated is called the *switch bounce time*. Figure 8.20 shows a common type of push-button used in many embedded applications, its interface, and the manifestation of contact bounce in the interface output when operated.

8.3.4 Characteristics of Switch Bounce

The waveforms in Fig. 8.20c illustrate the effect of bouncing in the switch output, denoting that the phenomena occurs in both, switch closures and openings. The top plot shows the closing bounce, while the bottom waveform shows the behavior upon switch opening. Typical bouncing times changes from one switch to another. An informal study carried by Ganssle shows that although most switches bounce for 10 ms or less, it is possible to find some switches with bouncy behavior extending for over 150 ms [52]. Quality switches typically have less than 10 ms bounce time. If we are uncertain about the bouncing time of a particular switch, a simple experiment with a digital storage oscilloscope (DSO) can help.

8.3.5 Problems Associated to Contact Bounce

Bouncing causes that a single switch throw be interpreted as multiple operations, causing in many cases incorrect system operation. Consider for example an embedded system implementing a TV remote control. If the channel flipping button were not properly debounced, depressing the channel up key would actually make the unit jump several channels at once. Other scenarios can be devised.

Switch bouncing is particularly troublesome when the switch action is set to trigger an interrupt or other edge triggered action. Every bounce will account for a trigger, potentially unleashing a chaotic software behavior. Polling might also be affected by bouncing, particularly if the polling interval is shorter than the bounce time. To avoid such situations, the solution is to implement some sort of switch debouncing approach in every design using mechanical switches. Note that this recommendation is valid not only for manually operated switches, but also for any application where system operation were driven from the opening or closure of mechanical contacts.

8.3.6 Switch Debouncing Techniques

The problem of bouncing can be approached from either a hardware or software standpoint. A hardware approach will insert circuit components such as a filter or some form of digital logic to suppress at the digital circuit input the transient pulses caused by the switch bounce. Software-based approaches allow the bouncy signal to enter the MCU, and then deals with the transient in the program, preventing the interpretation of the transitions associated to the switch bounce. Below we present some of the most widely used hardware and software bounce techniques.

8.3.7 Hardware Debouncing Techniques

Hardware debouncing techniques attempt to filter the switch transient behavior, feeding to the digital input pin one clean transition each time the switch is operated. There are different ways to debounce a switch by hardware. Below we discuss representative hardware debouncing circuits.

SR Debouncing Circuit: One of the most effective ways of debouncing a switch is by using a Set-Reset (SR) latch between the switch and the digital input, as illustrated in Fig. 8.21.

The SR latch circuit will debounce SW1 regardless of its bounce time. R1 and R2 in this circuit act as simple pull-up resistors. When SW1 is in the up position, the latch will remain set, yielding a logic high at the output. When the switch is flipped, the latch will be reset and the output will transition from high to low. When the contact bounces, the latch will just remain reset regardless of the number bounces. A similar situation will happen when SW1 is toggled to the set position. If multiple switches were to be debounced using this method, a 74HC279 with four SR cells in a single package, or a 74HC373, which has eight, might become handy. One disadvantage of this circuit is SW1. It needs to be a SPDT switch, which is more expensive and bulkier than a regular NO momentary pushbutton.

RC Debouncing Circuit: A cost-effective solution to debounce the contacts of a switch can be achieved by using an RC network as debouncer, as illustrated in Fig. 8.22.

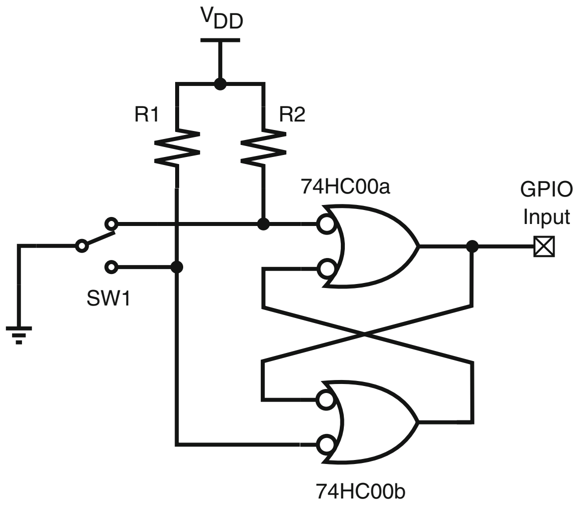


Fig. 8.21 SR latch-based switch debouncing circuit

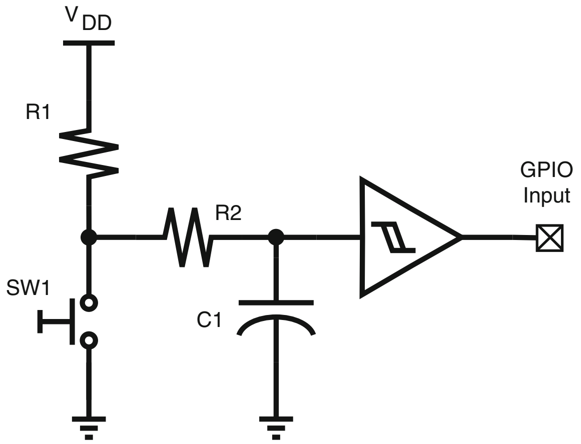


Fig. 8.22 RC network-based switch debouncing circuit

An RC debouncer uses a resistor-capacitor pair to implement a delay in the switch line. To understand how it works, let's refer to Fig. 8.22. Assume SW_1 has been open long enough, to have C1 charged to V_{DD}. Note that while SW_1 is open, C_1 charges through $R_1 + R_2$. When SW1 is closed, C1 begins to discharge through R_2. If the time constant R_2C_1 is large enough, when SW_1 contacts bounce and open, voltage V_x at the Schmitt trigger input would not have reached the gates's input-low threshold V_T^-, preventing the circuit output from making a transition. To determine the values of R_2 and C_1, we just need an estimate of the bounce time t_{bounce}.

The formula describing the voltage at node x would be that of discharging C_1 through R_2:

$$V_x = V_{DD} \cdot \exp^{-\left(\frac{t_{bounce}}{R_2C_1}\right)} \tag{8.18}$$

Since (8.18) has two unknowns, R_2 and C_1, we need to estimate one of them. It is usually easier assigning a value to C_1 and solving for R_2 since the latter can be more easily accommodated with a standard resistor value. Thus, considering that we want to have $V_x \geq V_T^-$ at the end of t_{bounce}, we obtain the value for R_2 as:

$$R_2 \geq \frac{t_{bounce}}{C_1 \cdot \ln\left(\frac{V_{DD}}{V_T^-}\right)} \tag{8.19}$$

To calculate the circuit requirements for debouncing the switch aperture, let's assume that before the switch is released, the contacts have been closed long enough to completely discharge C_1, making $V_x(0) = 0V$. Opening SW_1 will cause C_1 to begin loading through $R_1 + R_2$. Again, we want the time constant $(R_1 + R_2)C_1$ to be long enough such that V_x remains below V_T^+ before the bouncing finishes. So, with C_1 and R_2 already assigned, all we need to do is to compute the value of R_1. In this case the value of R_1 can be obtained via the charging equation for C_1, that describes V_x as:

$$V_x = V_{DD}\left[1 - \exp^{-\left(\frac{t_{bounce}}{(R_1+R_2)C_1}\right)}\right] \tag{8.20}$$

Considering that we want $V_x \leq V_T^+$, the sum of R_1 and R_2 needs to satisfy:

$$(R_1 + R_2) \geq R_{min}, \tag{8.21}$$

where

$$R_{min} = \frac{t_{bounce}}{C_1 \cdot \ln\left(\frac{V_{DD}}{V_{DD}-V_T^+}\right)} \tag{8.22}$$

Thus, resulting in $R_1 \geq (R_{min} - R_2)$.

This circuit solution shall work in most cases. Some designers might recommend replacing R_2 by a shortcircuit. This alternative in addition to risking not providing enough delay for $SW1$ closure, will also cause an accelerated wear of SW_1 contacts due to the current peak resulting from the shorting of C_1. Depending on the threshold values of the Schmitt trigger buffer, there is a possibility of ending up with a value of $R_{min} > R_1$, which makes the expression for R_1 an absurd. In such a case, a modified version of the circuit in Fig. 8.22 would be recommended, inserting a diode in parallel to R2, with its anode connected to node x [52]. This would make the circuit to charge C_1 through R_1 and the diode, while the discharge would occur exclusively through R_2. It is left as an exercise to the reader obtaining the expressions for the circuit components.

IC Debouncer Circuit: Sometimes, instead of using discrete components to implement a hardware debouncer, it might be more convenient an integrated solution. This might be particularly convenient if a debouncing solution for multiple switches must include externally to the MCU all resistors, capacitors, and Schmitt trigger buffers. In such cases, commercial off-the-shelf alternatives such as the ELM410

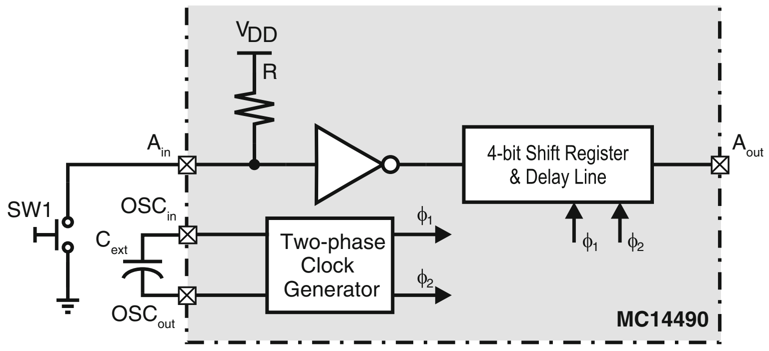


Fig. 8.23 IC-based switch debouncing circuit using the MC14490

from Elm Electronics or the MC14490 from ON-Semiconductors might result appropriate. As an example, let's consider On Semiconductor's MC14490 Hex Contact Bounce Eliminator. The chip contains six independent debouncing circuits for equal number of switches. It provides on-chip pull-up resistors, buffers, and a reloadable shift registers to provide independent rising and falling edge debouncing for all six switches. A single on-chip clock generator provides the internal frequency ϕ driving the shift registers. Figure 8.23 shows a block diagram of the debouncer structure.

The debouncing strategy in this IC assumes the switch contacts have settled when no bouncing transitions are detected for four consecutive cycles of internal clock ϕ. The frequency of ϕ is determined with external capacitor C_{ext} (one per chip). Some of this IC solutions might result handy and compact, but they are also expensive and parts that might not be offered by all chip distributors.

8.3.8 Software Debouncing Techniques

Software debouncers have the attractiveness of not requiring external components beyond the switch and pull-up resistor illustrated in Fig. 8.19. However, they require CPU cycles to remove the bouncy portion from the switch signal. There are a few general rules that should be followed to design better software debouncing routines:

- A software debounced switch shall not drive an interrupt input. Interrupt inputs usually drive the clock lines of input flip-flops, which might behave erratically with the very short pulses generated by the switch bounce.
- Avoid wasting CPU cycles by sending the CPU to do-nothing NOP cycles or executing useless instructions while you wait for the switch contacts to settle. This will waste not only CPU cycles, but also energy. Activate some low-power mode while you wait or get the CPU to do some other useful task.
- Keep response time the shortest possible. Humans can detect response delays as small as 75 ms, which make the system feel "slow".

- Keep solutions simple. A debouncing routine is not expected to be complex. Afterwards, it is just a switch what is being interfaced.

A few representative software approaches are described below.

A Polling Debouncer: This simple alternative polls the switch port with a constant polling period longer than the expected switch bouncing time. If for example the mean bounce time of a switch were 15 ms, then polling the switch about every 20 or 25 ms would suffice to avoid the switch bounce. To avoid wasting cycles while waiting for t_{bounce} to expire, one could use a timer to set the poll time and send the CPU to a low-power mode. This alternative, although simple, might fail to debounce a switch with bounce time that exceeds the assumed value of t_{bounce}.

A Counting Debouncer: A better approach is to assume the contacts have settled if they have not bounced for a certain number of samples *n* [52]. If *n* consecutive settled samples are collected, then a valid edge indicator is returned. Otherwise, if a change in the switch status were detected before collecting the *n* samples, then the count would be reset and counting restarted. A feasible implementation could use a timer to periodically sample the switch. Typical values of inter sample time could be between 1 and 10 ms, while setting *n* to around a dozen samples. The pseudo assembly fragment below would provide a suitable implementation of the timer ISR.

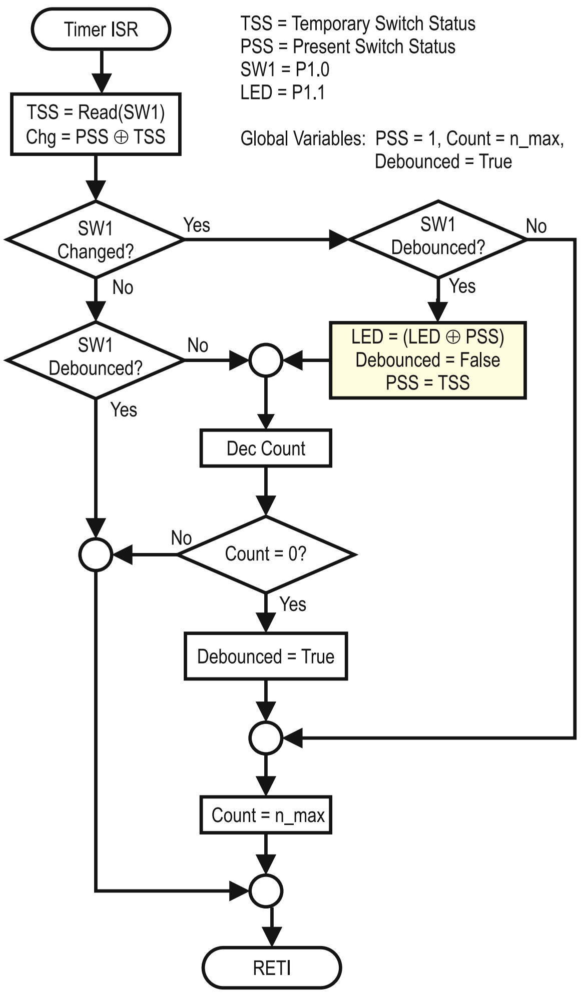
```
;=========================================================================
; Timer service routine for debouncing a switch closure (Closed = Low)
; Global variables "S_Count" and "EdgeOK" initialized to FFFFh and FALSE
;-------------------------------------------------------------------------
Timer_ISR   BIT.B #SW1,&PxDat    ; Load switch status into carry flag (C)
            JC Bouncing          ; If high then contact was found open
            RLC.W &S_Count       ; Valid contact state detected & counted
            CMP #F000h,&S_Count  ; Is it the 12th steady sample?
            JNZ Exit             ; Valid closure but not there yet
            MOV.B #TRUE,&EdgeOK  ; Count top reached
Bouncing    MOV #FFFFh,&S_Count  ; Restart counter
Exit        RETI
;=========================================================================
```

The assembly function above is compact, simple, effective, and portable. However it only debounces the switch closure and would delay the notification of the received edge. Example 8.4 illustrates an improved solution that debounces both the switch closure and opening while also preventing edge repetition due to long switch closures and providing early notification of the switch state change.

Example 8.4 (SW Debounced MPB & LED) *Redesign the software solution in Example 8.1 to provide for a software debounced switch interface for SW1. Solution is to maintain the toggle functionality without repetition due to the switch being held pressed.*

Solution: *In this case we adopt the counting approach coupled with an edge detection capability that allows detecting either a closure or opening of the switch. A complete solution is provided where the main program initializes all structures: I/O port, Timer, and variables, and then sends the MCU to a low-power mode. Timer A0 wakes the MCU approximately every 2.2 ms to sample SW1. The whole debouncing*

Fig. 8.24 Flowchart for software debouncing timer ISR in Example 8.4



process is performed in timer A0 ISR. A flowchart of the function is illustrated in Fig. 8.24.

An assembly implementation of this solution is provided below. The implementation targets an MSP430x2xx, like that in TI's LaunchPad. The solution makes use of Timer A channel 0 to space samples in time. Although the discussion of timers is ahead, in Chap. 7, the provided comments should guide the reader through the solution steps.

```
;===================================================================
; Symbolic constants and addresses are declared in MSP430 header file
; Assumes fclk = 32.768KHz from LF XTAL1
;-------------------------------------------------------------------
#include "msp430g2231.h"
```

```
;-------------------------------------------------------------------
Top_Cnt EQU 12 ; Number stable SW samples to be debounced

RSEG DATA16_N
Count DS 1     ; Sampling counter
PrsSwStat DS 1 ; Present switch status  R7
Debounced DS 1 ; Debounced state  R8

RSEG CSTACK    ; Stack segment

RSEG CODE
Reset MOV.W #SFE(CSTACK),SP        ; Initializes the stack area
      MOV.W #WDTPW+WDTHOLD,&WDTCTL ; Stops the MSP430 WDT

;----->Port 1 Setup
      BIC.B #BIT1,&P1DIR ; Set Px.0 bit 0 as input
      BIS.B #BIT0,&P1DIR ; Set Px.1 as output
      BIC.B #BIT0,&P1OUT ; Begin with D1 off

;----->Timer A0 Setup
      MOV.W #TASSEL_2+MC_2,&TACTL ; Source from ACLK, continuous up
      MOV.W #72,&TACCR0           ; Sampling set to about 2.2ms
      MOV.W #CCIE,&TACCTL0        ; TACCR0 interrupt enabled

Main  BIS.B #BIT0,&PrsSwStat ; Present SW state is OPEN
      BIS.B #BIT0,&Debounced ; Present status is debounced
      MOV.B #Top_Cnt,&Count  ; Initialize Count
      BIS.W #LPM0+GIE,SR     ; Enable interrupts, LPM0
      NOP

;-------------------------------------------------------------------
; Timer A0 service routine for debouncing SW1 and toggling LED1
;-------------------------------------------------------------------
TimerA0_ISR PUSH R5 ; Used for temporary switch state (TSS)
            PUSH R6
            MOV.B &P1IN,R5 ; Load current switch status into R5
            AND.B #BIT4,R5 ; Screen-out SW1 bit
            RRA.B R5       ; Align LED and SW1 bits
            MOV R5,R6      ; Save TSS into R6
            XOR.B &PrsSwStat,R5; Detect if switch state changed
            JZ NoChg ; No change wrt previous state

            BIT.B #BIT0,&Debounced  ; Check if already debounced
            JZ ChgNDbc              ; Changed but not debounced
            XOR.B &PrsSwStat,P1OUT  ; Debounced & changed => Toggle LED
            CLR.B &Debounced        ; Now SW1 is undebounced
            MOV R6,&PrsSwStat       ; Change the present switch state

NChgNDbc    DEC.B &Count            ; Update count of stable samples
            JNZ Exit                ; If less than top count then exit,
            BIS.B #BIT0, &Debounced ; else, set debounced as true

ChgNDbc     CLR.B &Count            ; Reset the sample counter
            JMP Exit

NoChg       BIT.B #BIT0,&Debounced  ; Check if already debounced
```

```
            JZ NChgNDbc             ; No Change & not debounced

Exit        POP R6                  ; Restore registers
            POP R5
            RETI

;-------------------------------------------------------------------
COMMON INTVEC ; Interrupt Vectors
;-------------------------------------------------------------------
           ORG TIMERA0_VECTOR
           DW TimerA0_ISR
           ORG RESET_VECTOR
           DW Reset
END
;===================================================================
```

8.3.9 Switch Arrays

Many applications require multiple switches to be interfaced to the MCU. Under such situations, switch arrays come in handy to facilitate the interface. Two general types of switch arrays can be identified: Linear arrays, also called DIP Switches, and matrix type, also referred to as keypads or keyboards.

8.3.10 Linear Switch Arrays

A linear switch array contains multiple independent SPST slide switches, typically in a dual-in-line package, which lends them the denomination of DIP Switches. DIP Switches come in groups from two to about twelve, being the DIP-8 one of the most commonly used in MCU systems. Figure 8.25 shows the symbol and actual package of a DIP-8 switch array.

DIP switches are frequently used to provide hardwired codes or configuring hardware components in a easily modifiable way. Each switch in a linear array needs to be individually interfaced, and if they are meant to be directly read by the MCU, each will require its own I/O pin. Moreover, if the switches are meant to be read by the CPU while they are operated, debouncing techniques will also be required.

If a linear switch array were to be hardware debounced, the same techniques available to individual switches would apply. One must consider however the impact on the design recurrent costs if this modality were to be applied.

Software debouncing techniques usually offer a more cost effective alternative when dealing with switch arrays (shall CPU load permit). In this case, instead of individually debouncing each switch, it is more convenient to debounce them all simultaneously. This is particularly simple if all the switches are connected to contiguous pins in the I/O port(s). An approach similar to that of the counting debouncer

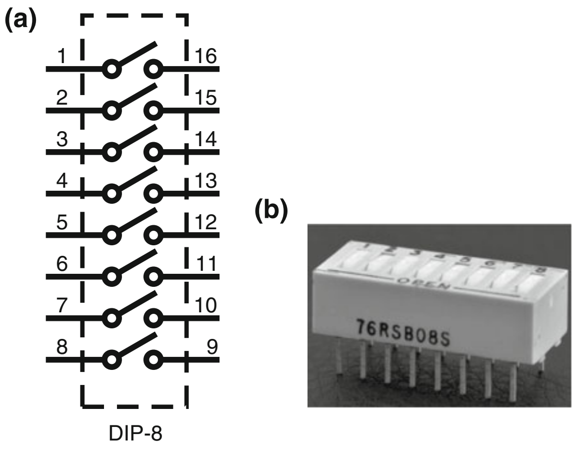


Fig. 8.25 A typical DIP switch: **a** DIP-8 symbol, **b** actual DIP-8 switch

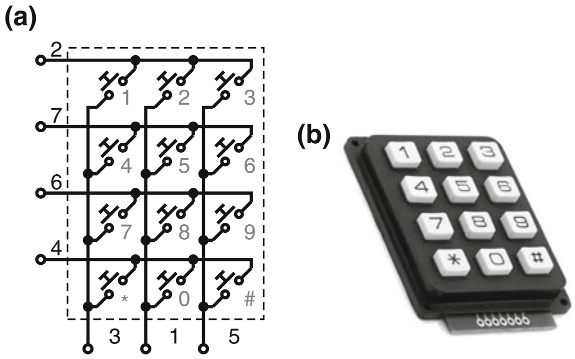


Fig. 8.26 A 3-by-4 keypad showing the internal arrangement and appearance. **a** Switch arrangement, **b** Actual keypad

in the previous section could be followed, except that instead of acquiring and checking a single switch line, the entire switch port(s) is treated at once.

The algorithm depicted in Fig. 8.24 could easily be adapted to debounce a switch array by changing the instances where the switch is read and its changes detected by accesses to the entire switch I/O port. The adaptation of the function is left as an exercise to the reader.

8.3.11 Interfacing Keypads

The second type of switch arrays correspond to keypads and keyboards. Keypads have switches arranged in a matrix form where individual switch status can be decoded via a scan algorithm applied to rows and columns. Figure 8.26 shows a schematic diagram and an actual picture of a twelve-key keypad arranged as a three-by-four matrix.

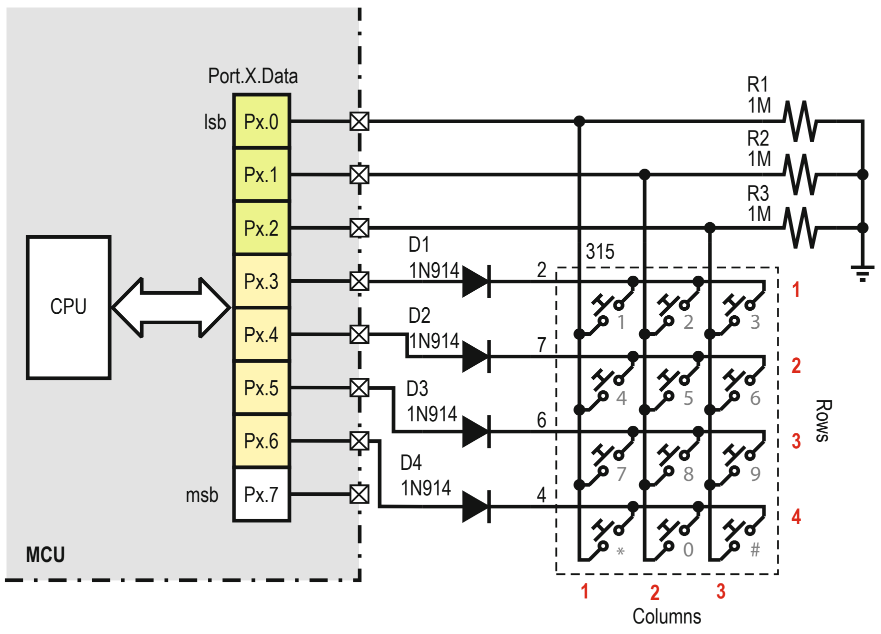


Fig. 8.27 Interfacing a 3-by-4 keypad to a GPIO port

The main advantage of arranging the switches in a matrix form is the reduction in the number of I/O lines required to read the entire switch array. Consider for example, a 64-key keyboard. Arranging the keys to form an 8-by-8 matrix allows using only sixteen IO lines to interface the array, instead of the 64 lines it would have required should we had chosen to interface each key individually. In general, an array of N keys can be interfaced using $2\sqrt{N}$ I/O lines when arranged in matrix form.

Although a matrix approach simplifies the interface, decoding de array to detect individual key closures requires a dedicated keypad scanning algorithm. Two possible types of solutions can be chosen: devote CPU cycles and I/O lines to scan the keypad, or using a dedicated interface chip.

CPU-based Keypad Scanning: An interface for having the CPU scanning a keypad requires allocating I/O lines to connect row and column lines to the MCU. Row lines can be connected as inputs and columns as outputs (or viceversa). This arrangement allows for issuing a scan code through the output lines and reading the return sequence through the input lines. The designation of either rows or columns as inputs can be arbitrarily done if the key array has an equal number of columns and rows. When the number of rows and columns is different, it is generally preferred to designate the group with the lesser number of lines as inputs. This is because the set of lines designed as inputs need to contain key interfaces to the I/O port. Figure 8.27 shows a schematic diagram of an interface for a 3x4 keypad to an MCU I/O port.

The connection illustrated in Fig. 8.27 presumes input lines, in this case P1.0 to P1.2, are bounce-free. If a software debouncing algorithm were to be implemented, an

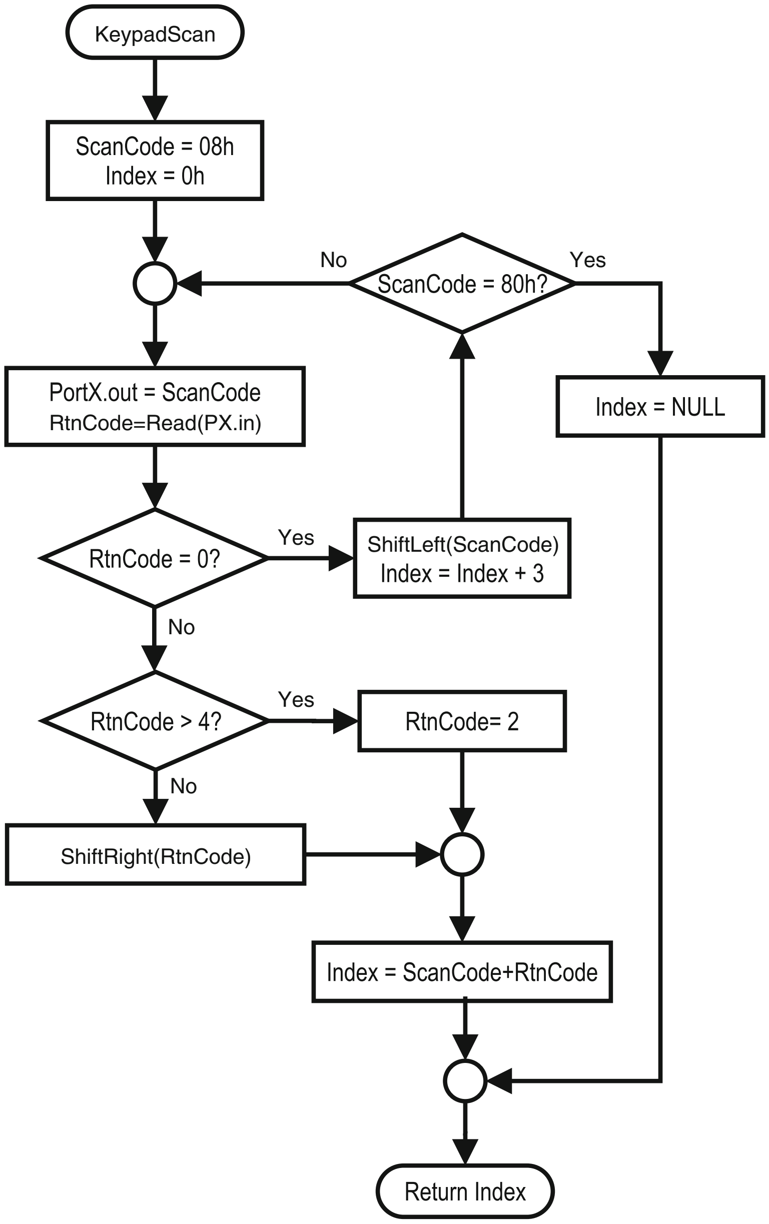


Fig. 8.28 A scanning algorithm for the 3x4 keypad in Fig. 8.28

approach serving all input lines at once, like that described in Sect. 8.3.10 for a linear array, could be applied. If a hardware debouncing solution were preferred instead, each input line would require the addition of hardware debouncing components.

To understand the basic scan algorithm let's assume the keypad rows are connected to port outputs and the columns to inputs using pull-down resistors as illustrated in Fig. 8.27. As in the case of single switches, the pull-down resistors will cause the corresponding switch to be read as a low-level when open. If a high-level voltage were applied to only one of the keypad rows by setting the corresponding output pin to high, reading the keypad columns via input pins would unequivocally give us the state of all the keys in the row that was driven to high. For example, setting row 2 (PortX.4) to high while key "8" is depressed, would return code "010" through lines PortX.0 through PortX.2.

Thus, by performing a sequence that sends scan codes "0001", "0010", "0100", and "1000" would activate each row one at time. If after sending each individual scan code we read the column lines (return code), the combination of each scan code with the corresponding return code would unequivocally allow us to identify any key depressed in the entire keypad. A keypad scan algorithm would just need to perform this sequence in a loop. Thus, if while this algorithm were running keys 1, 5, and 8 were depressed, one at a time, the corresponding seven-bit codes resulting from their scan-return codes would be "0001001", "0010010", and "1000010", respectively. Note that in these bit sequences, the upper four bits represent the scan code and the lower three bits contain the corresponding return code.

The trick that makes this basic algorithm work without missing the detection of any single keystroke is the microcontroller's ability to perform the scan sequence much faster than a human can type.

A few observations can be made in regards to this algorithm and its supporting hardware. First, if multiple keys in the same row were simultaneously depressed, the row return code would allow their identification within a single scan. However, if multiple keys of the same column were depressed, lines from different rows would become joined, with the potential to create a short circuit between the output port lines. Some keypads have embedded diodes in their keys, which prevent this situation. Inexpensive keypads rarely have this provision. In such cases it is advisable to include diodes in the scan code lines, as illustrated in Fig. 8.27 with diodes D1 through D4. Once the scan/return code combination has been obtained, all that is left is just assigning a meaning to each of the collected key codes.

Figure 8.28 shows a flowchart of a simple procedure to assign sequential numeric codes to the scan of a 4x3 keypad connected as illustrated in Fig. 8.27. The returned "Index" would assign a binary between 00h and 0Bh to the depressed key in the order "123456789*0#". If no key were depressed a null character is returned. If multiple keys were depressed, only the one with the highest return code will be returned.

Dedicated Keypad Interfaces: In some applications, devoting CPU cycles for scanning, debouncing, and encoding a keypad might not be feasible. In such instances it might be convenient to consider the usage of a dedicated keypad interface.

A dedicated keypad interface provides the convenience of accommodating in a single chip the necessary logic to scan, debounce, and encode the keys of matrix or non-matrix type keypads and keyboards. Such interfaces relieve the MCU from allocating I/O pins and cycles, reducing code complexity in keypad interfaces at the expense of increasing the count number of board components and increasing the recurrent system costs.

Specific features of dedicated keypad interfaces vary from one vendor to another. Some commonly found features include:

- Debouncing Capabilities
- Key repetition
- Interrupt capabilities
- Serial output

A representative example of such type of interfaces is the EDE1144. This chip incorporates the ability of handling matrix type keypads up to 4x4, providing serial output and even the ability of tactile feedback via an external buzzer. Another example of dedicated keypad interface is the 74c922. This chip provides a simpler interface including the ability of handling keypads up to 5x4 and supporting key debouncing and encoding.

Despite their convenience, these types of chips tend to be expensive and might not be available with every vendor. For these reasons, when a dedicated keypad interface becomes necessary in an application, some designers prefer to devote a low cost MCU solely devoted to interfacing keypads or key arrays. For example, a small MCU like the MSP430G2231 could be programmed to serve a keypad up to 5x5, allowing to define by software all the desired features in the keypad performance. An application note for the MSP430F123 published by Texas Instruments under the number SLAA139 describe an ultra low-power keypad interface that provides a good starting point to such an application.

8.4 Interfacing Display Devices to Microcontrollers

The second most common type of interfaces in embedded systems are display devices. Coupled with switches and keypads, visual display components form the most common type of user interface that allows humans and other species interact with embedded systems.

Nowadays there is a broad selection of display devices that can be attached to microcontrollers to provide visual information. These range from discrete light emitting diodes (LEDs) to indicate binary conditions to intricate colorful plasma, LED, and Liquid-crystal Displays (LCD) flat panels capable of displaying real life still images and videos.

A broad classification of displays can be made, based on the amount of visual information they are able to provide. These include discrete indicators, numeric displays, alphanumeric displays, and graphic panels.

- **Discrete Indicators** refer to the usage of discrete light emitting diodes (LEDs) or some other display mechanism to provide single dot indicators. Traditional red, green, blue, and yellow LEDs provide for simple monochromatic, binary indicators able to provide an On/OFF or Yes/No indications. Polychromatic single dot indicators are also possible when using RGB LEDs. An RGB LED is just a combination of three monochromatic LEDs conforming the three basic colors: red, green, and blue. RGB LEDs, with their ability to produce colors, can be used to provide more than simple binary indications. Eight different indications ($2^3 = 8$) are possible by the binary combinations of its three basic LEDs. An even larger gamma of indications can be achieved by regulating the intensity of each individual basic color.

- **Numeric Displays**: Typically seven-segment displays that allow only displaying numeric information with decimal digits 0 through 9. Both, LED- and LCD-based displays of this type can be found.
- **Alphanumeric Displays**: These displays are able to produce both numbers and alphabetic characters using either denser segment arrays, like 14- and 16-segment displays, or the most commonly used, dot matrix displays. In this last category we predominantly find LCD- or LED-based displays, although there is a niche of household and audio applications where gas-discharge displays are predominant. A class of alphanumeric, dot-matrix-based displays deserves special mention: the dot-matrix type LCD display, widely used in many small embedded applications.
- **Graphic Panels**: These displays have the characteristic of being pixel addressable making possible to reproduce graphic information. A wide gamma of devices fall in this category. In the low end we find small monochromatic graphic LCDs with 128x64 dots or less, organic LED displays (OLED), and small LED matrix displays like those used in banner signs. In the high-end we find large LED screens like those in electronic billboards, and dense LCDs and plasma displays like those used in television receiver sets and computer monitors. Other graphic technologies include electronic ink (e-ink) displays like those used in early electronic readers.

In this section we analyze the most common types of displays used in small embedded applications. Their principles of operation are the same as those in larger display technologies, forming a basis for understanding how they operate in a larger scale.

8.5 Interfacing LEDs and LED-Based Displays to MCUs

Light Emitting Diodes (LEDs) provide the most basic display element used nowadays in embedded systems. An LED is simply a PN junction in which the photon emission phenomena due to conduction is enhanced. When electrons from the N-side in a forward biased PN junction cross to the P side and recombine with holes, the energy lost by the carriers due to the recombination process causes photons to be released. This is an intrinsic phenomena in all PN junctions. In an LED, wide bandgap materials and their dopant concentrations are manipulated to enhance the emission of photons and control their wavelength to produce light of different colors. A lens on the top of the LED package concentrates and focuses the emitted light to direct it in a particular

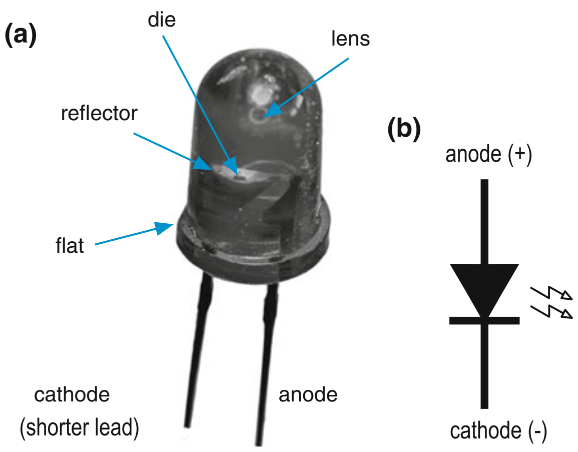


Fig. 8.29 Structure of a typical THT LED and LED symbol. **a** Structure, **b** Symbol

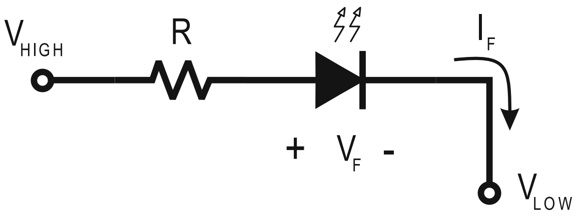


Fig. 8.30 Basic interconnect of a discrete LED

direction or just difusses it. Figure 8.29 shows the structure of an LED, along with its symbol.

Like any other type of diode, LEDs conduct and therefore, emit light, only when they are forward biased. LED specifications include their forward bias voltage (V_F), typically 1.2 V or larger, their forward current (I_F), and their brightness (B) for a given current intensity, measured in mcd (millicandela). Roughly speaking, one candela is about the amount of light delivered by a candle lit in a dark room.

8.5.1 Interfacing Discrete LEDs

The simplest LED interface requires a voltage source and a resistor R in series with the LED to limit the forward current circulating through the LED. Figure 8.30 shows the basic circuit of an LED interconnect.

The value of R is chosen to allow I_F circulate through the LED. Considering generic voltages V_{LOW} and V_{HIGH} applied to the anode and cathode of the diode as illustrated in Fig. 8.30, the value of R can be obtained with (8.23).

$$R = \frac{V_{HIGH} - V_F - V_{LOW}}{I_F} \tag{8.23}$$

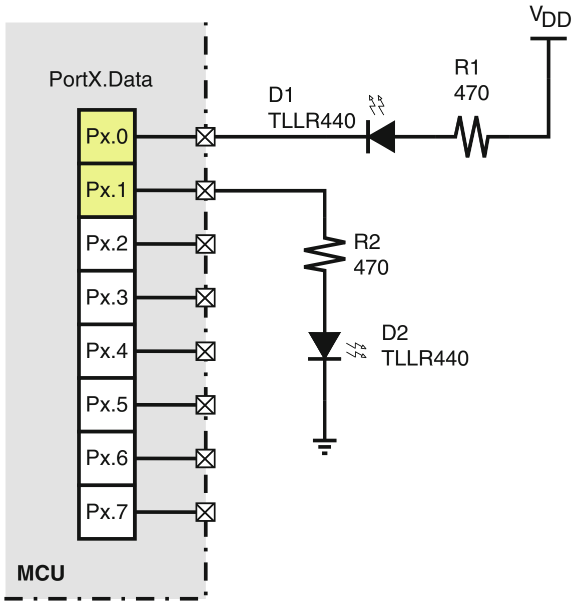


Fig. 8.31 Driving LEDs with I/O pins

In most cases, an individual LED can be directly driven by an MCU output pin. The main criteria to determine if a direct connection is possible are based on whether or not the output pin has enough driving capability to handle the LED forward current, and if the port output voltage is large enough to produce across the LED the required V_F. Most LEDs used for signalization require V_F around 1.8–2.2 V with current between 5 and 20 mA, which are manageable by most GPIO ports. LED connections requiring larger currents or voltages than those directly provided by a GPIO pin directly need buffering and/or voltage level shifting interfaces like those discussed in Sect. 8.8.

Depending on how the LED is connected, two different configurations are possible: LED turned-on with a logic high or turning-on with a logic low. Figure 8.31 shows both of these configurations. D_1 is connected to turn-on when Px.0 is low, while D_2 would be on with Px.1 is high. The values of V_{OL} and V_{OH} in each case provide the respective levels of V_{LOW} and V_{HIGH}. In each case we must ensure the I/O pin currents I_{OL} for D_1 and I_{OH} for D_2 are large enough to sink and source I_F in each case.

8.5.2 Interfacing LED Arrays

One of the earliest applications of LED-based displays used seven-segment numeric LED arrays. A seven-segment LED display features an arrangement of seven bar-shaped LEDs in a pattern that allows to form the shape of any digit between 0 and 9

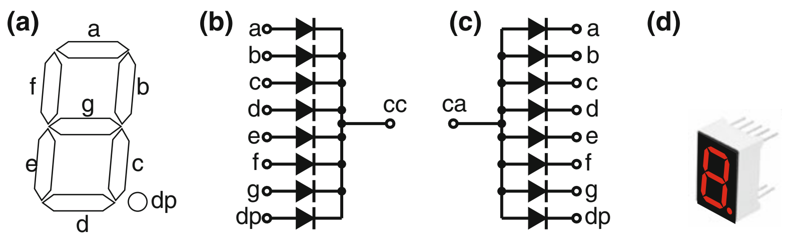
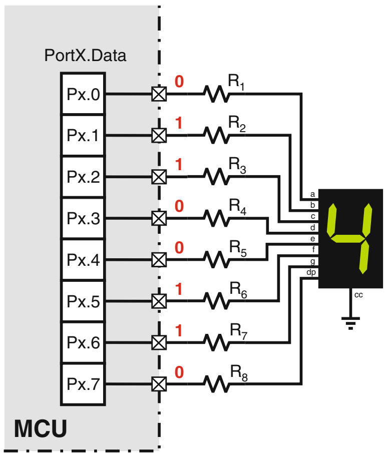


Fig. 8.32 Ordering of individual LEDS in a 7-segment display. **a** Segment arrangement, **b** Common cathode, **c** Common anode, **d** Actual Component

Fig. 8.33 Directly driving a 7-segment display module from a GPIO port



and other characters by selectively turning on or off specific LED segments. Many seven-segment display units also include an eighth LED to provide a decimal point to the right of the digit. Figure 8.32 shows the arrangement given to the seven LEDs forming a 7-segment LED display plus decimal point.

The segments in an LED-based seven-segment display are internally connected to either have a common anode or a common cathode terminal, as illustrated in Fig. 8.32b, c. This eases the interconnection of the modules in typical applications. The codes to illuminate a particular digit will depend on the numeral being represented. For example, to obtain the digit "4" on the display, assuming the least significant bit of the port is connected to segment "a", "b" to the next significant bit, and so on, we'd need to send the binary code "01100110", or 66 h, assuming a common cathode display. This case is illustrated in Fig. 8.33 with a single seven-segment display being directly driven by a GPIO port.

From an electrical point of view, driving the segments of a single 7-segment display is not different from driving set of discrete LEDs. For each segment we'd need to provide I_F at the specified V_F to have it lit. The values of resistors R_1 through R_8 are chosen the same way as in the case of individual LEDs. This assumes that the I/O lines have enough driving capability to source the I_F of each segment. In the event that I_F exceeds I_{OH}, a buffer would be required. Although individual transistors could still be used, dedicated seven-segment drivers offer a more compact solution. For example, the 74HC47, a BCD-to-seven segment could be used. This IC, besides providing enough current for each segment, also features inputs accepting BCD digits, which reduces the input pin requirement.

Interfacing multi-digit displays is usually done by time-multiplexing individual seven-segment modules. This technique, when compared to using a dedicated GPIO port per digitt, saves I/O pins at the expense of a slightly more elaborated software component.

A multiplexed, multi-digit seven-segment display shares the segment lines among all modules, while the common cathodes (or anodes in the case of CA modules) are used to individually turn on one module at a time. To show all digits in an n-digit number, the seven-segment codes for each digit are sent one at a time, each followed by a control code that activates only the seven-segment module where the digit is to appear, and leaving the other digit drivers deactivated.

If this operation were performed for all the n digits in the display, and the entire sequence repeated in a loop such that all the digits were refreshed at least 24 times per second, a human eye would see all the digits lit simultaneously, each with their corresponding digit displayed. The reason why this trick works is due to the persistence of vision phenomena in the human eye.

To achieve a suitable refresh frequency for the entire display, a timer channel could be devoted to set the digit refresh rate. Note that if we desire a refresh rate of f_{refresh} in an n-digit display, each individual digit will need to be refreshed at a frequency equals to $(n \times f_{\text{refresh}})$. At the same time, in each refresh cycle, for the entire display, each digit will be lit $1/n$th of the cycle. This will reduce the brightness of each digit in the same proportion, which could lead to dim displays if no counter measures were taken.

One alternative to increase brightness in such a situation would be by increasing the current through each segment in the same proportion the duty cycle is reduced, an objective easily reached by reducing the limiting resistors in each segment. When this strategy is used, care must be taken not to exceed the maximum current allowed per segment at a given duty cycle or per I/O port.

Another important observation is that the CA or CC terminal will circulate the sum of the currents of all ON segments. This sum could be as high as $8I_F$ if all segments are on. For most GPIOs this amount of current would exceed the port driving capability, requiring an external buffer. Example 8.5 illustrates a multiplexed LED design that considers these conditions.

Example 8.5 (Multiplexed multi-digit seven-segment display) *An interface for a three-digit multiplexed seven-segment CC LED display is desired to operate with a refresh frequency of 33.3 Hz. Provide a suitable hardware setup and a timer-based ISR to set the desired refresh rate.*

Solution: *In this case we will provide a solution using eleven I/O lines: eight for driving the segments in Port1, and three in Port2, bits 0 to 2, for controlling the common cathodes. Assume the MCU is an MSP430F1222 operating at 3.3 V, with*

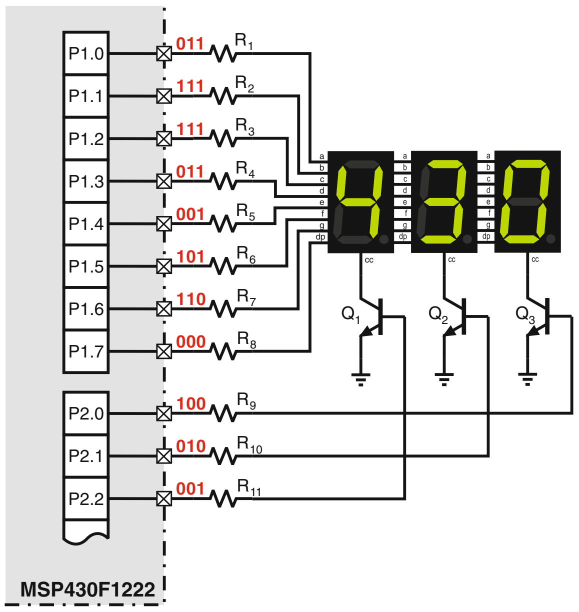


Fig. 8.34 Multi-digit seven-segment display arrangement

$f_{clk} = 32.768\,KHz$. Figure 8.34 shows a diagram illustrating a connection of the three modules to form a three-digit display.

Note that lines "a" through "f", driven by Port1, are shared by all three modules. The common cathode lines are controlled with lines P2.0 through P2.2, with the most significant digit driven by PY.2. Transistor Q1 through Q3 are necessary to drive the entire module current, which can be up to $8 \cdot I_F$, when all segments are on. Section 8.8 discusses how to design transistor interfaces.

Let's use commercial off-the-shelf, three-digit, common cathode, seven-segment modules similar to Kingbright's BC56-12SRWA [54]. These modules specify a typical brightness of 34 mcd at $I_F = 10\,mA$. This is a rather high brightness level for a seven-segment display. We would set it to provide an average of 5.67 mcd, which would allow enough brightness even on daylight. As the luminous intensity is proportional to I_F, the continuous current needed would be 1.67 mA. However, as we have three digits, the duty cycle would be 33 %. So we will set $I_F = 5\,mA$ to obtain the desired average brightness.

At 5 mA forward current, the display module specifies a forward voltage V_F of approximately 1.75 V. Note that the common anode could, in the worst case, manage 40 mA per digit, justifying the usage of transistor buffers Q1 through Q3 in the seven-segment CC terminals. For such a current, a Fairchild's 2n3904 NPN general purpose transistor would suffice. The 3904's data sheets specify at $I_{Csat} = 40\,mA$ a current gain factor $\beta_{sat} = 10$, $V_{CEsat} = 0.075\,V$, and $V_{BEsat} = 0.82\,V$ at room temperature of 25 °C. Thus, the values of the limiting resistors R_1 through R_8, and base resistors R_9 through R_{11} would result:

$$R_1 = \frac{V_{OH} - V_F - V_{CEsat}}{I_F} = \frac{3.1\,V - 1.75\,V - 0.075\,V}{5\,mA} = 255\,\Omega$$

$$R_9 = \frac{V_{OH} - V_{BEsat}}{I_{Bsat}} = \frac{3.1\,V - 0.82\,V}{4\,mA} = 570\,\Omega$$

The circuit setup shows next to the port outputs the binary codes to be sent if we wanted to display the value "430" on the display. Note there are three columns of binary numbers there. Each column represents an instant in time. The upper 8 bits represent the seven-segment (plus decimal point) code, while the lower three bits represent the corresponding control code.

For setting a display refresh frequency of 33.3 Hz with three digits we would need at least a 100 Hz digit refresh rate. This implies refreshing each digit every 10 ms. If we configure a timer channel to trigger an interrupt every 10 ms, that would yield the desired refresh rate. Figure 8.35 shows a flowchart for the timer ISR. The implementations details are left as an exercise to the reader.

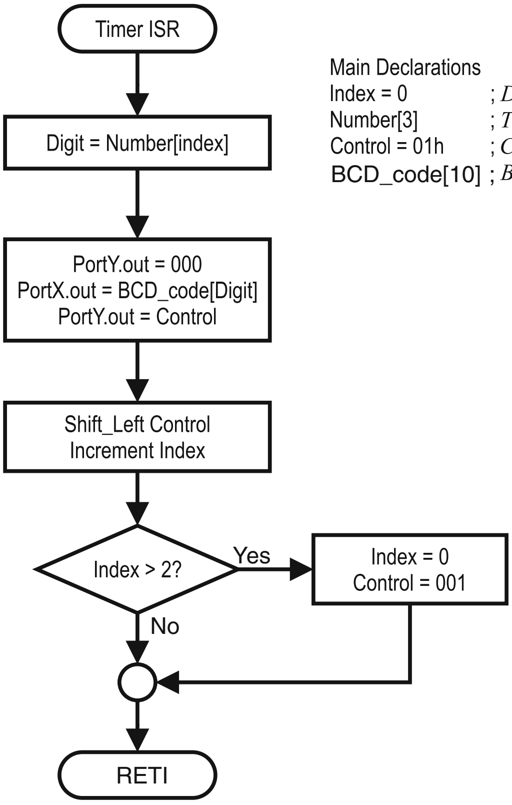
Some applications might either not have enough I/O pins or the CPU might be too busy to devote cycles to directly manage the refreshing of a multiplexed display. In such cases, using an external LED display driver might result appropriate. Several off-the-shelf alternatives can be identified, including the Motorola MC14489B, Intersil's ICM7218, and Maxim's MAX6951. They all provide the ability of driving up to eight seven-segment displays, providing serial interface to the MCU side and taking care of driving and multiplexing functions.

8.6 Interfacing Liquid Crystal Displays

Liquid Crystal Displays (LCD) have become the dominant display technology in household and portable devices. Its low-power consumption and compactness have been the major drives to this tendency. The once dominant cathodic ray tube technology has now become obsolete, being displaced by LCD flat screens in desktop and portable computers. In the arena of portable electronics, there has never been any technology with more dominance than LCDs.

8.6.1 Principles of Operation of LCDs

All LCD technologies are based on the property of crystalline solid materials of twisting the polarization angle of coherent light. Liquid crystalline materials, also called nematic state materials, were first discovered in 1888, and their thin-layer properties studied for many years. In 1970, eighty-two years after their discovery, liquid crystals found their way to mass consumed electronics in the form of numeric displays for



Main Declarations

Index = 0	; *Digit index*
Number[3]	; *Three-digit BCD number*
Control = 01h	; *Common Cathode control*
BCD_code[10]	; *BCD to seven-segment LUT*

Fig. 8.35 Timer service routine for multi-digit display

wristwatches. Nowadays, liquid crystals are the base for the most common type of display technology employed in embedded applications.

Liquid crystal materials do not emit light. Instead, they manipulate the light coming from some other source when it passes through the crystal body. Typical light sources in LCD applications include ambient light or some form of backlighting illumination. The manipulation of light in the liquid crystal can be made to produce patterns of shadows in front of the light source, allowing the formation of images.

What makes possible the manipulation of light inside a liquid crystalline material resides in the properties of its molecules. Nematic state materials are composed by rod-like crystal molecules that in their natural state are organized in a helix form similar to that of a DNA strand. These helixes work as light wave guides, twisting the angle of polarization of a light wave passing through them. However, when the material is subjected to an electric field, its molecules change their helix alignment, loosing their light twisting properties.

Ambient light is composed of a virtually infinite number of traveling photon wavefronts with a random variety of transversal orientations. Each of these wavefront orientations, perpendicular to the traveling light direction, corresponds to a polarization direction. When ambient light passes through an infinitesimally narrow slit, only those wavefronts aligned with the slit will pass, resulting in polarized light. A surface made of parallel slits oriented in a particular direction will form a linear polarizing filter. Polarized light is also called coherent light.

When two polarizing filters, the first with an orthogonal orientation with respect to the second, are placed one in front of the other, no light would pass through them. The first filter would allow the passing of light with only one polarization direction. When such a wavefront reaches the second filter, their orientations will not match and therefore no light would pass.

If a thin layer of a liquid crystalline material, such as 4-n-pentyl-4-cyanobiphenyl ($C_{18}H_{21}N$), were placed between the two orthogonal polarizing filters, assuming the layer thickness has been tuned to twist just 90°, the crystal rods in the material would twist the polarization angle of the light coming from the first filter in the same amount, allowing it to pass through the second filter.

By placing transparent patterned electrodes made from Indium-Tin Oxide (ITO) above and below the liquid crystal material layer, it is then possible to control what areas of the liquid crystal will and will not twist light, allowing the creation of the dark and light features we see on a conventional LCD image. Figure 8.36 shows the fundamental structure of a twisted nematic LCD screen.

8.6.2 Control Signals for LCDs

Turning on dark features on an LCD screen is not as simple as just applying a DC voltage to the features' electrodes. When ITO electrodes are subjected to an electrical

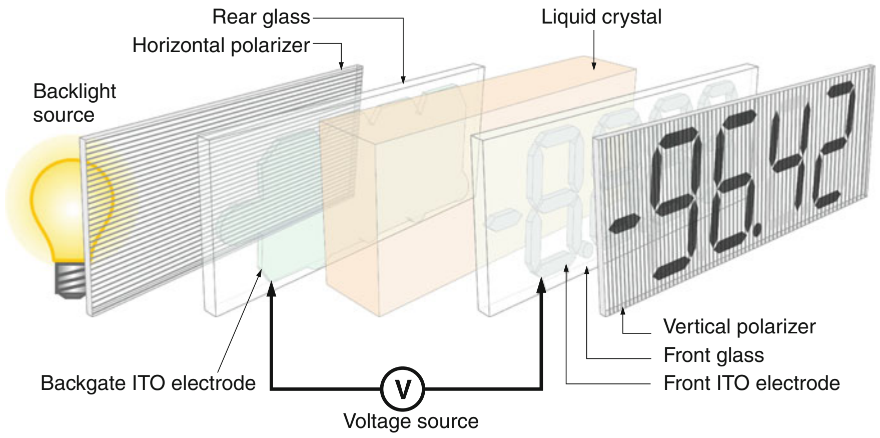


Fig. 8.36 Structure of a twisted nematic LCD

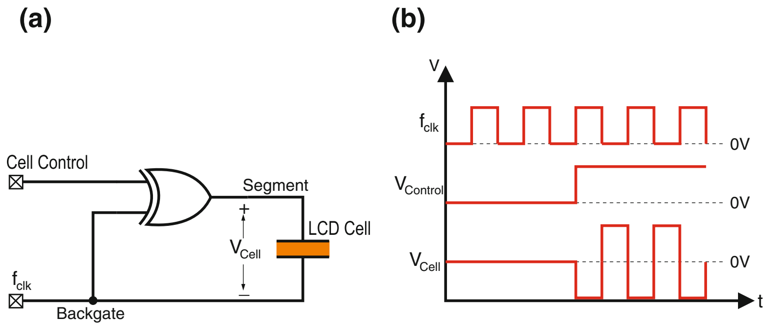


Fig. 8.37 LCD AC drive circuit and waveforms. **a** AC driver circuit, **b** Timing waveforms

field for extended periods of time their chemical characteristics change, loosing their transparency. Moreover, the chemical characteristics of the liquid crystal material itself changes when subjected to permanent DC voltages. A solution to overcome these difficulties is driving the LCD electrodes with an alternating voltage signal with average DC value of zero.

A commonly used technique to produce such an AC signal from a square wave input is illustrated in Fig. 8.37.

Assuming a digital clock signal with 50 % duty cycle, and a CMOS XOR gate, the cell control signal makes the XOR a selective inverter: Control = 0 the gate is a non-inverting buffer, while Control = 1 makes it an inverter. Thus with control in low, the same voltage is applied to both the backplane and cell, yielding a net voltage o zero on the cell. When Control is high, the segment and backplane signals are 180° out of phase, resulting in the cell a $2V_{DD}$ peak-to-peak AC waveform. This way the average DC voltage applied to the cell is zero, avoiding a destructive condition on the cell.

It deserves to note that unlike LEDs and other display elements, liquid crystal cells do not respond to the instantaneous peak value of the applied signal. Instead, they respond to the RMS value of the applied signal. The RMS value of the AC signal controls the contrast obtained in the feature. To have a feature appearing dark requires its cell to be driven with a signal whose RMS value exceeds the LCD threshold for darkness V_{Ton}. A feature will appear clear when the RMS value of the activation voltage is below V_{Toff}. Between V_{Ton} and V_{Toff} there is a transition zone where the feature appears grey.

In the process of driving a cell, it is observed that the AC signal will be actually turning the cell on and off. The human eye will not see the flickering if it happens faster than 24 Hz. For this reason the refresh frequency is specified at 30 HZ or above.

Power consumption in an LCD cell is mainly due to the driver circuit, as the cell itself behaves fundamentally as a capacitor. Thus, the higher the clock frequency the higher the power consumed by the display controller. Considering this fact, to maintain a low-power consumption the refresh frequency is usually set to a value between 30 and 60 Hz.

The complexity of the signalization necessary for driving an LCD screen grows rapidly as the number of segments on the screen increases. Note that besides the control signal, each segment needs to be fed an AC signal with average DC value of zero.

Three major techniques can be identified for driving LCDs, each resulting feasible depending on the total number of segments being controlled on the screen. These include:

- Direct LCD Driving
- Multiplexed Driving
- Active Matrix Driving

8.6.3 Direct LCD Drivers

As its name implies, a direct driver will directly drive each LCD segment with one dedicated line. This approach results cost effective for simple displays with a relatively small number of segments. LCD screens like those used in digital watches and meters typically have less than 36 segments, which make then good candidates for direct driving.

LCD screens designed to be directly driven have a single backplane electrode for all its segments, and a number of control lines equal to the number of segments in the display. Figure 8.38 shows a typical 3.5-digit LCD meter module and its internal connections. The dark area represents the backplane connection, while the light features denote the twenty-eight electrodes required for directly driving the module.

A controller for this module shall provide a 28-bit latched output for equal number of control signals. These, when XORed and referenced to a clock line for the refresh ratio would produce the required AC drive for the module. Implementation alternatives include using GPIO pins and a timer channel. Other alternative is using a dedicated, external direct-drive controller, like National Semiconductor's MM5452. This chip provides a serial connection to a host MCU, requiring only two GPIO lines. It features an internal oscillator for generating the refresh clock, plus 32 undecoded,

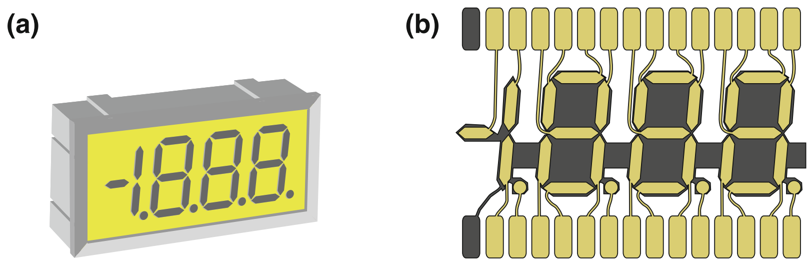


Fig. 8.38 3.5-digit meter LCD module with direct drive electrodes. **a** Module, **b** Electrode arrangement

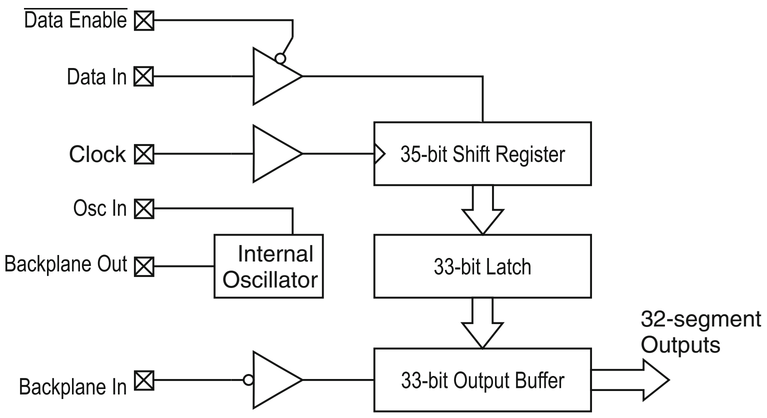


Fig. 8.39 Block diagram of MM5452 LCD direct driver

latched segment outputs that produce an equal number of direct-drive AC segment lines. The chip does not contain a character generator. It can be serially cascaded to allow for driving LCDs with more than 32 segments. Figure 8.39 shows a block diagram of the MM5452.

Despite their disadvantage in the number of segment pins, direct drive LCDs offer the best contrast ratio and viewing angle of all LCD driving techniques.

8.6.4 Multiplexed LCD Drivers

LCD screens featuring a large number of segments, like those used in calculators or in dot-matrix displays, might have hundreds or even thousands of segments. For such displays, the direct drive method results impractical due to the large number of required lines. In such cases, some form of multiplexing approach becomes necessary.

In a multiplexed approach, electrodes are arranged in a matrix form. The backplane electrode is partitioned to form n rows while the segment electrodes are grouped to form m columns. To operate a particular segment located at the crossing of row i and column j, the row and column voltages are set to values V_{BPi} and V_{Sj} respectively. If the segment is to appear dark (active), V_{BPi} and V_{Sj} are assigned values such that the—$|V_{BPi} - V_{Sj}| \geq |V_{Ton}|$, where the vertical bars denote absolute value. For making the segment appear clear, it is necessary to make—$|V_{BPi} - V_{Sj}| \leq |V_{Toff}|$.

Covering a frame requires a time-division multiplexing scheme of n time slots. In each time slot i backplane row BPi is applied voltage V_{BPi} and all columns simultaneously driven with their corresponding voltages segment voltages. The activation of each row in a time slot actually requires two voltage pulses, one positive and one negative, to comply with the requirement of a resultant AC waveform with zero net DC value. Recall that the LCD cell will respond to the RMS value of the AC signal.

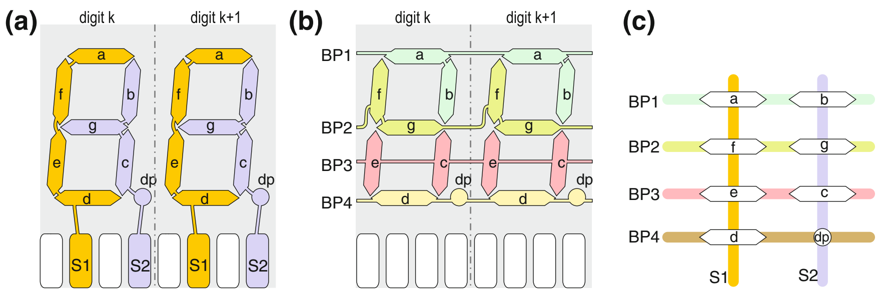


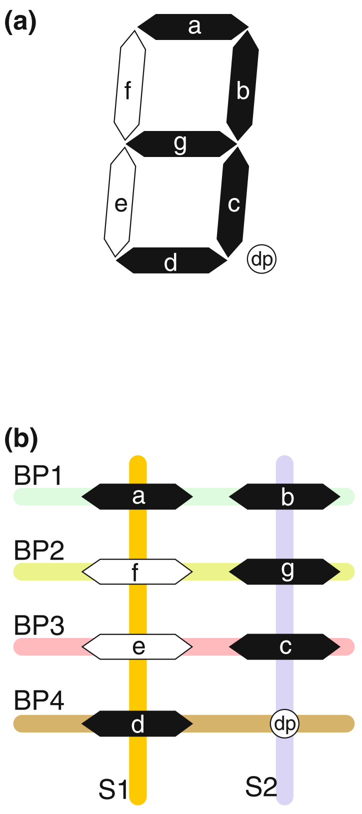
Fig. 8.40 Electrode arrangement for a 4-mux scheme in a 7-segment LCD. **a** Segment electrodes, **b** Backplane electrodes, **c** Digit electrode matrix

Multiplexing schemes are frequently designated as *n-mux*, to denote the number of backplane rows running on the LCD. The most commonly used approaches in seven-segment type LCDs are 2-mux, 3-mux, and 4-mux. Figure 8.40 shows a 4-mux electrode arrangement for a seven-segment plus decimal point LCD. The figure illustrates two consecutive digits. Observe the reduction in the number of used pins: two pins per digit plus four for an entire row. A 3.5-digit LCD under this scheme would require only eleven pins, compared to 28 when using a direct driver.

The signal sequence driving the rows and columns of a multiplexed LCD depends on the multiplexing ratio, the electrode arrangement, and the order chosen for the signal phases, among other factors. In a 4-mux mode for a seven-segment plus digital point LCD each refresh cycle will require 2×4 time steps to cover one screen frame, four positive and four negative. To visualize how the scheme works, let's assume we want to display character "3" in the LCD structure illustrated in Fig. 8.40. Assume the use of four voltage levels, $V1$ through $V4$, to excite the LCD and let's assign the signal phases all positive first and then all negative. Figure 8.41 shows this particular case. Sub-figure (a) shows the character to be displayed, (b) its activation matrix, and (c) the signal sequence applied to rows and columns in each time slot. In this last sub-figure the dashes denote when analog voltage values other than the signal maximum or minimum are applied. The timing diagrams in Fig. 8.41d show the signal waveforms sent to the electrodes. In particular, the last four diagrams show the net voltages seen by the LCD cells corresponding to segments a, f, e, and d.

Signals $BP1$ through $BP4$ activate the backplane one row at a time. Each frame takes eight steps, and then repeats. Column signals S1 and S2 are encoded from the segment activation patterns to determine which segments would come dark and which are light. Notice how the LCD cells see an AC waveform with net DC value of zero. At each time step the voltage in the cell is obtained as the difference between V_{BPi} and V_{Sj}. See how segments a and d have large impulses exceeding V_{Ton}, which will make them dark, while segments f and e never see such large impulses, remaining clear.

In general, the assignment of the number of analog voltage levels and their values for implementing a multiplexing LCD controller needs to consider the particular



(c)

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
BP1	1	–	–	–	0	–	–	–
BP2	–	1	–	–	–	0	–	–
BP3	–	–	1	–	–	–	0	–
BP4	–	–	–	1	–	–	–	0
S1	0	1	1	0	1	0	0	1
S2	0	0	0	1	1	1	1	0

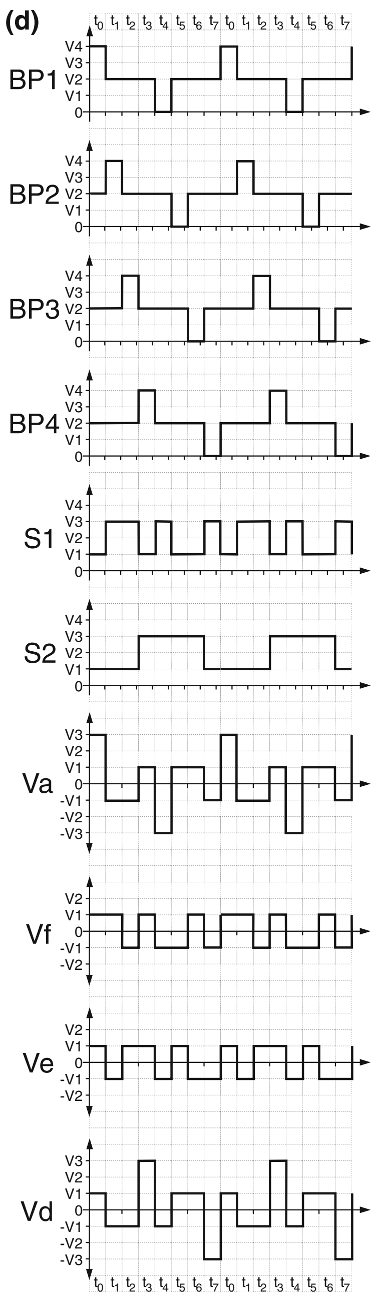


Fig. 8.41 Displaying one 7-segment character in 4-mux format. **a** Displaying character "3", **b** Activation matrix, **c** Signal sequence, **d** Timing waveforms in column S1

LCD characteristics and be optimized for the peak contrast between dark and clear features. Usually, a resistor ladder with as many stages as intermediate voltage steps is connected via an analog multiplexer to supply these voltages. Resistor values are assigned according to the optimal voltage ratios. Moreover, when extended temperature ranges are to be supported, a thermal compensation network is required.

Higher order multiplexing approaches further reduce the pin count in the interface, at the expense of requiring additional time slots to refresh the display and a more intricate signal pattern. As the number of time slots increases, the duty cycle per cell is reduced, resulting in a lower RMS value of the signal applied to each cell. This translates into lower contrast and reduced viewing angle in the display. To worsen contrast, multiplexed LCDs also suffer from cell crosstalk, which is the effect of one cell's activation voltage into its neighbors. In dense dot matrix arrays, crosstalk causes partial activation in the surrounding cells, diminishing contrast.

Another limiting factor taking relevance when the multiplexing ratio increases, is the response time of the LCD cell. Changing the cell state from dark to light or viceversa requires moving the crystal molecules inside the viscous liquid crystal material. This action is by nature slow, taking tens of milliseconds at room temperature, and worsening as temperature lowers. This slow response limits the minimum refresh time per cell and therefore the maximum number of segments that can be driven with via multiplexing.

Improved LCD technologies have encountered limited success in alleviating these problems. For example, super twisted nematic (STN) LCDs, which twist light nearly 270° instead of the 90° obtained in TN devices, have been found to alleviate the loss of contrast in higher-order multiplexing schemes. This technology has stretched multiplexing ratios, reaching up to 480 backplane lines. Despite such an improvement, multiplexed LCDs fall short in brightness and contrast in applications requiring higher line count and faster response time, like in displays used as graphic monitors or in color displays like those used in computer monitors and television sets.

8.6.5 Active Matrix LCDs

One of the most important improvements in LCD technology has been the introduction of the active matrix (AM) cell. An AM LCD incorporates a switching element within each cell, allowing for directly driving charge into the equivalent capacitor of the LCD cell. In today's LCDs, thin-film transistors (TFT) are the most widely used switching elements embedded in the cells. Figure 8.42 illustrates an array of cells, highlighting the cell detail and its electric circuit. A typical TFT in an LCD cell occupies less than 2 % of the cell area, which makes it minimally non-obtrusive for the cell visibility.

The inclusion of switching elements on the screen is achieved by etching the transistors directly on the glass substrate. The circuit formed with the transistor and the liquid crystal (LC) cell acts like a memory cell (notice the similarity with a DRAM cell), making possible to break the dependence of the cell activation duty

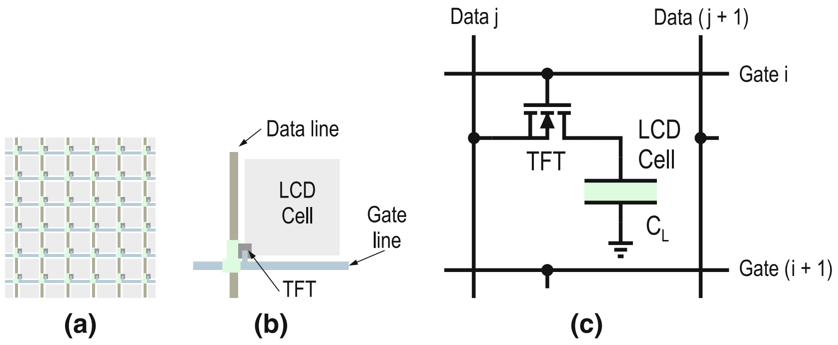


Fig. 8.42 Structure of an active matrix LCD cell. **a** Cell array, **b** Single cell, **c** Circuit

cycle from the size of the array. This cell design allows for using fast electronics to access the transistors while each transistor holds the charge level in the cell for as long as needed by the LC cell to obtain an optimal contrast ratio. Large LCD screens might include an additional capacitor per cell, in parallel to the LC capacitor, to improve charge retention.

The transistors can be operated in their linear region, allowing for precisely controlling the level of voltage applied to each cell. This permits controlling different amounts of light passing through the cell (cell light transmittance), making possible to obtain multiple precise levels of grey on the screen.

Although the construction of a TFT LCD is more complex than that of a multiplexed LCD, the signalization for an active matrix display is simpler. The scan sequence for a frame has two phases. In the first phase all transistors in one row (TFT gate) are simultaneously activated by applying them a positive voltage pulse. At the same time each column is presented with a positive voltage level that establishes the level of light transmittance of each LC cell in that row. At this point, the activation voltage is stored in the cell capacitor. The row voltage is then driven to the low level (a negative value), holding the column voltage in the cell. The same sequence is repeated for the next row. When all rows are scanned, the second phase begins. Here the sequence is repeated with the column voltages reversed with respect to the preceding pass. This way each individual cell is applied an AC signal with zero DC value. Figure 8.43 illustrates the basic signals driving the row and column lines of a particular cell.

The operation of a color TFT LCD is based on the same principle: a matrix of TFT-driven LC cells driven by a row-column scan circuit. The difference in a color screen is that each color pixel is composed of three individually addressable cells, each with one color filter on top, one for each of the three basic colors: red (R), green (G), and blue (B). By modulating the light transmittance of each cell and combining the three basic colors it is possible to produce images with millions of colors on the screen. Figure 8.44 shows the structure of a color LCD screen. Pixels are only a few tens microns in size, so the naked eye cannot see them individually.

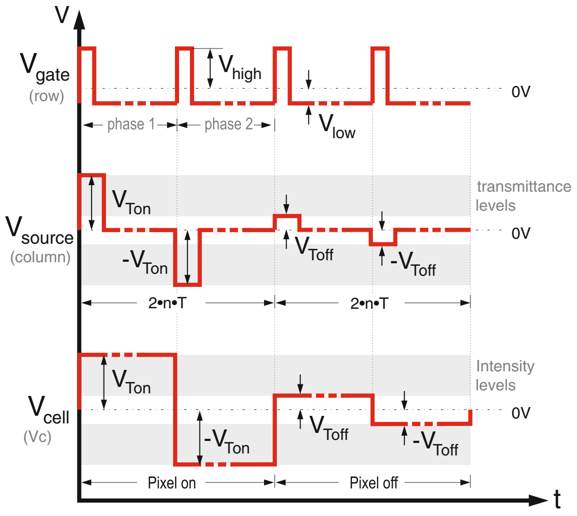


Fig. 8.43 Signal timing in a single active matrix LCD cell

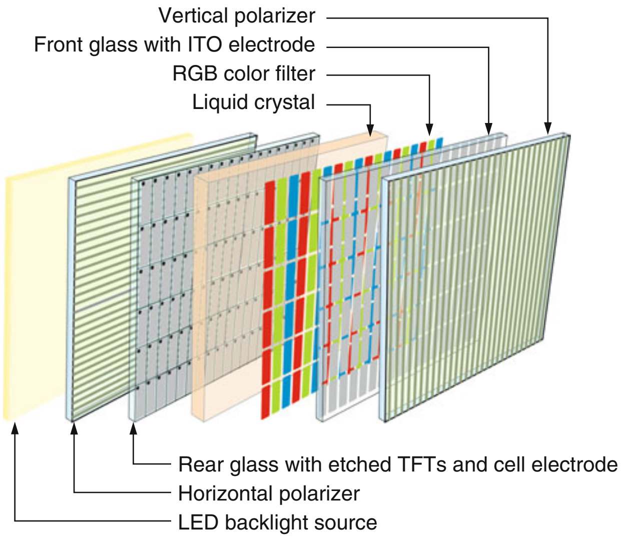


Fig. 8.44 Structure of color TFT LCD screen

8.6.6 Commercial LCD Modules

Despite the complexity in generating the signals to drive the rows and columns of multiplexed and active matrix LCDs, most commercial display modules feature integrated controllers that hide all the hassle associated with driving the display. The

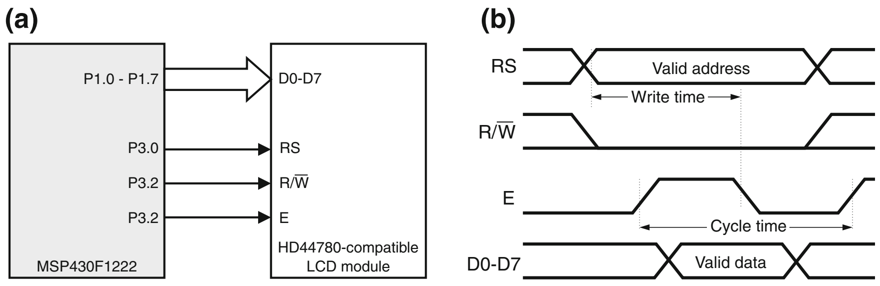


Fig. 8.45 Connecting an alphanumeric LCD module through GPIO lines. **a** GPIO connection, **b** signal timing

controllers' communication protocols provide for direct attachment through either parallel or serial interfaces.

A common characteristic of most LCD controllers is that after a power-up event, the controller must be initialized before it can process display commands or accept display data. The particular initialization sequence and commands used to control the display depends on the specific controller embedded in the module. Below we discuss a couple of representative examples.

Parallel Interface LCD Controllers

These controllers allow connection with 8-bit/4-bit microprocessor buses following either a Motorola 6800 or Intel 8085 signal style. Despite the MPU timing style, it is always possible to generate the control signals from I/O lines, making possible their interface through GPIO lines. Some of these devices allow to select the signal style and/or data width interface while being hardwired or during the command initialization procedure. Examples of these type of controllers include Hitachi HD44780U and compatibles, used for alphanumeric dot-matrix multiplexed LCDs and Samsung's Electronics KS0108B and compatibles, used for 64-mux, bare LCDs. From these, the HD44780-based modules are most widely used due to their embedded character generator and simple interface. KS0108B modules are more oriented to bit-mapped graphic displays, although the controller itself does not provide character generation or graphic support functions. These are typically provided by a host processor.

To illustrate a typical LCD module in more details, consider a Hitachi HD44780-compatible LCD module. The HD44780 is designed to drive dot-matrix type, multiplexed LCDs in alphanumeric mode. The controller features an internal 80-character display RAM, a character generator supporting both western and Japanese kana symbols, and an internal 16 × 40 multiplexed LCD driver. It provides a 6800-type bus interface customizable to operate in 4- or 8-bit mode, but can be connected through an MCU GPIO lines as illustrated in Fig. 8.45a. Line $R/\overline{W}$ could be hardwired to ground as the module can be operated as a write-only device.

Table 8.1 Signals in an HD44780-compatible alphanumeric LCD module

Signal	Type	Description
RS	Control	Command/Data: ‘0’ = Command, ‘1’ = Data
$R/\overline{W}$	Control	Read/Write line: ‘0’ = Write, ‘1’ = Read
E	Control	Enable strobe
$D0-D7$	Data	Bidirectional data lines

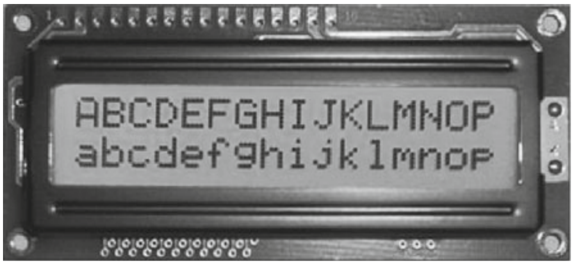


Fig. 8.46 A common 2 × 16 alphanumeric LCD module

The HD44780 control signals are shown in the timing diagram in Fig. 8.45b. A description of each signal is provided in Table 8.1. The timing signals can be readily generated via software in the MCU. Two important considerations to be made in such a connection are: (i) ensuring voltage compatibility of the voltage levels, as some modules might work at a voltage different from the MCU, (ii) verifying that minimum *Cycle* and *Write* times are satisfied. Once the interface is completed it only remains initializing the controller to make it operational. The initialization sequence in the HD44780 requires fundamentally six steps:

1. A warm-up delay of 40 ms or more after power-up.
2. Specify whether an 8-bit (default) or 4-bit protocol will be used.
3. Another delay, this time of at least 4.1 ms.
4. Resend the command in line 2 two more times.
5. A function set command specifying the number of lines and character set.
6. Three commands in sequence,issuing a display off, a clear, and set entry mode.

Figure 8.46 shows a picture of a fairly common 2 × 16 alphanumeric LCD module. The module data sheet provides detailed information on commands and other module features.

Serial Interface LCD Controllers

A growing number of LCD controllers with serial interfaces can nowadays be found in the market. The main attractiveness of these modules is their reduced number of I/O lines to connect with the host MCU, and the advantage of some of these devices in providing embedded support for character generation and some even including basic graphic functions.

Many of the devices are fundamentally adaptations onto multiplexed dot-matrix displays or TFT displays developed from joining a bare LCD module and a custom-designed MCU-based interface that besides driving the LCD provides an external serial interface.

Serial interface options concentrate in fundamentally three types of interfaces: Universal Asynchronous Receiver-Transmitter (UART) compatible, three- or four-wire Serial Peripheral Interface (SPI), or Inter-Integrated Circuit (I2C). Some stand-alone modules using an Universal Serial Bus (USB) can also be found. All these protocols are discussed in Chap. 7 in detail. Aside from the standard serial protocol assumed in these interfaces, their particular command sets allowing to initialize or operate the modules changes from one implementation to another. Some initiatives have begun to emerge providing standardized graphic languages for LCD modules, like 4DGL from the Australian company 4D Systems. These approaches still have limited market coverage and support a limited number of display alternatives. Below we provide a summarized overview of some common serial interfaces LCD modules.

- **UART-based LCD Modules** provide interfaces through the standard RxD and TxD serial lines (plus ground), typically at a default baud rate of 9,600BPS. Command-based interface supports changing the default baud rate and interacting with the pixels on the screen.
- **SPI LCD Modules** with three- or four-line interfaces provide standard synchronous communication using SDI and SDO lines for serial data-in and data-out and SCL for serial clock. Four-wire SPI interfaces also include a chip select (CS) line for module selection. Some modules also provide custom lines for indicating the type of information being fed into the module (command or data) and a dedicated reset line.
- **I2C Modules** adhere to the standard specification of using bidirectional clock (SCL) and data (SDA) lines. Since I2C provides addressing and multi-drop capabilities, multiple modules can share the serial bus. Internal logic differentiates command from data transmissions.

8.7 LCD Support in MSP430 Devices

Some MSP430 devices include dedicated hardware to facilitate interfacing liquid crystal displays to the MCU. In particular the MSP430x4xx and MSP430x6xx include on-chip support for direct drive and multiplexed type LCDs. The on-chip support provides the AC voltage signals required for handling both numeric and dot-matrix type multiplexed LCDs.

The features provided by both MCU series include display memory, signal generation, configurable frame frequency, blinking capability, and support for multiple

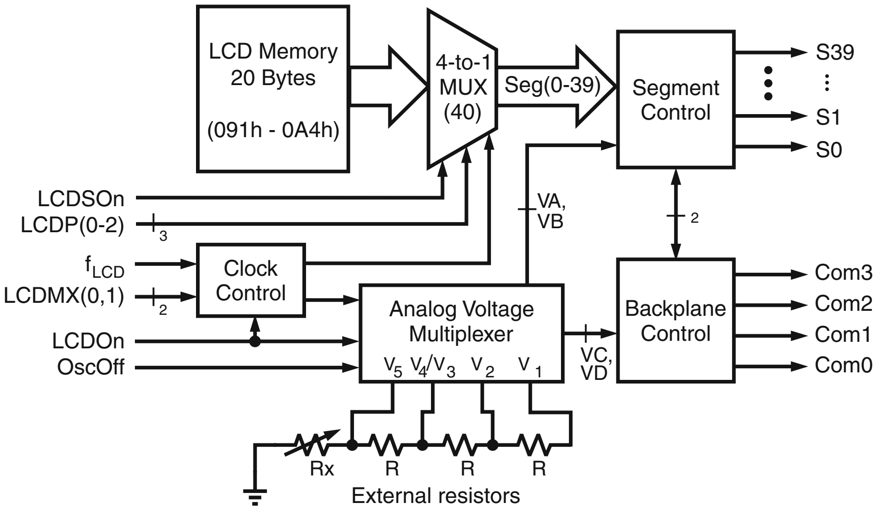


Fig. 8.47 Structure of MSP430 LCD Controller in x4xx MCUs

multiplexing modes. The sections below summarize the salient characteristics in both MCU families.

8.7.1 MSP430x4xx LCD Controllers

MSP430x4xx devices introduced the fundamental structure for supporting liquid crystal displays in MSP430 microcontrollers. The *LCD Controller* in this MSP430 generation was aimed fundamentally at managing seven-segment type LCDs. The controller structure, illustrated in Fig. 8.47, incorporates memory, digital and analog multiplexers, and segment control logic to drive either directly or in multiplexed mode up to 160 segments, depending on the particular x4xx device chosen.

A total of twenty bytes in the address range from 091h to 0A4h are allocated for the LCD memory. The memory contents is mapped assigning one bit per segment for a total of 160 segments. Segment $S0$ is aligned to the least significant bit of byte position 091h.

The multiplexer block features 40 4-to-1 units, used as either 1-to-1, 2-to-1, 3-to-1, or 4-to-1 depending on the multiplexing mode chosen. The segment control block routes the analog voltages applied to the segment electrodes through lines $S0$ through $S39$. Lines $Com0$ through $Com3$ allow application of backplane voltages. $Com0$ is used for static (direct) mode, $Com0$ and $Com1$ in 2-mux mode, and so on.

The LCD controller requires an external resistor ladder network to generate the LCD voltages. The network will contain from one to four 680 KΩ resistors, depending on the selected mux mode. The LCD clock frequency (F_{LCD}) is obtained from

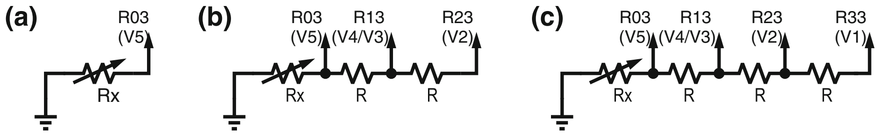


Fig. 8.48 Resistor networks for different mux modes in x4xx devices **a** Static (direct) **b** 2-mux mode **c** 3- and 4-mux modes

Table 8.2 Salient characteristics of x4xx multiplexing mode

MUX mode	LCDMX*x* bits	Max. no. segments	Backplane signals	Memory alignment
Static	00	40	Com(0)	The lsb of each nibble
2-MUX	01	80	Com(0,1)	Two lsb's of each nibble
3-MUX	10	120	Com(0-2)	Three lsb's of each nibble
4-MUX	11	160	Com(0-3)	All bits of each nibble

Timer1. Figure 8.48 shows the network configuration for each mode. Placing a potentiometer at Rx allows for manual contrast control on the LCD.

Operating the LCD requires control bits LCDON = 1 and OscOff = 0 to enable the module. Toggling LCDSON (LCD Segment On) allows blinking all segments simultaneously. Control bits LCDMX(0,1) set the multiplexing mode for the controller as well as voltage levels V_A, V_B, V_C, and V_D, driving segment and backplane electrodes.

Part of the LCD output pins are shared with GPIO pins. The functionality of shared pins is assigned via the I/O port register PxSELx. All LCD controller functions are configured via a single LCD control register (LCDCTL). Salient characteristics of each multiplexing mode are listed in Table 8.2.

MSP430x4xx LCD_A Controller

The latest device entries in the x4xx generation introduced a second on-chip LCD controller named *LCD_A*. LCD_A, whose block diagram is illustrated in Fig. 8.49, is a revised version of the basic LCD controller. It retains the basic functionality and structure of the original LCD Controller while improving in several aspects that include:

- The LCD frequency f_{LCD} is now directly obtained from ACLK, making no longer necessary to use a timer channel. Control bits LCDFreq(0-2) allow choosing a divider for ACLK, setting the value for f_{LCD}.
- An internal LCD Bias Generator including a resistor ladder and a charge pump circuit is used for obtaining voltages V_1 through V_5. The external resistor ladder network required in the previous generation controller is no longer necessary,

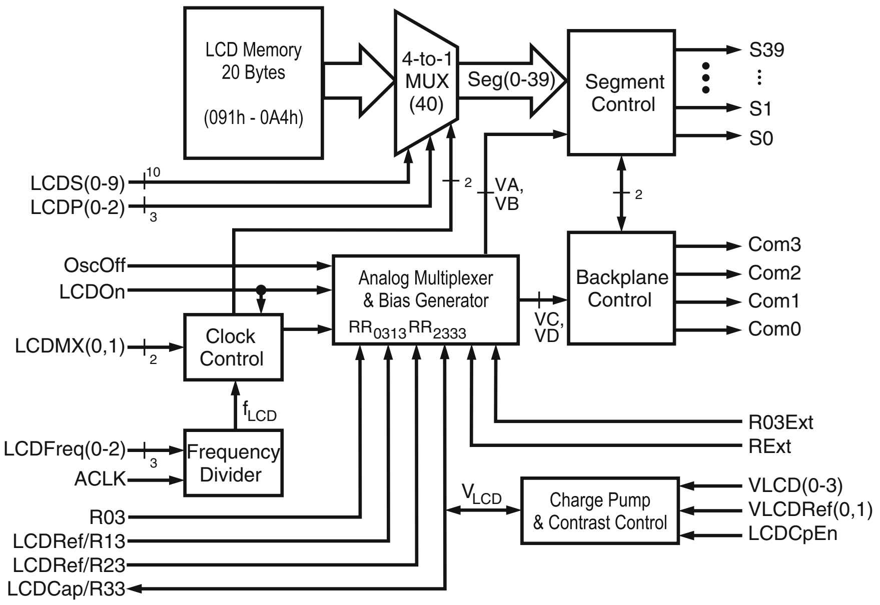


Fig. 8.49 Structure of MSP430x4xx LCD_A Controller

although if desired, it can still be used. The charge pump circuit requires an external 4.7 μF capacitor connected to R33 pin.

- An on-chip contrast control circuit now makes possible to adjust the LCD contrast ratio via software. Seven different contrast levels are possible, plus an eight level where segments are off.
- An enhanced, more flexible segment enable control logic is now included, allowing activation of segments in increments of four, starting at *S*0.

Due to the additional functionality embedded into the controller, configuring LCD_A takes now five registers: one LCD_A control register (LCDACTL), two LCD_A port control registers (LCDAPCTL(0,1)), and two LCD_A voltage control registers (LCDAVCTL(0,1)). The reader is referred to the MSP430x4xx Family User's Guide (SLAU056J), available at TI's MSP430 MCU home page for programming details.

8.7.2 MSP430x6xx LCD Controllers

The x6xx generation of MSP controllers feature two different LCD controllers: LCD_B and LCD_C. Both, like previous generations, continue aimed at servicing multiplexed LCDs, while introducing several new enhancements.

MSP430x6xx LCD_B Controller

The LCD_B Controller inherits most characteristics of LCD_A controller, supporting static, 2-mux, 3-mux, and 4-mux modes, and up to 160-segment LCDs. Like its predecessor, it also provides on-chip LCD bias generation, software controlled contrast ratio, and an internal clock divider that does not require a dedicated timer channel. In addition, LCD_B features several improvements that include:

- An expanded frequency divider for generating f_{LCD} that now allows using VLOCLK as clock source, in addition to ACLK.
- Support for a segment blink capability that allows blinking individual segments. This is achieved by adding a second 20-byte memory block used for indicating the blinking status. Each bit in the blinking memory indicates the blinking status of its homologous bit in the segment memory. The blink frequency is provided by a dedicated blinking frequency divider fed from the same source as the LCD frequency divider.
- The blinking memory can double as a display memory if not used for blinking purposes. When used as a secondary display memory, it allows to switch under software control between the two buffers by solely changing the LCDDISP bit in the memory control register (LCDBMEMCTL).
- Four interrupt sources (LCDFRMIFG, LCDBLKOFFIFG, LCDBLKONIFG, and LCDNOCAPIFG) served by a single interrupt vector. Each source has an independent enable and status flag that allows to determine the triggering source. LCDFRMIFG can be configured to trigger when a frame boundary is reached. LCDBLKOFFIFG and LCDBLKONIFG can be individually set to be triggered by the blink clock (BLKCLK) when a frame boundary that either turns off or on the segments is reached. The fourth source, LCDNOCAPIFG can be enabled as a warning service to detect when the LCD biasing charge pump has been enabled without an external capacitor attached to the LCDCAP pin.

Configuring LCD_B is achieved through a set of eleven 16-bit registers that include two for general control (LCDBCTL(0,1)), one for blinking control (LCDBBLKCTL), one for memory control (LCDBMEMCTL), one for voltage control (LCDBVCTL), four for port control (LCDBPCTL(0-3)), one for controlling the charge pump (LCDBCPCTL), and one for interrupt control (LCDBIV). All control registers are reset by a PUC event.

MSP430x6xx LCD_B Controller

The LCD_C Controller is, so far, the most powerful LCD controller embedded into MSP430 devices. LCD_C supports all features exhibited by its predecessor, LCD_B, and expands to support up to 8-mux mode LCDs. This allows the enhanced controller to drive display modules with up to 320 segments. This expanded segment driving capability makes LCD_C able to drive small dot-matrix displays with as many as eight 5×7 characters plus a nonblocking cursor. In terms of standard seven-segment characters, the controller can drive up to forty digits with decimal points.

Such an expanded multiplexing capability imposes limitations to the contrast attainable on the screen. In the case of 8-mux mode, the best voltage contrast ratio achievable is 1.446, a reduction of 16.5 % with respect to the best 4-mux mode contrast. This is still a viewable contrast for most LCD modules, but keep in mind that the higher the multiplexing ratio the lower the maximum attainable contrast.

Additional configuration and operation details for LCD_B and LCD_C controllers can be found in the MSP430x5xx/MSP430x6xx Family User's Guide available from Texas instruments in its MCU web page.

8.8 Managing Large DC Loads and Voltages

Managing direct current (DC) loads rated at voltages or currents beyond those directly manageable by an I/O pin require level-shifting and/or current buffering interfaces. These interfacing requirements can be met either using discrete components of packaged driver solutions. Here we discuss representative approaches.

8.8.1 Using Transistor Drivers

Perhaps one of the most common interfaces for managing DC loads is based on a transistor buffer. A transistor buffer not only allows to boost the amount of current that can be controlled by an I/O pin, but also allows for driving a load at a voltage different than that feeding the MCU. Figure 8.50 shows generalized topologies with bipolar (BJT) and field-effect (MOSFET) transistors used as buffer-driver.

In the cases illustrated, a generic DC load is being driven in subfigure (a) by an NPN BJT and in (b) by an N-channel MOSFET. In either case the transistors are turned-on with a logic-high voltage V_{OH} in the I/O pin. Diode D_C is optional for resistive loads, however it must be included when inductive loads are switched to protect the transistor from overvoltages due to the transient when the inductor is turned off.

The buffer topology can also be implemented using PNP or PMOS transistors to drive the load. The main consideration for such a change is that voltage polarities and current directions are reversed with respect to those in N-drivers, thus requiring a logic-low level voltage to turn the transistor on.

The MCU and load supply voltages could be the same ($V_{DD1} = V_{DD2}$) or different. We could, for example, drive a 12VDC load with a 3.3 V MCU. As long as we can provide enough base or gate voltage to turn on and off the transistor, the interface would work.

When a transistor is used to interface a binary load (ON/OFF), as is the case here, we want the transistor to operate as a switch, regardless of its use as a current

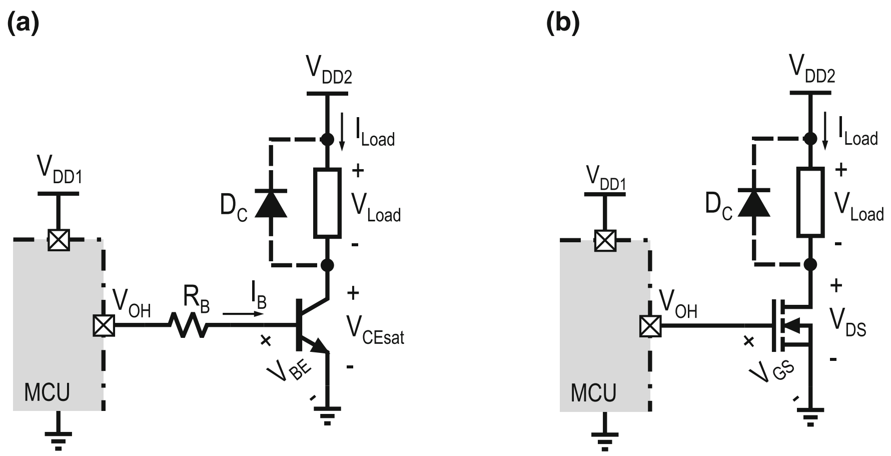


Fig. 8.50 Generalized topology of a transistor buffer using N-devices. **a** NPN buffer, **b** NMOS buffer

buffer, level-shifter, or both. If the interface is based on a BJT, we need to ensure the base resistor allows enough current into the transistor base to saturate it with $I_{C(sat)} = I_{Load}$. When a BJT saturates it exhibits a low V_{CEsat}, typically around 0.1 or 0.2 V, allowing virtually all the load supply voltage to drop across the load. To keep the base current at the minimum required, it is advised to keep the BJT at the verge of saturation.

In the case of a MOSFET driver the criteria is very similar to the above. The MCU side of the interface does not need a gate resistor, as seen in Fig. 8.50b, as MOSFETs are activated by voltage, not current. The gate voltage $V_{GS} = V_{OH}$ shall allow for placing the NMOS in the linear region with $I_D \geq I_{Load}$. This way, its voltage drop V_{DS} would be small, allowing for most of V_{DD2} to drop across the load. In a MOSFET, the maximum current in linear mode is obtained near pinch-off, but that would require $V_{DS} \approx (V_{GS} - V_{th})$, which might be too large a V_{DS}. Typically a mid-linear region value provides a better V_{DS} value.

The main criteria for choosing the BJT (MOSFET) to be used for a particular driver/buffer are:

Current The transistor must be able to handle the load current. In the case of a BJT (MOSFET) the maximum collector (drain) current I_{Cmax} (I_{Dmax}) must be larger, at least 15 %, than the load current.

Voltage The maximum V_{CE} (V_{DS}) would develop when the transistor is off. Thus V_{CEmax} (V_{DSmax}) must be large enough to accommodate V_{DD2}.

Power The transistor will dissipate power when conducting. Therefore, the device maximum power dissipation must satisfy $P_{Cmax} \geq V_{CEsat} \times I_{Load}$ ($P_{Dmax} \geq V_{DS} \times I_{Load}$). A safe margin of at least 15 % above it shall be allowed.

BJT drivers are usually faster than equivalent MOSFET drivers and require smaller control voltage (V_{BE}) to be switched, but as they are activated by current I_B, and in

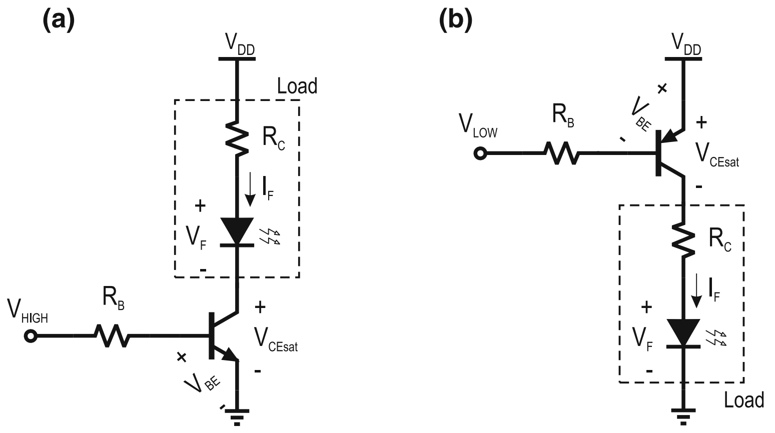


Fig. 8.51 Using bipolar transistors to buffer high-current LEDs. **a** NPN buffer, **b** PNP buffer

saturation β is small, large I_C values might lead to I_B values exceeding the GPIO capability. In such cases, multi-level drivers, like a Darlington pair could become necessary. Example 8.6 on the facing page illustrates one such case. The advantage of a Darlington pair is its large current gain $\beta_{eq} = \beta_1 \times \beta_2$, where β_1 and β_2 are the current gains of its two transistors. The pair, however, requires a larger control voltage ($V_{gs(eq)} = 2 \cdot V_{BE}$) and higher $V_{CE(eq)}$ as the output driver does not actually saturate.

A situation exemplifying the use of a transistor driver is the case of a light emitting diode whose I_F exceeds the port driving capability. A transistor buffer would offer a simple solution. For such a case, the transistor is chosen to handle the LED current. Figures 8.51 and 8.52 show a few possibilities. The topologies in Fig. 8.51 show how to solve the problem using a bipolar transistor.

Using the transistor data, we first choose R_C to limit the current passing through the diode and to fix its forward drop V_F. Taking as reference the NPN-based buffer, the value of R_C can be obtained as:

$$R_C = \frac{V_{DD} - V_F - V_{CEsat}}{I_F} \tag{8.24}$$

Assuming the BJT is at the verge of saturation, we use the minimum value of β as the saturation current gain β_{sat}. Some manufacturers provide explicit values of β_{sat} in their data sheets. This allows obtaining $I_{Bsat} = I_F/\beta_{sat}$ which then leads to calculating R_B as:

$$R_B = \frac{V_{HIGH} - V_{BEsat}}{I_{Bsat}} \tag{8.25}$$

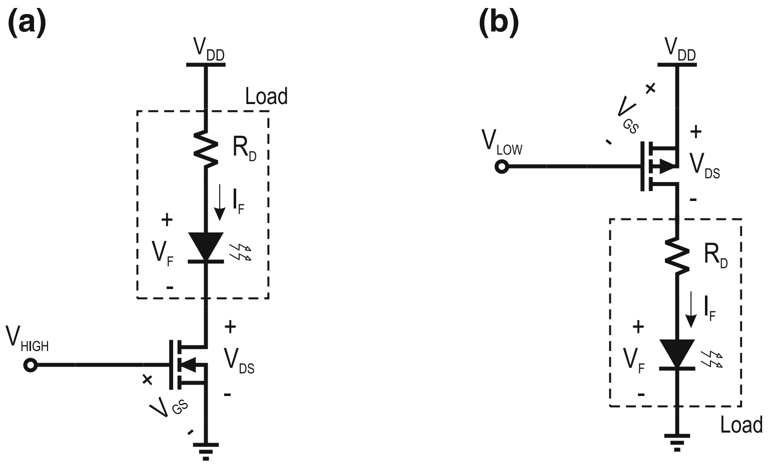


Fig. 8.52 Using MOSFETs to buffer high-current LEDs: **a** an NMOS buffer, **b** a PMOs buffer

The process for obtaining the values of R_C and R_B for the PNP buffer in Fig. 8.51b is fundamentally the same, except that now voltages and currents are reversed with respect to the NPN case.

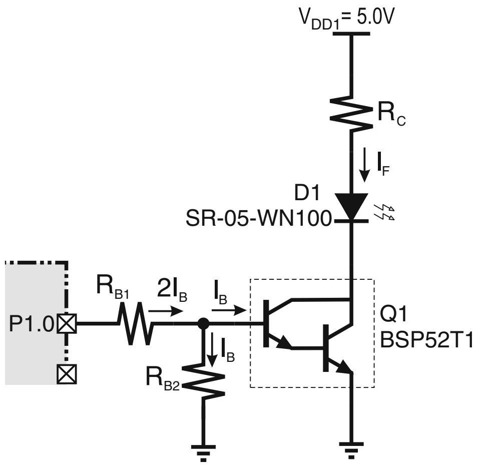
A final note about the transistor buffers is that we typically connect the load such that the conduction biasing is not affected by voltage fluctuations in the load. In the case of BJTs the load is connected to the collector. Connecting the load to the emitter would introduce feedback in the V_{BE} voltage that could take the transistor out from saturation. Due to similar reasons, in MOSFETs, the preferred terminal is the drain.

The value of R_D is obtained considering the MOSFET biased in the linear region close to pinch-off for a small V_{DS}. Obtaining the equations for R_D is left as an exercise.

Example 8.6 *A LED-based flash for a smart phone camera is to be driven by an MSP430 MCU. The flash is designed to produce a 1/60 s burst of light when the camera shutter is depressed, if the subject is too dark. The LED, a Luxeon SR-05-WN100 part, requires a forward voltage of 3.2 V and 700 mA to produce a 180 Lumens output. Assume a boost converter elsewhere in the camera provides a 5VDC/1.5 A supply for the flash, and a light sensor, already embedded in the camera provides the indication of darkness. Design a suitable interface to trigger the flash from an I/O pin of the MSP430F1222.*

Solution: *For this application, the load current requirement of 700 mA cannot be directly provided by the MCU through an I/O pin. The MSP430F1222 data sheet indicates that at room temperature, to maintain V_{OH} compliant with $V_{IHmin} = 80\,\%V_{CC}$, I_{OH} shall not exceed 20 mA. A typical solution with a single stage driver results in a $\beta_{sat} \simeq 10$, yielding $I_{Bsat} \gg 20$ mA. For this reason, the recommended buffer in this case shall be a darlington pair. A suitable driver in this case would be an ON Semiconductor's BSP52T1. Rated at $I_{Cmax} = 1.0$ A in a convenient surface-mount*

Fig. 8.53 Darlington-based LED buffer



package and $V_{CEmax} = 80\,V$, this driver comes out as a convenient choice. It's data sheet specifies that at $I_{Csat} = 700\,mA$ it features a $V_{CEsat} = 1.0\,V$ and $\beta_{sat} \simeq 1000$. Figure 8.53 shows a schematic of the LED connection.

Note that in this case, feeding the LED with $V_{DD} = 3.3\,V$ will not allow for having a $V_F = 3.2\,V$ in the LED due to the required $V_{CEsat} = 1\,V$ of Q_1.

Applying Eq. 8.24, the value of R_C would result in approximately 1.2 Ω at 0.5 W. For $\beta_{sat} = 1000$ the base current would be 0.7 mA. At the darlington's base it results convenient the addition of resistor R_{B2} to accelerate the transistor turn-off time. Otherwise it would turn-off via the I/O port leakage, which might result too slow if pulsed operation were desired. In this case we would want to have similar turn-on and turn-off times, so we will set the current through R_{B2} to be the same as I_B, making the current through $R_{B1} = 2I_B = 1.4\,mA$. According to the BSP52T1 data sheet, for $I_{Csat} = 700\,mA$ the turn-on voltage V_{BEsat} would be 0.95 V, allowing to obtain the values of R_{B1} and R_{B2} as:

$$R_{B1} = \frac{V_{OH} - V_{BEsat}}{2I_B} = \frac{3.1\,V - 0.95\,V}{1.4\,mA} \simeq 1.5\,K$$

$$R_{B2} = \frac{V_{BEsat}}{I_B} = \frac{0.95\,V}{0.7\,mA} \simeq 1.4\,K$$

8.8.2 Using Driver ICs

Some applications might need more than just a few buffering interfaces. When many lines need to be interfaced for either boosting current or level-shifting, using discrete transistors might become cumbersome. For such cases, IC drivers become an attractive alternative.

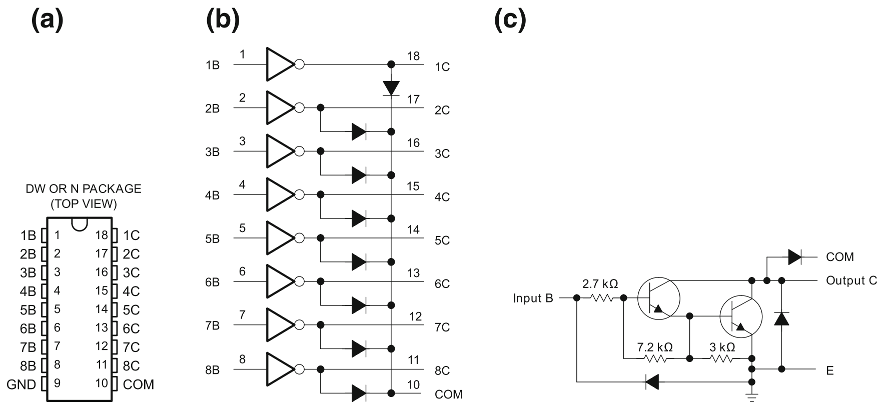


Fig. 8.54 TI's ULN2803A Darlington transistor array (*Courtesy of Texas Instruments, Inc.*) **a** Package, **b** Logic diagram **c** Internal circuit

IC drivers can accommodate several individual buffers in a single package, simplifying interfaces that require handling multiple lines. Examples of such ICs include Texas Instrument's ULN2803A and similar devices offered by other semiconductor manufacturers. TI's chip offers eight independent Darlington transistor arrays in a convenient 20-pin package. Figure 8.54 reproduces a package top view, internal logic diagram, and driver schematic of the chip.

Some of the features of this IC include a driving capability of up to 500 mA per driver at voltages as high as 50 V, internal clamping diode to manage inductive loads, and fast response time. The reader is referred to the device's data sheets for a complete list of features and specifications.

8.9 Managing AC Loads

Unlike DC loads, alternating current (AC) loads cannot be directly driven by I/O pins. In this case an isolating interface is recommended. The simplest form of controlling AC loads from a GPIO is through a relay.

8.9.1 Mechanical Relays

A relay, in its most classical form is an electromechanical device where an electromagnet is used to change the status of a mechanical switch. This allows controlling the mechanical status of the switch contacts via an electrical signal. Relays can be considered the precursors of transistors in the sense of acting as an electrically controlled switch. Figure 8.55 shows the structure of an electromechanical relay and its schematic symbol.

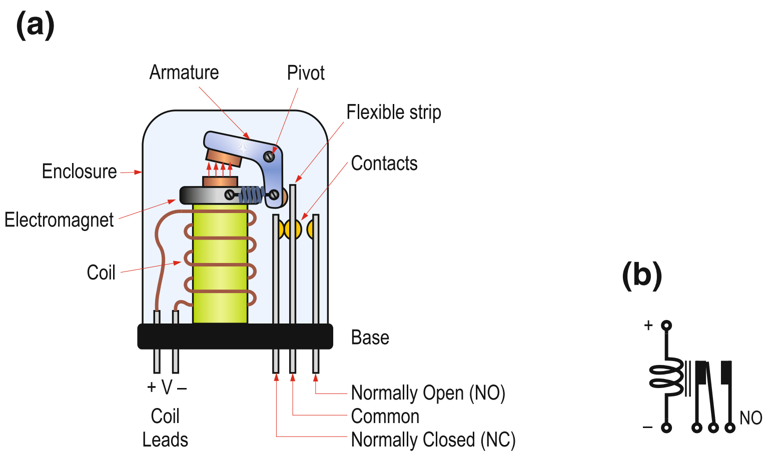


Fig. 8.55 Electromechanical relay with SPST NC/NO contacts, **a** Structure, **b** Symbol

A voltage applied to the coil terminals energizes the electromagnet, which mechanically moves the armature, changing the contacts state: NO contacts close while NC open. The relay remains in this state until the coil is de-energized. Contacts are enclosed to reduce ambient induced deterioration. A relay is considered an isolating interface as the contacts and coil are physically separated.

The specifications of a relay minimally provide the contact and coil ratings:

Coil Voltage: Nominal voltage necessary in the coil in order to cause the armature to flip. This can be either an AC or DC voltage.

Coil Current: Amount of current in the coil circuit when the relay is activated. Some manufacturers provide instead the coil impedance, which could readily yield the current once the coil voltage is known.

Contact Current Rating: Maximum amount of current the contacts can withstand in a permanent regime. Exceeding this value will accelerate contact deterioration. This must be selected to safely handle the load maximum current.

Contact Voltage Rating: Magnitude of the maximum voltage an open contact can withstand. The contact voltage can be either AC or DC, regardless of the specified coil voltage. This specification must be large enough to cover and safely exceed the load voltage.

Other relay specifications include their response time (time to change contact state upon a valid coil input), and operating lifetime (expected number of contact operations before failure).

Mechanical relays require a certain amount of current in their coils to be activated. Small DC relays could be directly connected to GPIO pins if their coil current and voltage specs do not exceed those of the pin. Note that relay coils are inductive loads and therefore any control interface must include a clamping diode to protect the pin buffer. The response time of a mechanical relay is large, typically dozens of milliseconds.

A preferred method for driving relay coils is through a buffering transistor, in a topology similar to those illustrated in Fig. 8.50a, b, including the clamping diode D_C.

8.9.2 Solid-State Relays

Solid-state relays (SSR) replace the electromagnet and mechanical contacts of traditional relays with some form of silicon controlled rectifier (SCR), typically a triac or an arrangement of power MOSFETs, eliminating all mechanical parts in their operation. The coil circuit in an SSR is also modified, being replaced by an optical switch that triggers the SCR gate. These modifications make solid-state relays more durable and faster than their electromechanical counterparts.

SSR are designed to be replacement for electromechanical relays, therefore both types have similar interfacing and specification requirements. In the load side, like their counterparts, SSR can switch large AC and DC loads. Given the electronic nature of their control circuits, SSRs can be controlled with lower currents and voltages in the MCU side than mechanical relays, therefore becoming more appropriate for embedded systems interfaces.

Figure 8.56 shows a simple way to interface an SSR to an MCU for controlling an AC load. The major internal SSR components are also denoted to better understand the interface. The SSR input fundamentally connects to the SSR internal opto-switch. The load is driven by a triac, serving as switching element. An internal zero-crossing detector is used to keep the triac on while the SSR input is active. The relay in the figure corresponds to a *zero-switching* SSR as it would turn-on or -off the load at the next point the AC signal crosses zero after the input has been commanded.

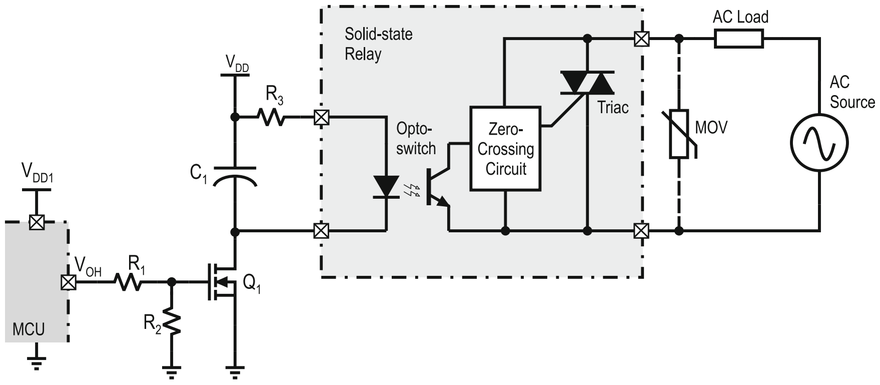


Fig. 8.56 Solid-state relay: internal structure and MCU interface

The SSR input circuit is driven by an NMOS buffer that allows for driving the current and voltage requirements of the SSR input. Divisor R_1/R_2 allows setting the desired V_{GS} at Q_1. R_2 also helps accelerating turning-off Q_1. In the case V_{OH} satisfies the SSR input voltage requirement and the pin current were able to drive the opto-switch, R_1 and Q_1 could be omitted. Resistor R_3 is used to limit the current through the opto-switch LED. Capacitor C_1 serves as a filter to reduce the amount of power supply noise reaching the SSR input.

In the output, the SSR terminals are in series with the AC source and load. The optional Metal Oxide Varistor (MOV) in parallel to the SSR output protects against surges in the AC line. The values of all external components will be dependent on the particular SSR being used and the MCU controlling the circuit.

8.9.3 Opto-Detectors and Opto-Isolators

Opto-detectors and opto-isolators are based on the same principle: the ability of semiconductors to generate an electrical current when bombarded by energetic particles like photons or other forms of radiation. When the depletion region of a *pn* junction is exposed to sufficiently energetic photons, electrons are liberated from the material, producing an electrical current. This is the inverse phenomenon that generates light in an LED. A *pn* junction exposed to light becomes a photodiode.

Photodiodes are typically used in reverse bias. When light hits a reverse biased *pn* junction, a reverse current directly proportional to the intensity of the incident light is generated. This principle is used to build light detectors and other light operated devices. Solar cells offer an example of a large area photodiode where the current generated by the photoelectric emission is collected and used as energy.

A phototransistor is a BJT with the collector-base junction exposed to light. The electrons liberated by the photoelectric emission in the collector enter the base region, where they are amplified by the transistor current gain factor (β_F).

In microcontroller applications, photosensitive devices like photodiodes and photo-transistors are frequently paired with light-emitting diodes to form optical switches. An optical switch operates by detecting the current changes caused in a photodiode or phototransistor by the blockage or exposure to the LED light.

Photodiodes can be tuned to respond to light of specific wavelengths. Their pairing with LEDs of the same wavelength, allows for developing photo-switches able to discriminate light from undesired sources. For example, a TV remote control uses infrared LEDs and photodiodes in the transmitter and receiver, respectively, to only respond to the excitation produced by the transmitter and not to ambient light. Moreover, pulsing the LED on the transmitter allows serially encoding bit patterns into streams of light/darkness impulses that are decoded as zeros and ones at the detector side. Figure 8.57 shows an LED-phototransistor pair forming an opto-coupler and a possible form to interface it to an MCU.

Resistor R_1 limits the LED current, while R_2 establishes the maximum current through the phototransistor. Setting low the output port driving the LED makes it

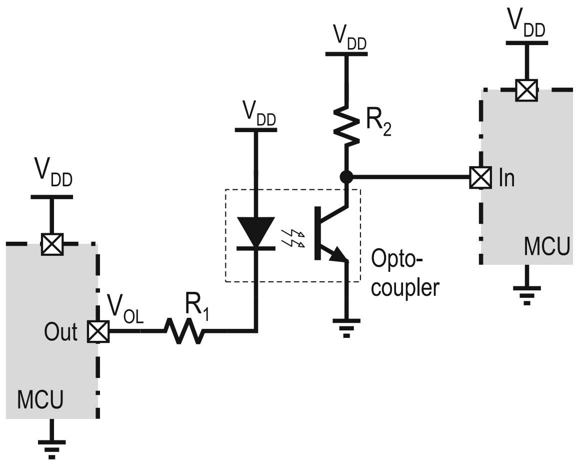


Fig. 8.57 Interfacing an optocoupler to an MCU

conduct, shining light on the phototransistor. When the phototransistor is exposed to light, it begins to conduct, bringing to low the voltage at node x. This condition can be read by an input GPIO pin. The values of R_1 and R_2 are obtained using actual application conditions.

8.10 MCU Motor Interfacing

A number of applications in embedded systems require precise mechanical movement. A few common examples include plotters, inkjet printers, and CNC (Computer Numerical Control) machines where precise movement under computer control is needed. The list of alternatives to produce computer controlled movement include DC motors, servo-mechanisms, and stepper motors. All three can be defined as electro-mechanical devices capable of transforming electric power into mechanical power in the form of a rotating shaft. However, there are fundamental differences among them in terms on how the electro-mechanical conversion is made and particularly on how they can be controlled.

8.10.1 Working with DC Motors

The shaft of a DC motor continuously spins when applied energy. Assuming a constant mechanical load, the output mechanical power of a DC motor will be proportional to the input electrical power, up to its rated capacity. Thus at constant load, the motor speed would be a function of the applied voltage.

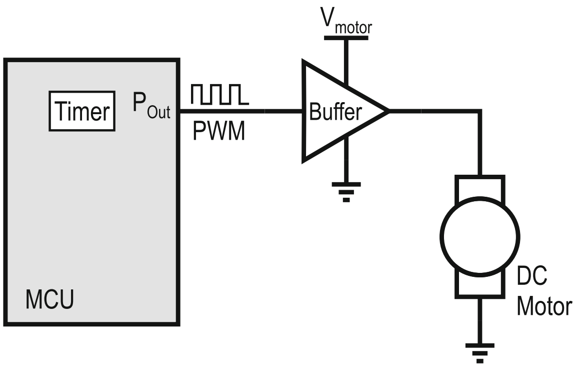


Fig. 8.58 Open-loop DC motor speed control via PWM

Applications where only the spinning speed is important can be readily served with DC motors. A variable speed electric fan is a representative example of this type of application. The spinning speed of a DC motor can be readily controlled from a microcontroller using pulse-width modulation (PWM). A pulse-width modulated signal is a square wave whose duty cycle can be controlled by the MCU. The amount of power carried by a PWM signal is proportional to the waveform duty cycle.

Generating a PWM signal using an MCU results straightforward with an internal timer and Output pin. Thus, controlling the speed of a constant load DC motor from an MCU is a simple task. Figure 8.58 shows a diagram of an open-loop DC motor speed control using PWM that can achieve such an objective. The buffer in the figure can be implemented as discussed in Sect. 8.8. Chapter 7, in Sect. 7.4.3 offers a discussion on how to produce a PWM signal with an MCU.

If a reversing control were desired in the DC motor interface, an *H-Bridge* driver could be employed. An H-bridge contains two pairs of complementary transistor that allow for reversing the voltage applied to a load under the control of an MCU output pins. Figure 8.59 shows a basic topology for a reversible DC motor H-bridge driver implemented with bipolar transistors.

The circuit is controlled with complementary inputs *Fwd* and *Rev*. Setting $Fwd = 1$ and $Rev = 0$, transistors $Q_1 - Q_3$ are simultaneously turned on, allowing current flow from left to right in the motor, as indicated by the dash-dot line in Fig. 8.59. Making $Fwd = 0$ and $Rev = 1$, will activate transistors $Q_2 - Q_4$ instead, reversing the current direction and therefore the polarity of the voltage applied to the load. Diodes D_1 through D_4 protect their respective transistors from the inductive turn-off transient. Inputs *Fwd* or *Rev* could be driven by a PWM signal to also allow for speed control. The circuit is also frequently implemented with MOSFETs instead of BJTs, allowing similar functionality.

Although the basic functionality of an H-bridge can be obtained using discrete devices, IC-level solutions offer a better alternative. Consider for example, TI's DRV8833, a dual CMOS H-bridge motor driver IC. This chip encapsulates two full H-bridges able to handle a continuous 3 A load at up to 10.8 V while providing short circuit protection, thermal shut-down, and supporting low-power modes.

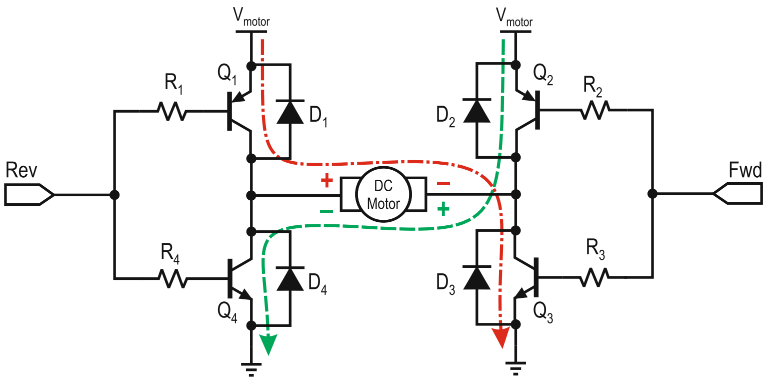
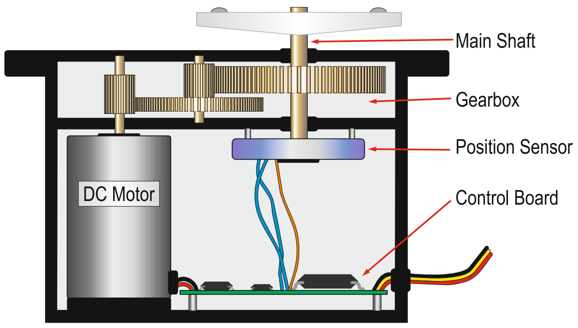


Fig. 8.59 Reversible speed H-bridge DC motor driver

Fig. 8.60 Pictorial view of internals of a small servomotor



Using a basic DC motor with an open-loop PWM speed control results fairly appropriate in applications where speed control is not tight and loading conditions can be considered constant. If instead, a more precise control of the rotational speed were needed or if accurate positional control were needed, the DC motor control would require to include feedback and some form of control strategy.

8.10.2 Servo Motor Interfacing

A servo-motor is a DC motor with feedback control that allows for precise position control. A servo-motor includes four basic components: a DC motor that provides the basic electromechanical conversion, a control board housing the feedback electronics, a set of gears that slow-down the DC motor rotational speed, and a position sensor, typically a potentiometer. Figure 8.60 shows a pictorial diagram of the components in a servomotor.

The external shaft of a servo motor does not continuously spin like that in a DC motor. Although its internal motor may make multiple turns at once, the gearbox reduces the rotation ratio resulting in a restricted travel angle of about 200° or less (typically 180°). The positional sensor is attached to the main shaft axis, providing feedback positional information to the control circuit, which implements a PID feedback controller, allowing precise shaft angular position control.

The external interface of a servo motor has only three wires: V_{DD}, *GND*, and *Control*. Power is applied trough the $V_{DD} - GND$ terminals while the desired position is specified through the *Control* pin with a PWM signal. The period of the PWM signal is typically 20 ms ($f = 50$ Hz), and the width of the ON portion of the signal specifies the desired angular position. Specs might slightly change depending on the shaft travel angle, but typical specs define 1000 μS to 2000 μS for the extreme positions (−90° to +90°), with 1500 μS for center position. Interfacing to an MCU requires a single PWM output, and the control pin in most cases can be directly driven from an I/O pin or simply a level shifter.

Servo motors come in a wide variety of sizes, from small units used in radio-controlled models, to industrial sizes, in DC or AC versions moving heavy equipment.

8.10.3 Stepper Motor Interfaces

A stepper motor shaft moves in discrete increments in response to digital pulse sequences applied from a controller. The structure of a stepper motor is different from that of a DC motor, as it incorporates multiple windings to make possible the stepping behavior. Depending on how the rotor and stator are designed, stepper motors are classified into three types: variable reluctance, permanent magnet, and hybrid.

8.10.4 Variable Reluctance Stepper Motors

A variable reluctance (VR) stepper motor has a soft iron, non-magnetized, multi-toothed rotor and a wounded stator with three to five windings. The motor is designed with a larger number of poles in the stator than teeth in the rotor. Figure 8.61 shows a sectional view of a VR stepper with four windings, A, B, C, D in the stator (eight poles) and six teeth in the rotor.

The stator windings in the motor are unipolar, meaning that they can be polarized in only one direction. Torque is developed as poles and teeth seek to minimize the length of the magnetic flux path between the stator poles and rotor teeth. The sequence and combination of pole activation determines the rotation direction and step size. Two different step sizes can be produced: full step, and half step. In full step mode the motor will advance in angular increments equal to its nominal resolution. In half-step mode the resolution is doubled as the motor advances only half the nominal angular resolution.

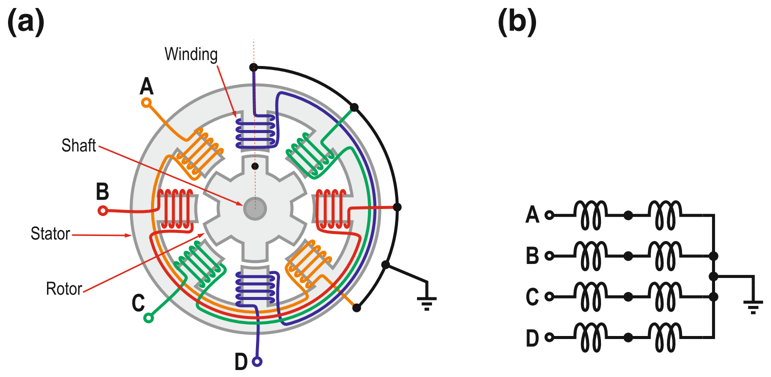


Fig. 8.61 Sectional view of a variable reluctance stepper motor and winding arrangement. **a** Motor cross section, **b** Winding arrangement

Full-step resolution can be achieved two ways: using a single phase sequence (also called wave sequence) or using a two-phase sequence (routinely called full-step mode).

The simplest form of driving a stepper motor is in the single-phase (wave) mode. In this mode the stator windings are activated one at a time in an orderly sequence. For a VR motor like that in Fig. 8.61 a single-phase operation can be obtained with an activation sequence A-B-C-D repeated over and over. This sequence would cause the rotor move in a clockwise direction with a resolution of 15° per step, for a total of 24 steps per revolution. If the number of stator phases p_s and rotor teeth t_r are known, the step size θ_s can be obtained as

$$\theta_s = \frac{360^\circ}{t_r \times p_s}. \tag{8.26}$$

Figure 8.62 shows the positions assumed by the rotor with a single phase activation sequence. The dot in the rotor allows to track its position. Assuming the rotor was initially at position 0°, as indicated in Fig. 8.61, the sequence (A-B-C-D) would cause a total rotor displacement of four steps or 60°.

A two-phase sequence can be obtained by activating two consecutive windings simultaneously. One such a sequence would be AB-BC-CD-DA. This sequence would also cause the rotor to move in steps of 15° clockwise, however, the alignment would be 7.5° out of phase with respect to a single-pole activation sequence as the rotor teeth position that minimizes the air-gap flux falls in-between the two excited poles. This operation mode produces the highest torque in the motor and also consumes the most power. Figure 8.63 shows the rotor positions for consecutive DA and AB activations.

A half-step operation can be obtained by combining the single and dual phase mode sequences in a way that entries of each activation sequence are alternated. This would produce a sequence DA-A-AB-B-BC-C-CD-D. Under this sequence the

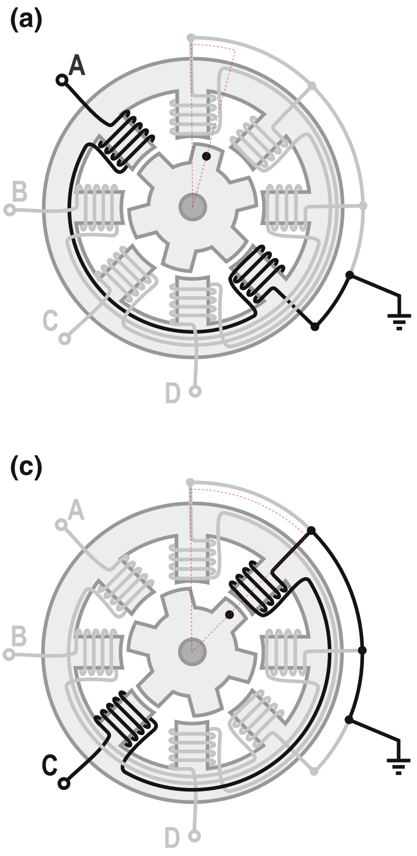


Fig. 8.62 Full-step wave activation sequence A-B-C-D in a 4-phase VR stepper motor. **a** Phase A excited: angle 15°, **b** Phase B excited: angle 30°, **c** Phase C excited: angle 45°, **d** Phase D excited: angle 60°

rotor teeth would minimize the flux path when aligned to stator poles with single phase excited and in-between the next two excited phases, resulting in an angular travel only half the step size. For the VR motor used in the illustration, this sequence would yield a resolution of 7.5°, duplicating the number of steps per revolution. The increased resolution causes a smoother motor operation but comes at the expense of a reduction in the motor torque.

From an MCU standpoint, either of these modes can be software controlled. An interface would require four I/O pins, each with unipolar drivers. Figure 8.64 shows a block diagram of an interface. In this case the buffers can be just a power PMOS providing both current buffering and level-shifting. The clamping diodes in the diagram protect the drivers against the motor inductive transient.

The functional mode, full- or half-step would depend on the programmed sequence. The speed of rotation can be controlled through the frequency with which the

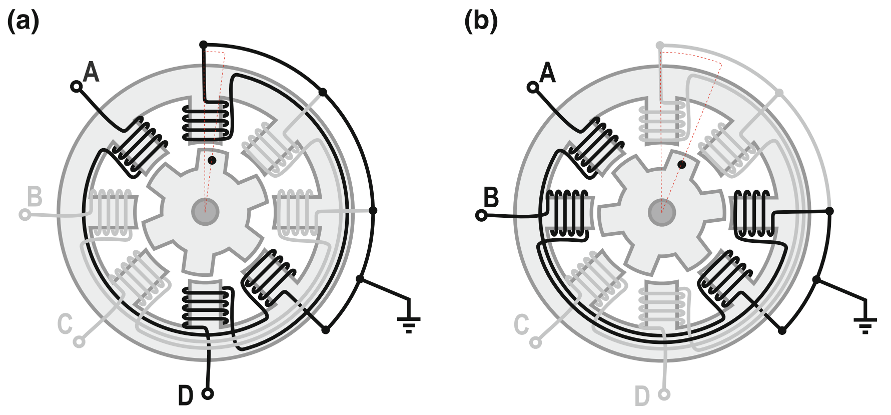


Fig. 8.63 Two-phase activation sequence DA-AB in a 4-phase VR stepper motor. **a** Phases D and A excited: angle 7.5°, **b** Phases A and B excited: angle 22.5°

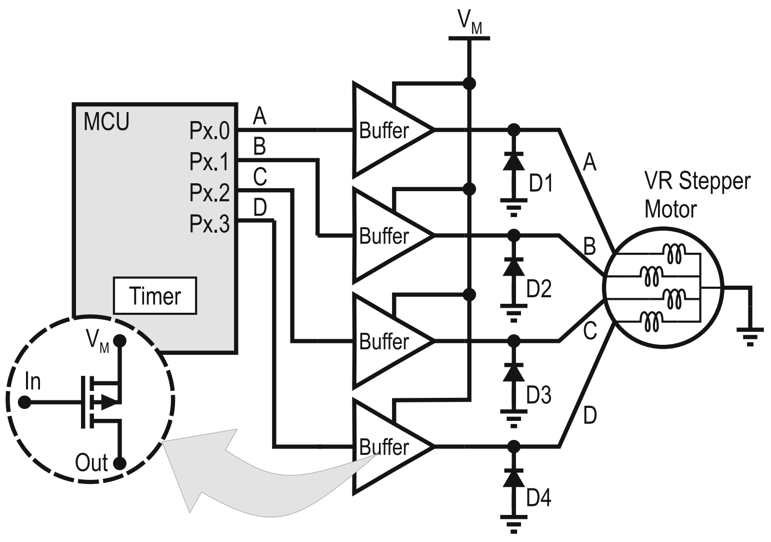


Fig. 8.64 Interface block diagram for a 4-phase VR stepper motor

winding activation sequence is fed to the stator. Continuous rotation is possible by sending out the activation sequence in a loop. Tables 8.3 and 8.4 show the command sequences to be sent through the designated I/O pins to advance a four-phase VR stepper like that shown in Fig. 8.61 when using single-phase full-step and half-step modes, respectively. Each row in the tables correspond to the code sent in each time slot of operation. Note that the last column in both tables refer to the angular position of the shaft. Every command will rotate the shaft by the corresponding step amount (15° or 7.5°) with respect to the previous position. Sending out the sequence in the reverse order would cause the motor to rotate in counterclockwise direction.

Table 8.3 Command sequence for full-step, single-phase operation of 4-phase VR stepper motor

Px.0 (A)	Px.1 (B)	Px.2 (C)	Px.3 (D)	Position
1	0	0	0	15°
0	1	0	0	30°
0	0	1	0	45°
0	0	0	1	60°

Table 8.4 Half-step control sequence for a 4-phase VR stepper motor

Px.0	Px.1	Px.2	Px.3	Phase	Position
1	0	0	1	DA	7.5°
1	0	0	0	A	15.0°
1	1	0	0	AB	22.5°
0	1	0	0	B	30.0°
0	1	1	0	BC	37.5°
0	0	1	0	C	45.0°
0	0	1	1	CD	52.5°
0	0	0	1	D	60.0°

8.10.5 Permanent Magnet Stepper Motors

Permanent magnet (PM) stepper motors are constructed differently than VR motors. Unlike VR's, the rotor of a PM motor is built using permanent magnets and has no teeth. PM motor stators have multiple windings, which can be configured to operate as either unipolar, bipolar, or bifilial windings. Torque in a PM motor occurs as a result of the excited stator poles attracting opposite poles and repulsing similar poles among the magnets in the rotor.

Unipolar Windings

Unipolar PM steppers typically have two stator windings, each split between two opposite poles, resulting in a total of four stator poles. The center taps of both winding is made externally available. Center taps can be independently brought outside the stator carcass providing a total of six terminals: four for the stator poles and two for the center taps. Some motors might have the center taps internally interconnected and brought out as a single terminal, providing five external wires.

Figure 8.65a shows a sectional view of a permanent magnet stepper motor with two unipolar stator winding and six magnetic poles in the rotor. Sub-figure (b) shows the arrangement of its windings. The winding terminals are labeled $A - \overline{A}$ for the first winding, with center tap $A1$ and $B - \overline{B} - B1$ for the second winding.

Operation of a PM stepper can be achieved by applying a fixed DC source V_M to the stator center taps and selectively grounding the winding ends. This way, the current

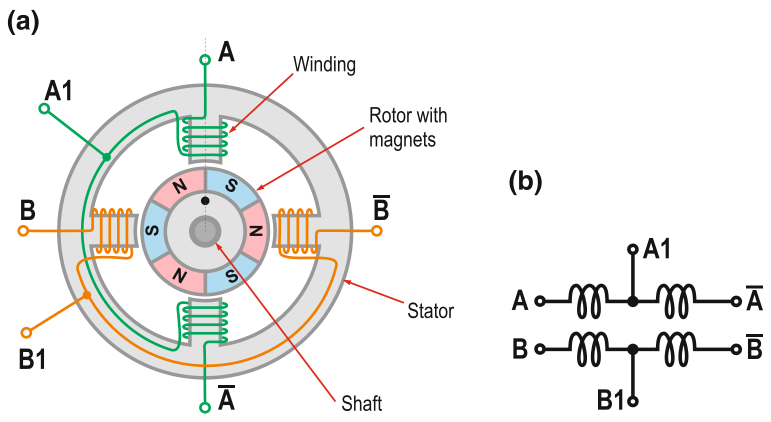


Fig. 8.65 Sectional view of a permanent magnet stepper motor and windings connections. **a** Motor cross section, **b** Winding arrangement

circulating through each winding determines the magnetization in the corresponding stator pole and thus determining which rotor poles will be attracted or repelled. Note that under this scheme only one half of the winding is used at a time, which limits the maximum torque the motor can generate. Operation from a microcontroller can be accomplished in a straightforward manner by using controlled switches which when closed, will ground the stator poles selected for being excited.

For the motor illustrated in Fig. 8.65, by having terminals $A1$ and $B1$ permanently connected to a power source (V_M), and grounding the winding terminals in a sequence $A - \overline{B} - \overline{A} - B$ repeatedly will cause the rotor to move in clockwise direction, incrementing its angular position by 30° in each step. Note that in this case, the number of steps per revolution is equal to the number of poles in the rotor. Figure 8.66 shows the rotor positions for each step in a sequence $A - \overline{B} - \overline{A} - B$, assuming and initial position of 0° as illustrated in Fig. 8.65. The rotor dot in the illustration allows tracking the shaft position after each step.

Permanent magnet motors can also be operated in one- and two-phase modes. The sequence illustrated in Fig. 8.66 corresponds to a single-phase or wave mode operation. Activating phases in pairs, in this particular case $BA - A\overline{B} - \overline{BA} - \overline{A}B$, moves the motor in single-step two-phase mode, which produces a higher torque at the expense of consuming twice the power as in the single-phase case. This mode also produces a half-step phase shift in the shaft angular position with respect to the positions reached in single phase mode. Combining, in an interleaved way, the single- and double-phase sequences has the same effect as in a VR motor: doubling the resolution, making the motor operate in half-step mode. Figure 8.67 illustrates the rotor dynamics in this mode for a partial sequence of three steps. In this case the resulting step resolution would be 15°, raising the number of steps per revolution from 12 to 24.

Interfacing a unipolar PM stepper motor to an MCU is quite similar to the arrangement for a VR motor. It requires only four unipolar driving elements that could be

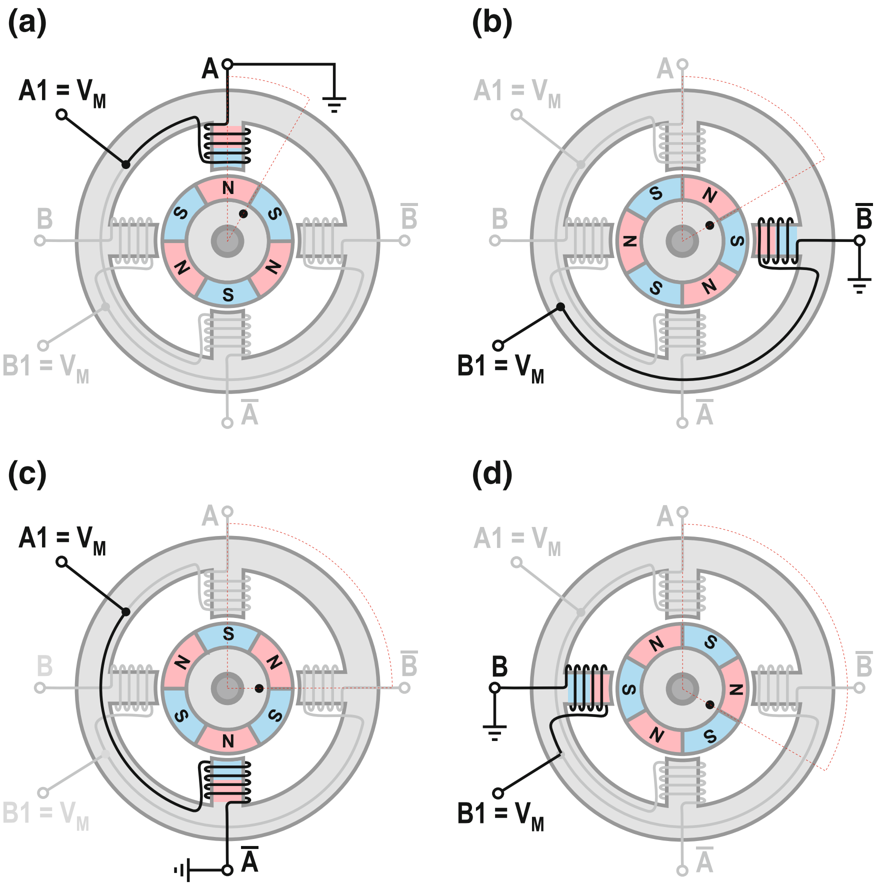


Fig. 8.66 Full-step wave activation sequence $A - \overline{B} - \overline{A} - B$ in a PM stepper motor. **a** Phase A excited: angle 30°, **b** Phase $\overline{B}$ excited: angle 60°, **c** Phase $\overline{A}$ excited: angle 90°, **d** Phase B excited: angle 120°

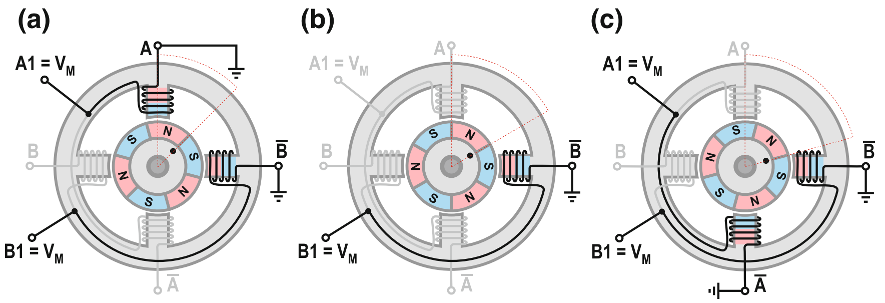


Fig. 8.67 Half-step sequence $A\overline{B} - \overline{B} - \overline{BA}$ in a PM stepper motor with six rotor poles. **a** Phase $A\overline{B}$ excited: angle 45°, **b** Phase $\overline{B}$ excited: angle 60°, **c** Phase $\overline{BA}$ excited: angle 75°

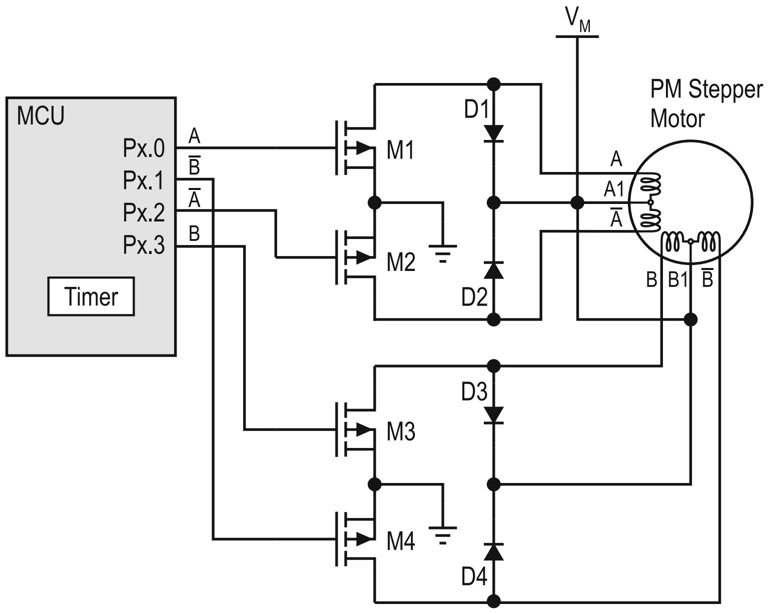


Fig. 8.68 Block diagram of a PM stepper motor to MCU interface

Table 8.5 Command sequence for full-step, single-phase operation of 4-phase PM stepper motor

A	$\overline{B}$	$\overline{A}$	B	Position
1	0	0	0	30°
0	1	0	0	60°
0	0	1	0	90°
0	0	0	1	120°

accommodated with power MOSFETs. In this case power NMOS devices would be recommended since we want to switch the connection to ground in the controlled windings. Figure 8.68 shows a block diagram of the connection. Note that NMOS transistors $M1$ through $M4$ have their source terminals connected to ground, performing the desired switching action. Diodes $D1$ through $D4$ provide clamping protection for the transistors against the inductive transient of the motor winding.

Table 8.5 shows the code activation sequence for controlling the stepper motor exemplified in this discussion in single-phase, single-step mode from an MCU interface like that of Fig. 8.68. This table assumes the shaft is at position 0°, as illustrated in Fig. 8.65 prior to the first command. Note that the binary sequence is the same as in the case of a VR stepper with unipolar winding, so figuring out the code activation sequence for half-step operation can be readily determined from Table 8.4.

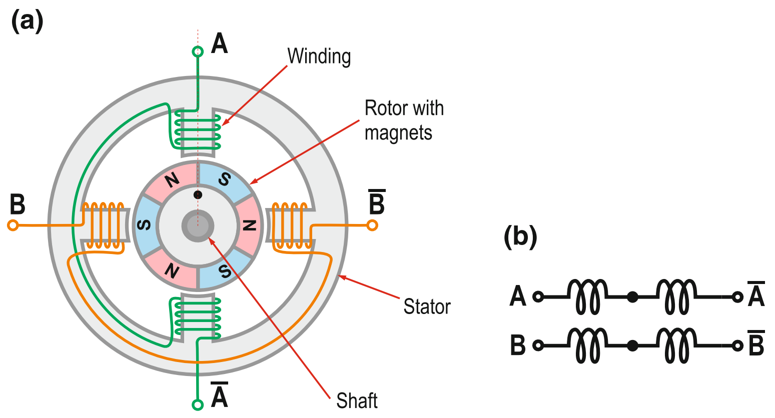


Fig. 8.69 Sectional view of a bipolar permanent magnet stepper motor and windings connections. **a** Motor cross section, **b** Bipolar winding arrangement

Bipolar Windings

A permanent magnet stepper motor with bipolar windings provides no external connections to the center taps, having only four external wire connections. In this configuration the activation of a stator pole uses the entire winding, generating the maximum torque the winding can provide. Control of stator magnetic poles is achieved by changing the direction of the current circulation in the winding. Figure 8.69 shows a sectional view of a PM stepper with bipolar windings, and the internal configuration of its coils.

The necessity of reversing the windings polarity calls for an external circuit more complex than that used in the case of unipolar windings. A polarity reversing driver can be satisfied by using an H-bridge per winding. Figure 8.70 shows a connection of the windings of a bipolar PM stepper motor using two H-bridges, one per winding.

Transistors $M1$ through $M4$ and $M5$ through $M8$ make up the two bridges, the first controlling winding A and the second for winding B. $D1$ through $D8$ are protective clamping diodes. Optional gate resistors (not shown in the figure) might be added to accelerate the MOSFET turn-off (see Fig. 8.56 for reference).

The software for operating a bipolar PM motor is not radically different from that used in the previous motor types analyzed above. Single- and double-phase full step operation is possible by activating one or both windings at a time. The interface circuit will apply V_M to A and GND to $\overline{A}$ when Px.0 is low and Px.1 is high, or viceversa when reversed. A similar situation occurs with B and $\overline{B}$ by controlling Px.2 and Px.3. Setting both Px.0 and Px.1 to high will just deactivate winding A. Similarly will happen to winding B by setting Px.2 and Px.3 to one simultaneously. Therefore, single-phase operation can be achieved by sending-out pins Px.0 through Px.3 the bit patterns in Table 8.6.

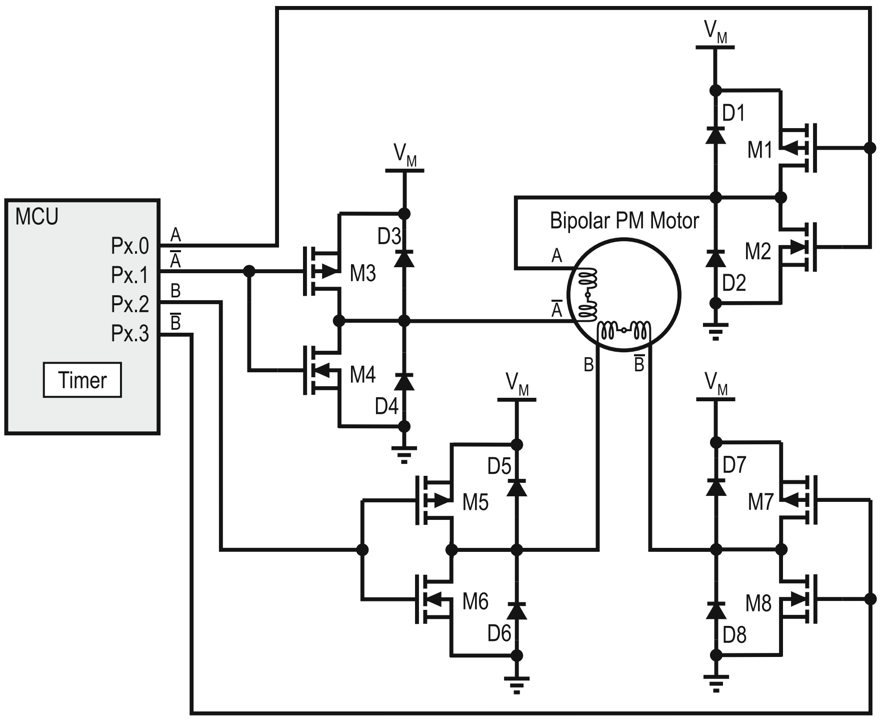


Fig. 8.70 Block diagram of a bipolar PM stepper motor to MCU interface

Table 8.6 Control sequence for single-phase, wave operation of a bipolar PM stepper motor

Px.0	Px.1	Px.2	Px.3	Phase	Position
0	1	1	1	A	30°
1	1	0	1	B	60°
1	0	1	1	$\overline{A}$	90°
1	1	1	0	$\overline{B}$	120°

Controlling the motor with a double phase activation in a sequence $\overline{B}A - AB - B\overline{A} - \overline{AB}$ produces a higher torque than a single phase excitation, while consuming twice the power. It causes a rotation in full step increments, shifted half a step forward with respect to a single-phase sequence. This shift allows for combining the single- and double-phase sequences to operate the motor in half-step resolution, as has been the case for the previous motor types. Obtaining the command sequence for half-step operation can be readily accomplished from the pattern in Table 8.6 once it is realized that the phase activation sequence would be $\overline{B}A - A - AB - B - B\overline{A} - \overline{A} - \overline{AB} - \overline{B}$. The obtention of the actual command table is left as an excursive to the reader.

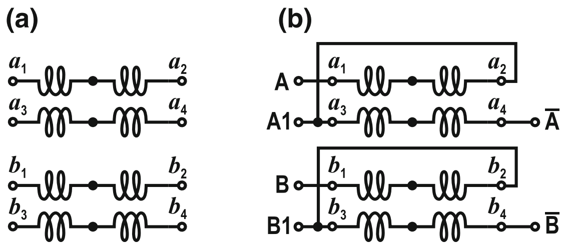


Fig. 8.71 Winding arrangement in a bifiliar stepper motor, illustrating their series and parallel connections. **a** Internal winding arrangement, **b** Serial connection as unipolar, **c** Parallel connection as bipolar

Bifiliar Windings

Some stepper motors provide the flexibility of being configurable to operate as either unipolar or bipolar by including dual windings in each phase and pole, with their terminals externally available. These are called bifiliar winding stepper motors. A permanent magnet, bifiliar stepper motor provides eight external wires that allow for configuring its windings as desired. Figure 8.71 shows the internal winding arrangement for this type of motors, illustrating how to externally connect them to produce either unipolar or bipolar equivalent windings.

The windings of each pole can be connected in series as illustrated in Fig. 8.71a to produce an equivalent unipolar winding. Leaving the center tap unconnected would produce a series bipolar winding (not illustrated in the figure), which allows for high voltage motor operation. The alternative connection in Fig. 8.71b illustrates how to make a parallel connection to produce equivalent unipolar windings. This alternative allows operating the motor with a lower voltage than the series connection.

Once a series or parallel connection as either unipolar or bipolar mode is established, interfacing and controlling the motor can be made as was discussed in the previous sections for their corresponding equivalent mode.

8.10.6 Hybrid Stepper Motors

Hybrid stepper motors combine features from both variable reluctance and permanent magnet steppers to produce a machine with the ability to produce high torque at low and high speeds with very fine resolution.

A hybrid stepper motor has two multi-toothed, soft iron disks in its rotor with a permanent axial magnet between them. The soft iron disks become extensions of the magnet ends, assuming each the characteristics of their respective end of the magnet. The disks are accommodated to have their teeth interleaved, allowing to accommodate a large number of interleaved magnets. Figure 8.72a shows the arrangement of components in a 16-teeth hybrid rotor. Subfigure 8.72b shows a frontal view of the rotor allowing to see the interleaved magnetic poles.

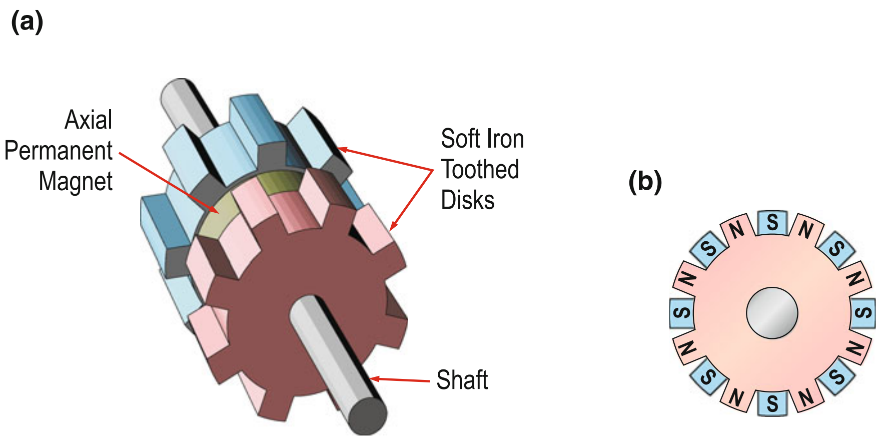


Fig. 8.72 Rotor arrangement in a hybrid stepper motor, illustrating components and frontal view. **a** Rotor components, **b** Rotor front view

The stator of a hybrid motor also has teeth in its poles to define preferred points of alignment with the rotor poles. The combination of toothed rotor and stator poles allows for accommodating a dense arrangements of alignment points in the structure, yielding motors with a large number of steps per revolution. Hybrid motors can produce up to 800 steps per revolution in full-step mode and twice that number when driven in half-step mode.

Hybrid stepper motors might have either unipolar, bipolar or bifiliar windings, therefore, operating them from an MCU will require the same types of interfaces and control algorithms that were described earlier for these types of stator windings. Figure 8.73 shows a sectional view of a hybrid stepper motor with a 16 teeth rotor and a two-phase bipolar stator, producing a resolution of 11.25° per step or 32 steps per revolution. Driving this motor in half-step mode would result in 64 steps per revolution.

8.10.7 Microstepping Control

Besides operating in either half- or full-step modes, modulating the pulses applied to the windings of a stepper motor can further reduce the step size. This driving modality is called *microstepping*. A microstepping driver, instead of just exciting and de-exciting in a binary way the windings of a stepper motor, gradually transitions power from a winding to another. This creates a smoother motor operation, and depending on how many levels are designated for the transition is the number of microsteps resulting from a single step. To better understand the concept, let's consider a two-phase, bipolar stepper motor with windings A and B. In a full-step single phase mode, switching from phase A to B simply turns off phase A and turns on phase B,

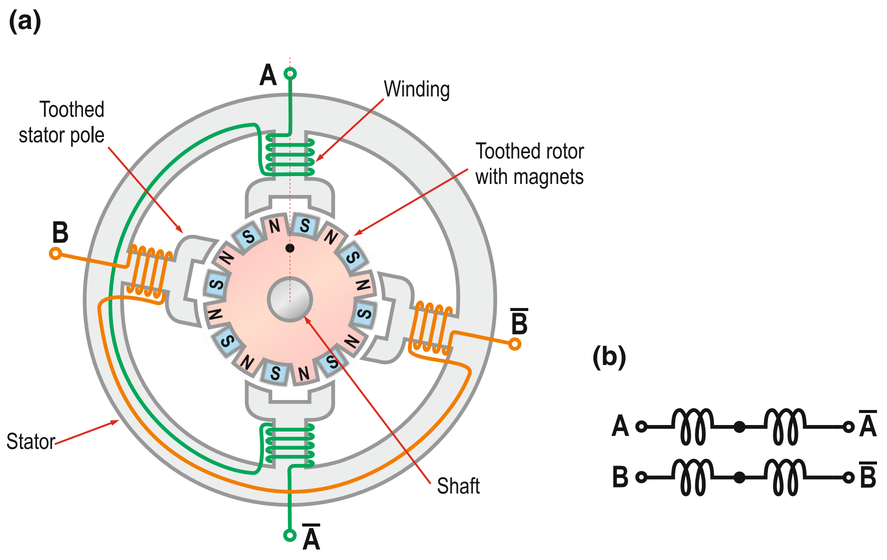


Fig. 8.73 Sectional view of a hybrid stepper motor with bipolar windings. **a** Motor cross section, **b** Bipolar winding arrangement

alternating polarities. The resulting voltage waveform in windings A and B would be like the continuous staircase waveforms shown in Fig. 8.74.

The ideal waveform for a smooth behaviour would be a sine/cosine waveform like the dotted line in Fig. 8.74. In a practical implementation, a discrete approximation of the sinewave like that in the dot-dash waveform in Fig. 8.74 would be used. Each discrete level would produce a stable position for the rotor, resulting in a microstep. For the case illustrated, the motor would operate with four microsteps. Assuming a sixteen-pole hybrid motor like that illustrated in Fig. 8.73, this technique would increase the step resolution from originally 32 steps per revolution (11.25° per step) in full-step mode to 128 steps per revolution or 2.8° per step. This scheme requires only two bits to encode the four discrete levels used in the sequence. Commercial microstepping drivers are capable of producing up to 32 discrete levels per step (6-bit), significantly increasing the step resolution. The increased resolution comes at the expense of reduced torque. Note that most of the time the windings will be excited below their maximum value. Also, a microstepping driver is more complex than a conventional full/half stepper driver.

8.10.8 *Software Considerations in Stepper Motor Interfaces*

The interface examples shown in the previous sections illustrated ways of controlling a stepper motor directly with I/O ports. In direct I/O control, a software routine is

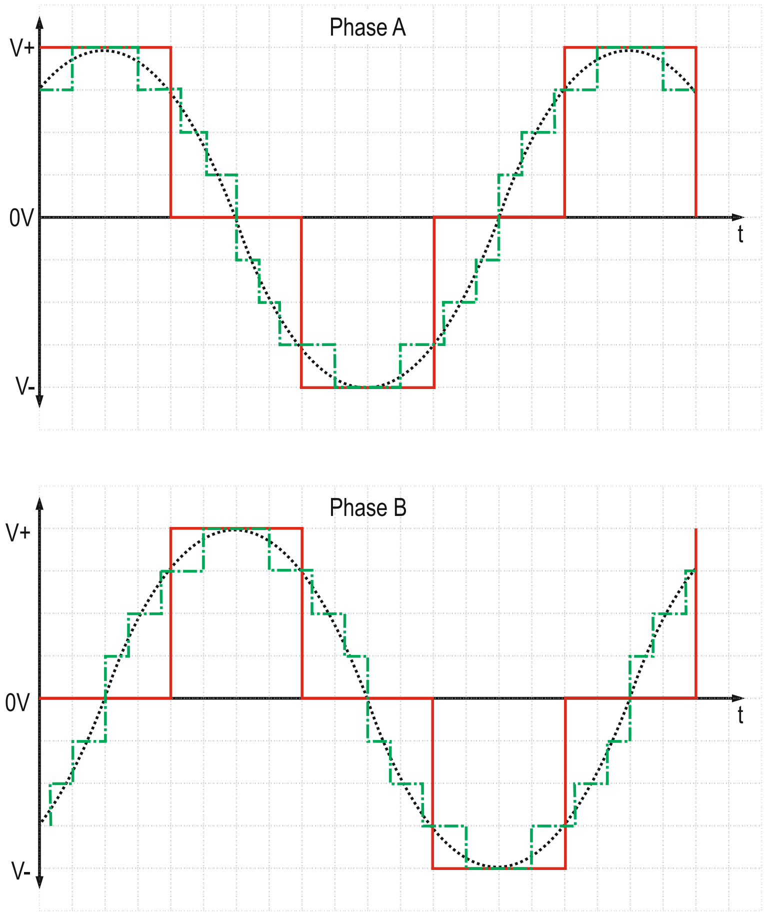


Fig. 8.74 Waveforms in a microstepping driving approach

required to keep track of the control sequence to make possible the correct shaft rotation. The software requirements for the processing routine are relatively simple:

- A look-up-table (LUT) holding the command sequence. If a PM bipolar motor were used, the LUT would have only four entries: 07h, 0Dh, 0Bh, and 0Eh, assuming byte entries with the least significant four bits actually holding the control code. This would allow easy alignment with the hardware connection illustrated in Fig. 8.70.
- A circular pointer to orderly access the sequence. By circular we mean rolling over the pointer from bottom to top of the LUT or viceversa when the LUT boundaries are reached.
- A timer function to establish the rotation speed. The usage of a timer channel allows for precise periodic updates of the codes in the motor control I/O pins.

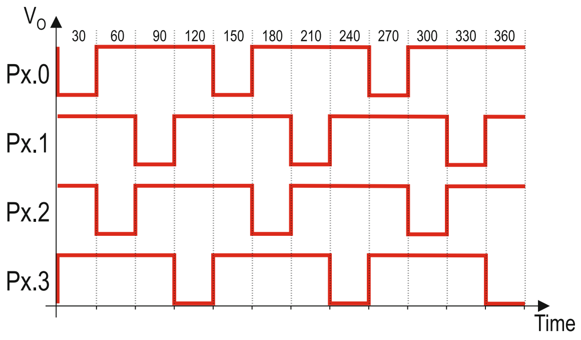


Fig. 8.75 Timing control waveforms for a PM bipolar stepper motor

With the pointer initialized to the top of the LUT, every time the timer function is triggered, its ISR updates the motor control I/O pins with the next code entry in the table. Updating the pointer in a circular way allows for continuous control code output. If the pointer update is done by incrementing its value, rotation would be clockwise, while by decrementing would produce counterclockwise rotation. Considering for example the case of a permanent-magnet stepper with two bipolar windings, the resulting waveforms in the I/O pins caused by the described software routine would be similar to that illustrated in Fig. 8.75. Note that this timing diagram is just the same sequence described in Table 8.6.

In applications where devoting CPU cycles to keep track of a stepper motor sequence might become difficult, a dedicated stepper motor controller can become an option. A dedicated stepper motor controller has internal logic to keep track of the motor stepping sequence, providing for forward and reverse operation. Numerous commercial off-the-shelf options from different vendors can be identified. A couple of representative examples are Motorola's MC3479 and Texas Instrument's DRV8811.

Motorola's MC3479 is an example of a simple stepper motor driver, capable of operating a two-phase, bipolar stepper motor in either full-step or half step modes in forward or reverse direction. The chip is capable of directly driving coils with voltages in a range from 7.2 V to 16.5 VA with up to 350 mA per coil. Its digital interface allows specifying the step mode ($\overline{F}/HS$), rotation direction ($\overline{CW}/CCW$), and step clock (CLK), among other signals. The analog side allows feeding the motor voltage V_M and connection to the motor windings $L1-L2$ and $L3-L4$ for the respective winding phases. Figure 8.76 shows an internal block diagram and typical application of this chip. For more details on this chip, the reader is referred to its data sheets [66].

TI's DRV 8811 provides a more complete solution for stepper motor interface. It supports two-phase bipolar stepper motors up to 2.5 A per winding. Besides the basic indexing functionality and FWD/REV and full/half stepping capabilities, the 8811 supports microstepping motor control with up to eight different current levels. The chip also features thermal shut down, short-circuit, and undervoltage lockout protections. The external interface requires only a few discrete resistors. The chip package features an exposed pad to remove heat through the PCB that allows operation without

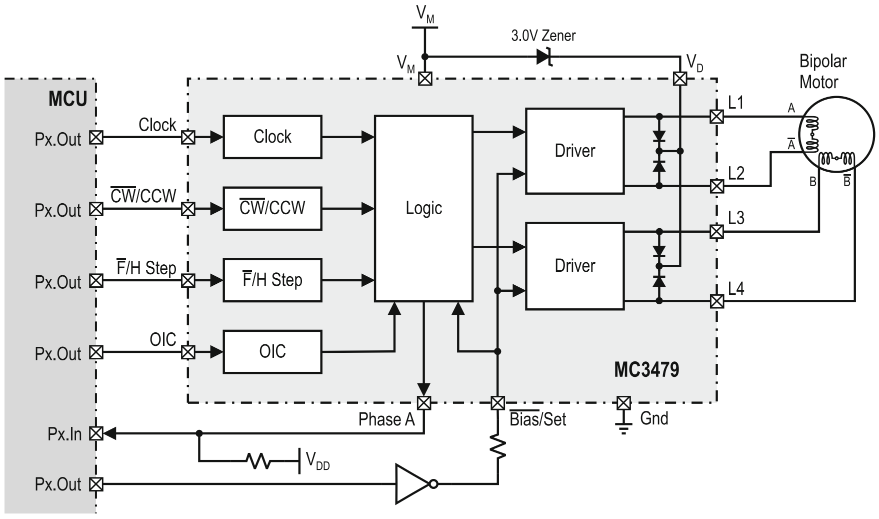


Fig. 8.76 Interfacing a bipolar stepper motor using the MC3479

a heatsink in most application conditions. Figure 8.77 shows the internal structure and typical application diagram of a DRV8811. For more details on configuring and operating this chip, the reader is referred to its data sheets [67].

8.10.9 Choosing a Motor: DC Versus Servo Versus Stepper

When compared against other types of motors, DC motors are relatively inexpensive and simple. When powered, their open loop rotational speed can be reasonably controlled using PWM or a DAC interface if the motor operates under constant load. Under general loading conditions however, the lack of a feedback control mechanism will cause cumulative deviations from the desired set point, being this a reason why they cannot be used without feedback in applications requiring precise control. By including some form of feedback, like a tachometer or rotary encoder, optional gears, and adding a control circuit their application range can be expanded to a wide number of areas. However this solution involves additional hardware and software components.

DC motors are the preferred solution when the controlled process can be dealt with the motor speed. A number of applications allow open-loop operation, like in the case of variable speed fans or electric toys. One weakness of DC motors is the need of brushes and commutators to pass energy to the rotor winding. This components tend to reduce the motor life due to mechanical wear. Although brushless alternatives for DC motors are available, these need additional electronics to synchronize the stator magnetic field with the rotor permanent magnet poles to produce spin, making them more reliable, but also more expensive.

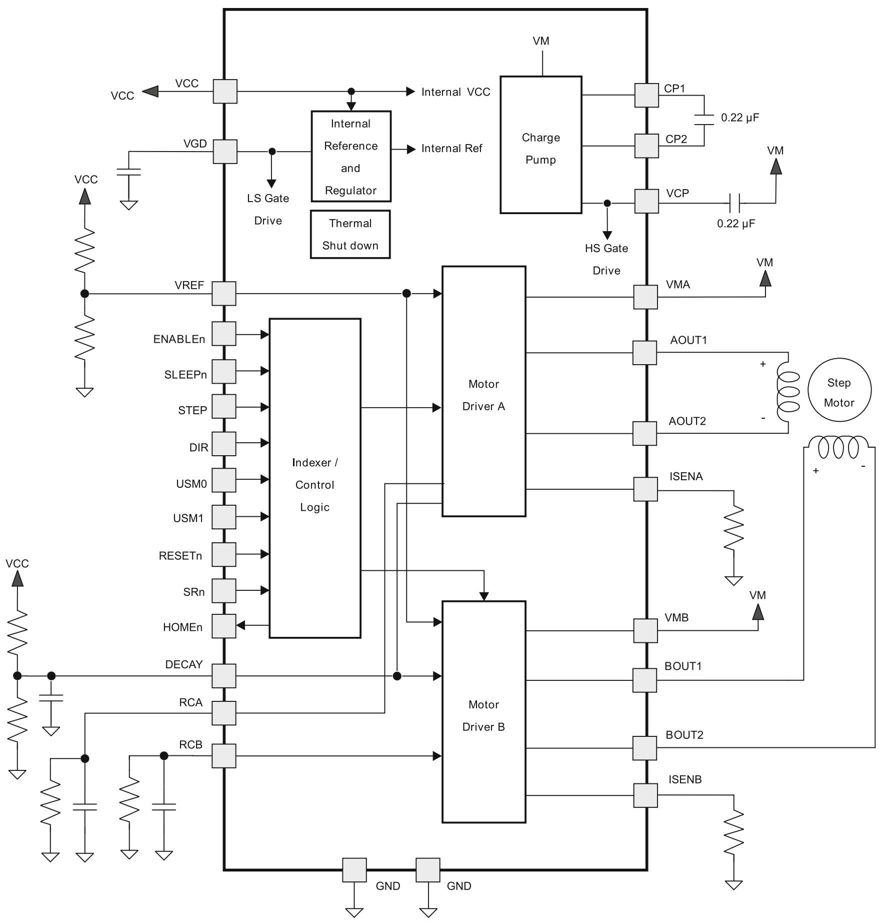


Fig. 8.77 Interfacing a bipolar stepper motor using TI's DRV8811 (*Courtesy of Texas Instruments, Inc.*)

Servo motors, although based on DC motors, provide an elegant solution for positional control applications as they include an internal feedback loop, gears, and a control circuit that allow for precise angular position control. Servo motors are more expensive than either DC or stepper motors because of the inclusion of gears and the control electronics necessary for their operation. Servos are available with brushed and brushless DC motors as well as with AC motors, offering a wide range of sizes and power. Brushed servo motors suffer from the same weakness as brushed DC motors in the limited life of brushes and commutators. In addition, those using potentiometer-based feedback sensors, the potentiometer usually becomes a cause of failure. Nevertheless, servos are very efficient, lighter than steppers, produce better torque at high speeds, and provide a better reserve of power for momentary torque surges.

Stepper motors offer the most precise open-loop position control because of their inherent stepping nature. For equal power rating, steppers are less expensive than servo motors and allow direct shaft loading. Also, stepper motors offer a higher reliability than any brushed DC motor option as no brushes or commutators are needed for their operation, and in comparison to servos, steppers have less parts prone to failure. Errors due to friction or other factors do not accumulate in stepper motors, and if operated with a load less than or equal to their nominal torque, they rarely miss a step. Steppers however, tend to be noisier and produce more heat than DC or servos counterparts, and are also less tolerant to overload.

8.11 Summary

This chapter provided concepts and applications of developing interfaces for microcontroller-based systems using their general purpose input-output capabilities. The Chapter introduced concepts of GPIO structure and elaborated to the point that allowed for integrating user interfaces, large loads, and motors.

General purpose I/Os form the most important means for microcontrollers to interact with the external world. Using their Data-In/Data-Out registers, configuring their direction registers, and enabling their interrupt capabilities were the main subject of Sect. 8.1. Next the fundamental considerations to interface switches and switch arrays, LEDs, LED arrays, numeric alphanumeric, and graphic LCDs, were discussed in detail in Sects. 8.3 and 8.4.

The subject of handling large loads, like heaters, solenoids, motors, lightning fixtures, etc. were discussed in Sects. 8.2 and 8.10. Every section featured how MSP430 devices support each of the base functions discussed, along with hardware and software examples illustrating their applicability.

8.12 Problems

8.1 An embedded system incorporates an MSP430F2274 clocked by a 4.096 MHz at 3.3 V, driving two seven-segment displays and two discrete LEDs. The seven-segment display draws 7.5 mA per segment at 1.8 V and the discrete LEDs operate with 10 mA each at 1.6 V. Three push-buttons are needed for completing the user interface. Design a suitable interface to connect all LEDs, seven-segments, and push-buttons. Estimate the circuit power requirements and recommend non-regulated power supply and a suitable regulator.

8.2 Assume the system described in Problem 1 is to be fed from a 4.0 V, 2400 mAH lithium battery. Estimate the expected battery life, assuming LEDs are driven dynamically at 30 % duty cycle. What would be the regulator efficiency? Determine the MCU thermal dissipation for the given loading conditions and verify if it is operating in a safe temperature range.

8.3 Provide an initialization sequence for the MCU in problem 1, and write a program that will make the system operate as a down counting stopwatch. Label the three keys as “b1”, “b2”, and “b3” and write a short program to display the key name on the seven-segment display while the corresponding key is depressed.

8.4 Devise an algorithm for joining a keypad scan function and a software debouncing function for the keypad such that a seven-segment display shows the character corresponding to the depressed key. Assume only single key depressing will be allowed.

8.5 Write a program to return on R5 the ASCII code corresponding to the depressed key in a 16-key keypad. Show how shall the keypad be interfaced to the MCU.

8.6 Provide a stand-alone program that will make a dedicated 12-key keypad scanner from an MSP430 Launchpad, placing in a predesignated memory location the ASCII code of the depressed key.

8.7 Design a dumb HEX terminal using 16-key keypad and a 4x20 alphanumeric LCD on an MSP430F169. Provide a keypad buffer in the MCU with a capacity of 20 characters with a last-in first-out replacement policy for the data sent to the LCD.

8.8 A 120VAC fan is to be controlled by an MSP430F2273. Assuming the maximum fan current is 2A, provide a solid-state relay interface using a Sharp S108T02. Verify both voltage and current compatibility in the control interface. Add a single push-button interface to the system and a status LED to be lit when the fan is ON. Provide a short program to toggle the fan with the push-button.

8.9 A standard Futaba S3003 servo-motor operated at 5.0 V moves between its two extreme positions 180° apart when fed a 50 Hz signal with 5–10 % duty cycle. Nominally the servo moves to its center position with a 7.5 % duty-cycle signal. Assuming the digital signal requires less than 1mA when pulsed, and the motor consumes anywhere between 8 mA and 130 mA in a load range from idle to full torque. Provide a suitable interface to control the servo from an MSP430 Launchpad I/O pin and provide a software-delay based function that would allow controlling the shaft with a resolution of 0.5°.

8.10 A 12VDC, 1.5A per phase, two-phase PM bipolar stepper motor with eight rotor poles is to be controlled from an MSP430F2274. Provide a discrete component H-bridge interface to control the motor with a GPIO port in the MSP and indicate the actual components to be used. Provide a safe to operate interface and a function accepting as parameters(via registers) the direction in which the stepper will move and the number of steps in each move. Assume the motor is to be operated in half-step mode. Determine the number of steps per revolution to be obtained in this interface.

8.11 Provide an enhanced interface for the motor in problem 10 by using a DRV8811. Upgrade the driving software function to operate the motor in microstepping mode with eight microsteps per full step of the motor. Calculate the new step resolution of the motor.