Chapter 6
Fundamentals of Interfacing

Before a microcontroller can be put to work, the fundamental components that enable its functionality must be in place. These components are what we call the basic MCU interface. The first part of this chapter deals with the discussion of how to provide the fundamental requirements of power, clocking, reset, and bootstrapping needed to enable the functionality of any MCU. MSP430 devices have different levels of support for their basic interface. The supports level changes depend on the specific generation and family. In this chapter we discuss how these requirements are satisfied across all device generations.

6.1 Elements in a Basic MCU Interface

Any microprocessor, needs a basic interface to become usable. This interface includes four fundamental elements:

- A power source to feed power to the CPU and system peripherals
- A clock generator to synchronize the system operation
- A power-on reset circuit (POR), to take the system to its initial state
- A booting function, to initiate the system software execution

Microcontrollers frequently have several of the required elements embedded within their chip, while stand-alone CPUs require external components to provide all the elements for their basic interface. These components are pictorially illustrated in Fig. 6.1. The next sections provide insight into of each of these components.

M. Jiménez et al., *Introduction to Embedded Systems*,
DOI: 10.1007/978-1-4614-3143-5_6,

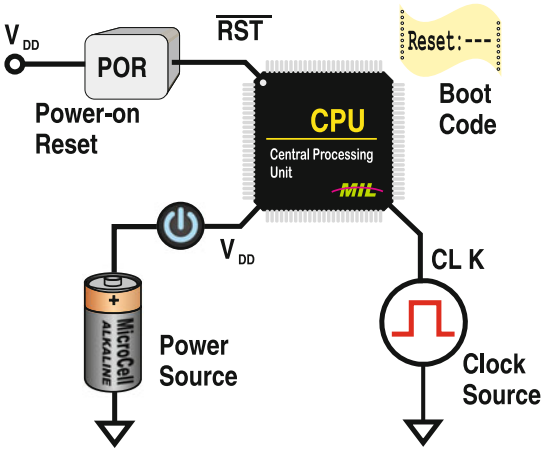

Fig. 6.1 Components in a basic CPU interface: power source, clock generator, power-on-reset, and boot sequence

6.2 Processors' Power Sources

Regardless of the processor type, power sources need to be externally provided. They can be implemented from different sources: batteries, wall connected AC power outlet, or some forme of energy collecting or scavenging device. In every case, selection and design criteria must be applied to properly feed power to the CPU and its peripherals.

Power requirements of a microcontroller-based application are not too different from those of any other type digital system. Basic requirements call for a steady supply voltage and enough output current to feed, under worst case conditions, the nominal system load. Such a load typically includes the CPU and its surrounding peripherals. The electrical characteristics provided in the data sheet of a device (μP, MCU, or peripheral) clearly indicate the allowed values for supply voltage and regulation. It is of outmost importance to differentiate between *Absolute Maximum Ratings* and *Recommended Operating Conditions* when searching for the appropriate supply voltage level for a device.

The Absolute Maximum Ratings of a device specify levels of stresses that if exceeded or used as operating conditions will affect reliability or cause permanent damage to the device. Therefore, we should never design for working at these levels. The safe operating levels are those specified in the *Recommended Operating Conditions* so these should be the choice for a good, reliable design, instead.

Table 6.1 shows the absolute maximum ratings of an MSP430G2231 microcontroller. Here we can observe the device's absolute limits for the maximum and minimum supply voltages along with other limiting parameters. For this device, setting $V_{CC} = 4.1\,\text{V}$ would be a poor design choice, because although this value does not exceed the absolute maximum, it would stress the device to the point where it would easily fail due to overvoltage.

Table 6.1 Absolute maximum ratings for MSP430G2231

Parameter	Condition	Limits
Voltage applied at V_{CC} to V_{SS}		$-0.3\,\text{V}$ to $4.1\,\text{V}$
Voltage applied to any pin		$0.3\,\text{V}$ to $V_{CC} + 0.3\,\text{V}$
Diode current at any device pin		$\pm 2\,\text{mA}$
Storage temperature range, T_{stg}	Unprogrammed device	$-55\,^{\circ}\text{C}$ to $150\,^{\circ}\text{C}$
	Programmed device	-40 to $85\,^{\circ}\text{C}$

Table 6.2 Recommended operating conditions for MSP430G2231

Symb.	Parameter	Condition	Min.	Max.	Unit
V_{CC}	Supply voltage	Program execution	1.8	3.6	V
		Flash programming	2.2	3.6	
V_{SS}	Supply voltage		0	0	V
T_A	Operating temp.	Free air	-40	85	°C
f_{clk}	Clock frequency	$V_{CC} = 1.8\,\text{V}$, Duty cycle $= 50\,\% \pm 10\,\%$	dc	4.15	MHz
		$V_{CC} = 2.7\,\text{V}$, Duty cycle $= 50\,\% \pm 10\,\%$	dc	12	
		$V_{CC} = 3.3\,\text{V}$, Duty cycle $= 50\,\% \pm 10\,\%$	dc	16	

Table 6.2 shows the recommended supply levels for the same device. Note that the maximum recommended supply voltage value keeps a safe margin of 0.5 V below the absolute maximum. The ground level voltage specification can be also observed to remain at zero, independently from the value chosen for V_{CC}.

In the case of the MSP430G2231, there is no specification for a regulation requirement. This is a characteristic commonly found in microcontrollers. MCUs frequently accommodate internal references and regulators for on-chip components susceptible to V_{CC} variations. For example, the MSP430G2231 embeds an internal 1.5 V voltage reference for its analog-to-digital converter and a programmable supply voltage supervisor that allows for selecting the minimum allowable operational voltage for the device, within the recommended range. This relaxes the regulation requirements of the unit. Nevertheless, it is always recommended to use a regulated power supply for whatever voltage level is selected for operating the chip. Processing speed, signal compatibility, power consumption, and system predictability are all dependent on the steadiness of the supply voltage. Moreover, older MCUs and microprocessors have less elaborated on-chip power supply modules. In such cases, the provision of a regulated supply becomes a must.

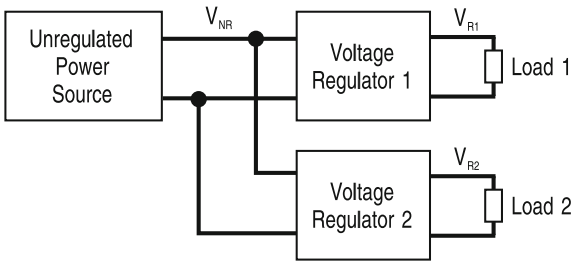

Fig. 6.2 Structure of a typical MCU power supply

6.2.1 Power Supply Structure

Bare power sources, regardless of their type, are far from ideal and thus rarely meet the voltage steadiness requirement for digital circuits or microprocessor-based systems. For example, the output voltage of an AC to DC converter tends to change with load fluctuations or input voltage changes. Its outputs also carry ripple voltage fluctuations that worsen as the load current increases. Batteries tend to decrease their voltage level as they give off their stored charge. Less conventional sources such as solar cells or other forms of energy scavenging devices are even less stable in terms of output voltage levels. For these reasons, bare power sources are frequently referred to as *unregulated power sources*. Making them usable in digital systems require some form of power conditioning between them and their load.

A typical MCU power supply includes an unregulated power source and a voltage regulator to feed the load. If more than one voltage level were required, multiple regulators would be used, one for each different voltage level. This situation is illustrated in Fig. 6.2 for a system requiring two different voltage levels V_{R1} and V_{R2}.

The unregulated source must be able of supplying the total power required by all loads plus the consumption of the regulators themselves. The requirements of each load must be met by the individual voltage regulators. To design the system power supply we need to determine the requirements of each load in terms of voltage level, peak current, and regulation.

The voltage level and regulation requirements are given by the microcontroller or MPU specifications. Each independent peripheral in the system will also have its own set of power specifications. Most devices allow in their specifications for a range of supply voltage values, easing the power supply selection process. However, once a specific voltage level is selected, it must be kept fixed at the selected value for proper system operation. Take as an example the case of the MSP430F149, for which the recommended operating conditions a supply voltage level goes from 1.8 to 3.6 V. If we choose to operate the MCU at 3.3 V, such a voltage level must be supplied through a regulated source maintaining it at the chosen value.

Microprocessors usually have more specific and restrictive requirements. For example, an OMAP-L137 requires a DVDD supply voltage of 3.3 V with a regulation of ±4.5 %. Such a requirement calls for a dedicated voltage regulator capable of maintaining the supply voltage level from 3.15 V to 3.45 V for all loading conditions.

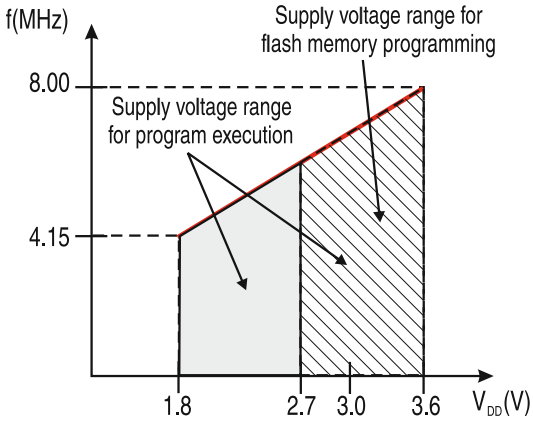

Fig. 6.3 Frequency versus supply voltage plot for an MSP430F149 MCU

6.2.2 Power-Speed Tradeoff

One important consideration when choosing the supply voltage of a digital system is its power-delay trade-off: the lower the voltage, the smaller the power consumption but also the slower its processing speed. Power in digital circuits, particularly in those based on CMOS technologies, varies with the square of the supply voltage. Reducing the supply voltage of a CMOS circuit by a factor of two produces a reduction in power consumption by a factor of four, which results very attractive from a power consumption standpoint.

However, the designer must also consider that a reduction in the power supply level carries a reduction in the maximum frequency at which the system can operate. In microprocessor or microcontroller based systems, the maximum frequency of operation is almost directly proportional to the supply voltage. This situation is illustrated in Fig. 6.3 for an MSP430F149. It can be observed that when V_{DD} is set to its minimum value ($V_{DD} = 1.8\,\text{V}$, the maximum clock frequency is limited to 4.15 MHz. To operate the chip at its specified maximum frequency of 8.0 MHz, the supply voltage needs to be raised to 3.6 V.

The best recommendation for choosing the supply level of a microprocessor-based application is using the lowest voltage level that allows for the proper system operation.

6.2.3 Power Supply Capacity

Further consideration regards the current and voltage output capability of the chosen power source.

With this in mind, let us first look at the regulator requirements. The sustained current capacity of a regulator must supply the maximum sustained current of the

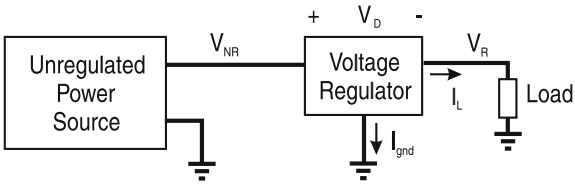

Fig. 6.4 Non-regulated source feeding a load through a 3-terminal linear voltage regulator

total load it feeds. Thus, given n loads $\{L_1, L2, \ldots, L_n\}$ to be fed by a voltage regulator, each with nominal current I_{Li}, the regulator feeding them must be able to provide a current $I_R \geq \sum_{i=1}^{n} I_{Li}$. As a practical measure, the regulator capacity is chosen with a 15–20 % of overhead capacity.

On the other hand, a voltage regulator needs to maintain a minimum voltage drop between its input and output terminals and also requires a small amount of current to operate. The minimum voltage drop between the input and output terminals is called the regulator's *dropout voltage* V_D. This requirement imposes a minimum voltage level for the unregulated power source, as indicated in Eq. (6.1).

Figure 6.4 shows a block diagram of the connection of a non-regulated source to a load via a linear regulator. Here we can also observe the regulator's ground current, I_{gnd}. This current is necessary to bias the regulator's internal pass element to maintain a steady output voltage. From this connection, we see that the minimum voltage level VNR for the unregulated power source must satisfy

$$V_{NR} \geq (V_R + V_D) \tag{6.1}$$

and the current capability for the supplied INR must satisfy

$$I_{NR} \geq (I_R + I_{gnd}) \tag{6.2}$$

Finally the total power supplied by the non regulated source is $P_{NR} = V_{NR} \cdot (I_R + I_{gnd})$ and the power delivered to the load $P_R = V_R \cdot I_R$. We can therefore estimate the regulator's efficiency as $Eff = P_R / P_{NR} \times 100\,\%$. This tells us that we should select V_{NR} as close as possible to its minimum value to reduce the power loss at the regulator.

6.2.3.1 Additional Considerations on Regulators

In general, when selecting a regulator, the designer can choose between linear regulators, either in their standard or low-dropout (LDO) versions, or switching regulators. Linear regulators use a pass device, typically one or more transistors, with a feedback network to maintain a steady output voltage. In a few words, the pass element is like an adjustable resistor that changes its value to keep a steady output voltage. In particular, standard linear regulators in their three-terminal packages are available in a wide range of input and output voltages and current ranges. They can provide

regulated currents, as large as 10 A, with small I_{gnd} requirements, typically less than 10 mA, and moderate dropout voltages (1.7 V $\leq V_D \leq$ 2.5 V). Among linear regulators, LDOs have the lowest V_D requirement, typically between 0.1 and 0.7 V at the expense of increased I_{gnd} (between 20 and 40 mA). On the downside, their maximum output current is in general limited, typically less than 1 A.

Another factor to consider when using linear regulators is their efficiency. The larger the difference $V_{NR} - V_R$, the lower the regulator efficiency. For example, neglecting I_{gnd} and choosing a 9 V source to feed a constant load at 3.3 V through a linear regulator would yield a maximum efficiency of 36.67 %. Using a 4.5 V source instead, the efficiency would be increased to 73.3 %. The minimum value of V_{NR} is actually limited by Eq. (6.1).

Switching regulators, also called DC-to-DC converts, operate by turning on and off the input voltage to the load, exerting control on the average voltage seen by the load through the duty cycle of the switching signal. Reactive elements like inductors and capacitors are used to smooth-out the output signal. This gives them great flexibility in producing output voltages that can be either higher or lower than the input voltage.

Switching regulators, are particularly attractive by their efficiency, frequently reaching values over 85 %. However, DC-to-DC converters tend to be a lot noisier than standard linear or LDO regulators. In fact, when the load includes noise sensitive components, it is common to place an LDO at the output of a DC-to-DC converter.

Example 6.1 evaluates two alternatives for feeding a hypothetical embedded design application.

Example 6.1 *Consider using a standard 9 V alkaline battery for feeding a 3.3 V, 120 mA load via a linear voltage regulator uA78M33C. Estimate the approximated battery life and usage efficiency assuming a constant load current.*

Solution: *This regulator is rated at 3.3 V/500 mA, has a maximum bias current* $I_{gnd} = 6$ mA *and a dropout voltage* $V_D = 2.0\,V$. *Using the Fig. 6.4 as reference, the total load current seen by the battery would be approximately* $I_{NR} = I_L + I_{gnd} =$ 126 *mA and the minimum required non-regulated input voltage to the regulator should be* $V_{NR} \geq 5.3\,V$.

A typical 6LR61 9 V disposable alkaline battery provides a charge of approximately 480 mAh at a discharge rate of 100 mA, and drops to about 380 mAh at 300 mA load [38]. Both these values correspond to a minimum battery voltage of 4.8 V. Interpolating these values for the particular load and minimum voltage of this problem, yields a usable charge of approximately 447 mAh. The expected battery duration would be about:

$$t_{bat} = \frac{Q_{bat}}{I_{NR}} = \frac{447\,mAh}{126\,mA} = 3.55\,h.$$

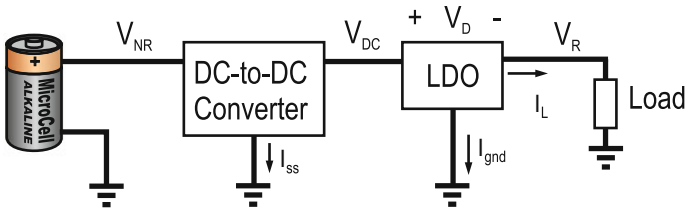

Fig. 6.5 Power supply integrating a DC-to-DC converter and LDO

In this case the efficiency at the regulator would be:

$$Eff = \frac{V_R \times I_R}{V_{NR} \times (I_R + I_{gnd})} \times 100\,\% = \frac{3.3\,V \times 120\,mA}{9\,V \times (120 + 6)mA} = 34.92\,\%$$

This means that over 65 % of the energy supplied by the battery would be lost as heat in the regulator.

A better design could use a DC-to-DC switching converter plus an LDO as illustrated in Fig. 6.5. A TPS54232 DC-to-DC converter can be configured to reduce the 9 V battery voltage to 3.45 V with 88 % efficiency. With an LDO like a TPS79933 operating with a nominal dropout of 150 mV and $I_{gnd} = 50\,\mu A$, its efficiency would be 96 %. The combined DC-to-DC–LDO efficiency would now be 85 %.

As the TPS54232 converter can accept input voltages down to 3.5 V, the usable battery charge is now estimated to 567 mAh. At 9 V the battery current will be approximately 52.3 mA, while at 3.5 V it will increase to about 134.5 mA. A rough battery life estimate based on a linear approximation with this current range yields 7.1 H of operation, nearly 7 h of continuous operation. This is almost twice the expected battery life. This estimate could be further improved if the load, assuming an embedded application, used some form of low-power mode.

6.2.4 Power Supply Distribution Tips

Many problems arising in microprocessor-based designs are caused by noise in the power supply lines. *Power supply noise* can be introduced by multiple factors, being among the most notorious the switching nature of digital circuits. Every change of state in a digital circuit induces a change in the power supply current. This creates a time varying current demand on the power supply line, which acting upon the inherent impedance of power supply distribution lines and components such as DC converters and regulators, cause voltage fluctuations that account for most of the power supply noise manifestations we see in digital circuits.

The most nocive parasitic impedance component in power distribution lines is inductance. Unfortunately, inductance is unavoidable in power lines, since it is related precisely to the change of current, as we know from electromagnetism. In addition

to this parasitic inductance, there are further contributions from the power devices. A linear, three terminal regulator contributes between 1 μH and 2 μH of output inductance to the power supply line. Switching regulators have much higher output parasitic inductance. Interconnections also contribute with about 20 nH of parasitic inductance per inch. When an active load is connected to such an impedance, the level of noise created will increase with the rate of change of the supply current, as dictated by Eq. (6.3). The rate of change of current in time increases with the circuit operating frequency, thus, the higher the frequency the more noisy the power supply lines would become.

$$v_L(t) = L\frac{di(t)}{dt} \tag{6.3}$$

Fortunately, most problems caused by power supply noise can be relatively easily mitigated by using simple measures when interconnecting and routing power supply distribution lines. The two most common techniques for reducing power supply noise are *Bypassing* and *Decoupling*.

Bypassing Techniques

Bypassing refers to the act of reducing a high frequency current flow in a circuit path by adding a shunting component that reacts to the target frequency. The most commonly used shunting devices in microprocessor-based designs are bypassing capacitors .

A bypass capacitor reduces the rate of change of the current circulating in the power line by providing a high-frequency, low impedance path to the varying load current. Two factors determine the effectiveness of a bypassing capacitor: size and location.

The size of a bypassing capacitor is selected in accordance to the frequency of the noise they are intended to attenuate. Assuming a supply voltage variation ΔV_{CC}, a supply current I_{CC}, and a worst case transient time ΔT are known, the value for a bypass capacitor C_b can be estimated with Eq. (6.4).

$$C_b = \frac{I_{CC}}{\Delta V_{CC}/\Delta T} \tag{6.4}$$

In most cases, a standard non-electrolytic capacitor between 1 and 0.01 μF shall work. The smaller the value of C_b, the higher the frequency it will shunt. Due to the parasitic components of a real capacitor, namely its nominal capacitance C_b and its equivalent series inductance (ESL), this formula is valid only through the resonant frequency f_c of the capacitor, as illustrated in Fig. 6.6.

The best place for placing a bypassing capacitor is the one that provides the shortest return path to the high-frequency current component being shunted. Such a selection would ensure a small path inductance and therefore would minimize the transient effect it introduces. Based on this premise, the best place for a bypass capacitor to

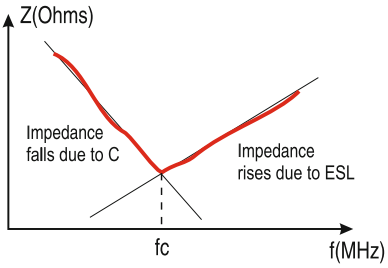

Fig. 6.6 Change of equivalent impedance of a bypass capacitor as a function of frequency

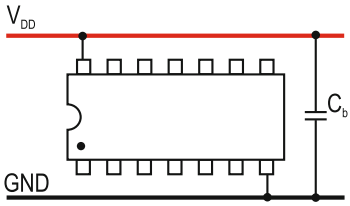

Fig. 6.7 Placement of a bypass capacitor as close as possible to V_{DD}–GND terminals

shunt the induced power supply noise of a packaged MCU or peripheral IC would be the closest possible to the V_{DD}–GND terminals. Figure 6.7 illustrates a possible bypass capacitor placement for a dual-in-line (DIP) package.

Source Decoupling

Decoupling refers to the isolation of two circuits in the same path or connection line. Power supply decoupling is usually achieved through the installation of low-pass filters in strategic points of the power distribution line to reduce the strength of high-frequency noise components from one side of the system, reaching potentially sensitive components in the other side.

AC-to-DC converters, switching regulators, and DC-to-DC converters make particularly noisy sources. In such cases, adding a decoupling circuit at the power supply output helps to isolate the power supply noise from the rest of the circuit. A practical decoupling circuit can be made by adding a fairly large capacitor at the output of the source. In most cases a 10–100 μF electrolytic capacitor shall suffice. If additional filtering were needed, an LC filter at the power supply output could be used. The electrolytic decoupling capacitor would also be large, as above, or even larger for very noisy loads such as DC motors. The value of the inductor, although not critical, is also recommended to be fairly large, with typical values in the range from 10 to 100 mH. In circuits sensitive to power supply ringing, the decoupling circuit could make the power supply underdamped. If this were an issue, a damping resistor of value $R_{Damp} = 2\sqrt{\frac{L}{C}}$ in series with the decoupling capacitor might be added for improved stability. Figure 6.8 shows a decoupling circuit including both, a filter inductor and damping resistor next to an electrolytic decoupling capacitor.

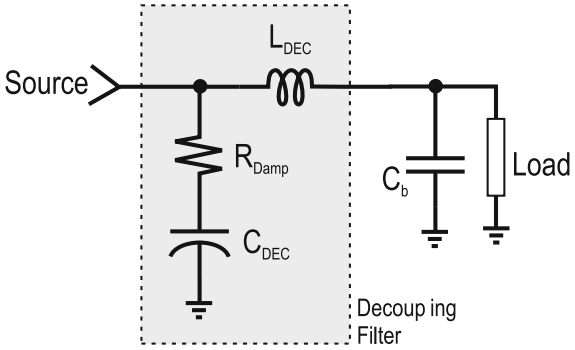

Fig. 6.8 Typical topology of a decoupling LC filter including an optional damping resistor

Analog Versus Digital Supply Lines

Microcontrollers that incorporate analog or mixed-signal circuitry such as embedded OpAmps, data converters (ADCs or DACs), analog comparators, and other functions, commonly provide separate, independent pins in their package for digital and analog supplies. This means separate lines for the digital supply (DV_{DD}) and analog supply (AV_{DD}), and their respective ground leads, *DGND* and *AGND*.

The reason for such a separation is to allow for independently routing of the power lines for analog and digital components. The pulsating operation of digital circuits introduces considerable amounts of noise into their power supply lines. Separation of analog and digital supplies, and their corresponding ground connections, allows for reducing the amount of digital noise reaching the analog components. Although we talk about separate routings, the ground signals must have a common point. To minimize interference, ground lines for analog and digital must meet only at the power supply output. The same rule applies to common V_{DD} lines. Figure 6.9 illustrates this type of connection. The case illustrated assumes a single V_{DD}/GND output at the power supply. Assuming this embedded system contains both, analog and digital circuitry, and the MCU provides for separated supply leads for analog and digital components, the connections shall route them separately.

To achieve such a noise isolation it is necessary to observe good circuit layout techniques. The list below includes several recommendations for making a good layout.

- Common points for both VDD and GND poles shall occur only at the power supply. If the system is enclosed, this is also a good point to ground the enclosure or chassis.
- If using wires for interconnections, physically place analog wiring separated from the digital wiring. Try to place each pair as close as possible to each other. This is, AVDD close to AGND, and DVDD close to DGND. If twisting each pair were possible, go for it. Whenever possible, use thick, sort wire connections to minimize the series impedance.

- When routing the interconnections on a printed circuit board (PBC), try to establish ground planes for both, analog and digital parts. Isolate the planes. Use wide traces for AVDD and DVDD. Route analog traces on the opposite side, right above the AGND plane. Do similarly for the digital traces and DGND plane.
- Use decoupling capacitors at the power supply output and bypassing capacitors at each package, in both, analog and digital sides.
- Try to locate the power supply block in the center of one of the sides of your board. This shall create better chances of separating one side for analog components and other for digital, at both sides of the power supply block.
- Do not surround analog components with digital or viceversa. This would make difficult to separate the routing of analog and digital components.

6.3 Clock Sources

The need for a clock source in microprocessor-based systems arises from the synchronous sequential nature of the digital logic making up most of the system components. The synchronous operation of the control unit's finite state machine (FSM) requires a periodic signal to yield precise transitions between the different states assumed by the CPU. Peripheral circuits also work driven by internal FSMs, requiring steady clock signals as well. Examples of such peripherals include timers, data converters, serial adapters, and displays, among others.

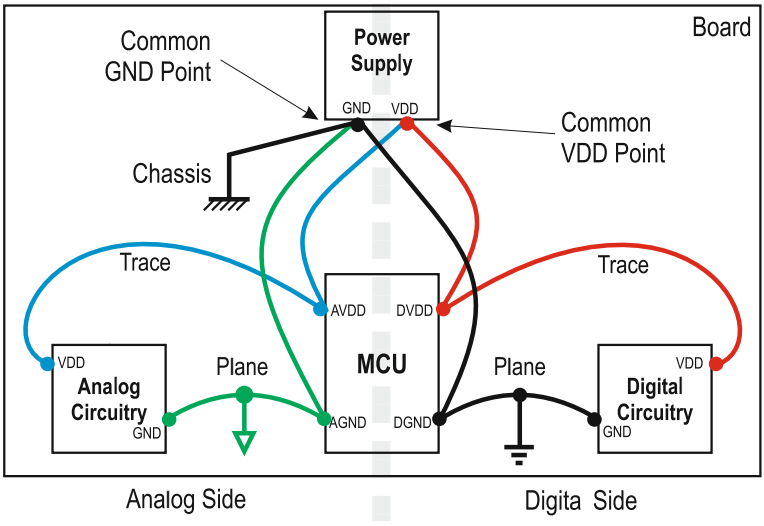

Fig. 6.9 Layout of analog and digital supply lines in a mixed-signal embedded board

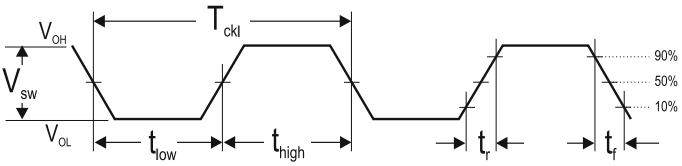

Fig. 6.10 Clock signal timing specifications

In many cases, particularly in MCU-based designs, the clock sources for both the CPU and peripheral devices are derived from the same clock generator, which is designated for that reason *System Clock*.

6.3.1 Clock Parameters

The clock source for a microprocessor system must provide a steady square wave signal. In this context, steadiness implies fixed amplitude swing (V_{sw}), fixed frequency ($f_{clk} = \frac{1}{T_{clk}}$), constant duty cycle ($DC = \frac{t_{high}}{T_{clk}}$), and sharp edges denoted by small rising and falling times (t_r, t_f)). Figure 6.10 illustrates the waveform parameters of a typical digital clock signal.

A minimum set of clock specifications for a microcontroller includes the clock swing, frequency (f_{clk}), duty cycle, and some measure of frequency stability.

The swing of a clock signal ($V_{sw} = V_{OH} - V_{OL}$) needs to match the levels established by the processor's logic levels. Given the processor supply voltage, the swing of the clock signal is expected to bring the low and high values to the safe margins established by V_{IL} and V_{IH}.

The frequency, duty cycle, and stability are determined by the oscillator used to generate the clock signal. The value of these parameters must be chosen to satisfy the requirements specified in the processor's data sheet.

The clock generator is provided entirely outside the chip using dedicated oscillators with square wave output. In microcontrollers, although the use of external, dedicated clock generators is possible, the most common configurations have their oscillators residing either partially or in its totality within the chip. When an entirely on-chip oscillator is used, the voltage swing, the clock frequency, stability, and duty cycle are internally set. Such sources are very convenient since no external component are required to establish the system clock.

However, not every application will benefit from using such internal sources. Most MCUs use resonant RC circuits for these internal clock sources, which as a consequence suffer from limited bandwidth, and are sensitive to temperature and supply voltage variations affecting frequency stability. Taking into consideration such limitations, MCUs clock generators also allow for complementing their oscillators with external components that aid in overcoming the limitations of internal sources.

Clock Frequency

External components are able to establish the clock frequency, duty cycle, and stability, and therefore they can be chosen to satisfy the application expectations, always considering the physical limits specified in the processor's recommended operating conditions. It is also important to remember that the supply voltage level V_{DD} at which the chip is operated imposes an additional limit to the maximum clock frequency at which the MCU can be operated. If we look back at Table 6.2, we would see that for an MSP430G2231, the recommended frequency range goes from DC up to 16 MHz, but the actual maximum frequency will depend on the supply level. For example, when operated with $V_{DD} = 2.7$ V, the maximum achievable frequency is only 12 MHz.

Clock Duty Cycle

The clock duty cycle defines the ratio of t_{high} to the period of the clock signal, expressed in percent. A clock with frequency 1 MHz and duty cycle of 40 % will have a period of 1 μs, and a $t_{high} \cong 400$ ns. The duty cycle is determined by the topology of the oscillator circuit and the value of its components. In the case illustrated in Table 6.2 for the MSP430G2231 the duty cycle is specified at 50 %. This is a common requirement in many modern MCUs.

Clock Stability

The clock stability requirement specifies a statistical measure of the maximum allowable frequency fluctuations of a clock signal over a given time interval. This is necessary because clock signal parameters vary depending on factors that include the oscillator type, capacitive loading, oscillator circuit age, supply voltage, and temperature. In the case of the MSP430G2231, Table 6.2 specifies a maximum frequency deviation of ±10 % from the nominal target value. This means that if for example we set the clock frequency to 12 MHz, the maximum deviation from this value the processor could tolerate would be ±120 KHz. This is a particularly relaxed specification, which is convenient from a design flexibility standpoint. However, the actual acceptable deviation depends not only on physical specifications, but also on the intended application. This situation is illustrated in Example 6.2.

Example 6.2 *Consider a 12 MHz clock signal that exhibits a ±10 % deviation from its nominal value. Two applications employing such a clock signal are analyzed: a dynamic display that sets a 60 Hz refresh ratio from this clock and a real-time clock slated to run uninterruptedly for weeks. Evaluate the impact of the clock accuracy on each system.*

Solution: *The impact of the clock deviation will be analyzed independently for each system.*

Impact on the display system*: A 10 % frequency deviation would translate into an equally proportional deviation in the refresh rate, implying that the actual rate could be 6 Hz off the target value. In the worst case, assuming a 10 % frequency loss, the refresh ratio would be 54 Hz. Considering that for persistence of vision, the human eye only requires 24 Hz of refresh ratio to perceive motion, this 10 % change of frequency can be deemed as negligible.*

Impact on the RTC*: A 10 % deviation in frequency would cause the RTC to drift from the actual time at a rate of 6 s each minute or 2 h and 24 min per day. At the end of only one week, assuming a negative δf_{CLK}, the error would accumulate to 16.8 h, which would be totally unacceptable.*

This simple example highlights the fact that the application itself is what ultimately defines the clock stability requirements.

Deviations from the nominal frequency in oscillator circuits can be caused by multiple factors. They can be generally grouped in factors affecting short or long term frequency stability. *Clock jitter* and *frequency drift* are two illustrative examples.

Clock Jitter

Clock Jitter refers to the uncertainty in the periodicity of a clock signal, manifested as the randomness with which the signal passes through a predefined voltage threshold. Jitter is a complex phenomena in clock signals. Its origin can be related to both stochastic and deterministic factors that include internal noise, particularly thermal noise; power supply variations, loading conditions, and interference coupled from nearby circuits (crosstalk). Its accumulation can lead to data and synchronization errors in digital systems, specially in communications channels. When given as a clock specification, the *total jitter* defines the maximum accumulated period deviation caused by all jitter sources that a processor can tolerate in its clock input over a number of cycles. This is a common specification for microprocessor clock sources.

Example 6.3 *Consider a microprocessor with a total system clock jitter specified at 150 ps under the JESD65B standard. This specification establishes that the total time deviation in the signal period resulting from the sum of all deterministic and random sources in the clock frequency over a minimum of* 10^4 *cycles cannot exceed 150 ps. For some devices, the number of cycles specified in the JESD65B standard establishes a bare minimum. A commonly used number is* 10^5 *cycles, but for some devices it can reach as much as* 10^{12} *cycles.*

Clock Drift

Frequency Drift refers to the linear component of a systematic change in the frequency of an oscillator over time. Frequency drift is typically measured in parts-per-million (PPM). Drift differs from jitter in the sense that jitter is a measure per cycle count while drift refers to the accumulated frequency deviation in time. Aging affects the frequency of many types of oscillators causing a frequency drift that accumulates in

time. For example, a 4 MHz oscillator with an age induced drift of 20 PPM per year will deviate its frequency by 80 Hz every year.

6.3.2 Choosing a Clock Source

The selection of the clock for a microcontroller application, although bounded by the specifications in the MCU data sheet, must be guided by the application requirements. The most important parameter to choose is the clock frequency. When selecting the clock frequency, the designer should look for the lowest frequency that allows for reliable and correct system operation. The source type, topology, and attributes are then decided to satisfy the rest of the parameters.

Some designers might feel tempted to assign the highest frequency allowed by the MCU, regardless of the system requirements. This could result in a poor design. The designer must always analyze the characteristics of the system at hand and take the appropriate decisions to avoid over-design. An overclocked system, besides running faster than needed will consume more energy, will dissipate more heat, and will be more expensive than necessary. Questions that should be answered in the process of selecting the right frequency and configuring the clock include:

- *What is the fastest event the system will need to handle?* The clock frequency must be set to assure that no event goes missed. The fastest process in your system would define the shortest time interval you'll need to keep-up with. Once you identify the frequency for such a process, all the other would fit. Consider for example the clock requirements for an embedded system handling the keystrokes of a typist on a keyboard. Assuming that no other event in this system would surpass the typist speed, a processor running in the KHz range would do fine. The requirements would not be the same for example for the controller of 100 Mbps embedded switch to handle internet traffic.
- *Has the value of V_{DD} been assigned?* As indicated in the previous section, the value of V_{DD} imposes a limit to the maximum clock frequency. Be aware of what this value is to know how high your processor can run. Sometimes the application speed requirement would dictate rising the V_{DD} level or even switching to a faster processor.
- *What peripherals will share the same clock frequency?* In many cases, the peripherals, not CPU speed, will dictate the clock requirements. In control or signal processing applications, where precise sampling periods are required, the speed of the clock driving the timer to set the sampling period and the speed of the ADC are of outmost importance. Other typical considerations arise when a real a time clock or baud rate generator is to be driven by the system clock. The frequency chosen must be the one producing the least error for counting seconds or for obtaining the desired baud rate in the serial communications port.
- *How precise does the clock need to be?* Real time clocks running for extended periods of time (days, weeks, months, etc.) or fast asynchronous serial channels

(19,200 baud or faster) require clocks with small frequency drift and low jitter. Most systems designed to react at the speed of their human oriented user interfaces will probably have low clock precision requirements. The precision requirement along with the actual frequency dictated by the application become primary factors to decide whether the limited accuracy of an on-chip oscillator would suffice or if an external, more precise alternative would need to be considered.

- *What are the capabilities of the Clock System in my MCU?* Understanding the structure and features of the clock support logic in your MCU will certainly simplify the system clock design. Even in the case of using dedicated, external clock generators, knowing the capabilities of the clock generator would allow the designer to take advantage of embedded features that simplify obtaining the necessary clock signal(s).

Certain applications are easier to implement using specific frequency values. For example, a "sweet" frequency for real-time clock applications is 32.768 KHz. The reason for such a sweetness is that by successively dividing this frequency by two, yields a exactly 1.000 Hz with no decimal fractions. Moreover, being $32,768 = 2^{15}$ implies that the division to produce a 1 Hz signal can be readily made with a single 16-bit timer. Note that for example, using a 4 MHz frequency, would not yield the same result. The closest value you will get is 0.953674 Hz. Such an approximation would cause an error of 4.63 % per second in the timekeeping process, which quickly accumulates drifting the measured time from the actual value. Table 6.3 lists some of the most commonly used frequencies in digital systems.

6.3.3 Internal Versus External Clock Sources

When selecting the actual clock source for an MCU-based embedded application, the designer has multiple options to choose from. One of the first questions arising

Table 6.3 Common frequencies for embedded systems applications

Freq. (MHz)	Typical application
0.032768	Real-time clocks. Allows binary division to 1.0000 Hz ($2^{15} \times 1$ Hz)
1.843200	UART clock. ($16 \times 115,200$ baud or $96 \times 16 \times 1,200$ baud)
2.457600	UART clock. ($64 \times 38,400$ baud or $2,048 \times 1,200$ baud)
3.276800	Allows binary division to 100 Hz ($32,768 \times 100$ Hz, or $2^{15} \times 100$ Hz)
3.579545	NTSC M color subcarrier and DTMF generators
3.686400	UART clock (2×1.8432 MHz)
4.096000	Allows binary division to 1 kHz ($2^{12} \times 1$ kHz)
4.194304	Real-time clocks, divides to 1 Hz signal ($2^{22} \times 1$ Hz)
6.144000	UART baud rates up to 38,400.
6.553600	Allows binary division to 100 Hz ($2^{16} \times 100$ Hz)
7.372800	UART clock (4×1.8432 MHz)
9.216000	Allows integer division to 1,024 kHz (2^{10})
11.059200	UART clock (6×1.8432 MHz)

is whether using an internal or external clock source. External clock sources are dedicated oscillators providing a square wave output with specific characteristics of frequency, duty cycle, and stability. These circuits use components such as quartz crystals or ceramic resonators that allow for establishing a base frequency that can be either divided or multiplied to produce virtually any frequency value required by a processor. External sources are more flexible than its internal counterparts in terms of providing a wider range of frequencies to choose from, providing larger driving capabilities, and high levels of accuracy. This flexibility comes at the expense of an increased number of board-level components, increased space, and higher associated cost.

Internal clock sources are common in microcontroller chips. Most contemporary MCUs provide internal oscillators that produce a square wave signal within the chip, requiring no external components at all. This kind of internal sources are convenient for keeping a low count of board-level components, reducing area and cost. However, these sources are frequently based on RC or ring oscillator topologies, that provide less stability and a limited set of frequency values to choose from. To overcome these limitations, the oscillators in MCUs also allow for using external components for completing their clock generators. This feature allows for using more stable references, like quartz crystal or ceramic resonators, or even dedicated external clock generators, at the expense of additional board-level components.

When deciding for an externally assisted configuration, the MCU clock generator might allow for using either RC assisted topologies, crystal-based circuits, or encapsulated external oscillators. RC-based topologies are easy to configure, cheap, and usually require only an external resistor or capacitor connected to the MCU clock inputs. The advantage of this option compared to using the internal MCU clock generator is the added flexibility in setting the clock frequency. However, this option will suffer of the same limitations of any other RC oscillator: limited bandwidth and frequency susceptibility to temperature and voltage changes.

The real advantage of using external components to configure the clock generator comes from the use of quartz crystals. This option allows taking advantage of the superior frequency stability of a crystal-based oscillator, plus the flexibility of a wider choice of frequency values, as the clock frequency is determined by the resonant frequency of the crystal.

6.3.4 Quartz Crystal Oscillators

Quartz is a piezoelectric material, meaning that the application of a controlled external voltage to a properly cut and shaped piece of crystal would deform it to a certain amount. If the external excitation were suddenly removed, the mechanical deformation would produce a voltage difference while it returns to its un-deformed shape. The duration and strength of this signal is a function of the crystal shape. This basic behavior is similar to that of a resistor-capacitor-inductor circuit (RCL) under a similar excitation, which implies the crystal has a very specific resonant frequency.

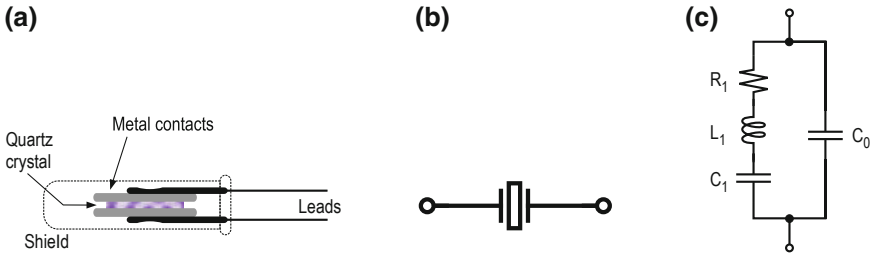

Fig. 6.11 Quartz crystal structure (**a**), symbol (**b**), and equivalent circuit (**c**)

Figure 6.11 shows the structure, symbol representation, and equivalent circuit of a quartz crystal.

The equivalent circuit denotes the nature of the resonance: a series RCL branch that accounts for the piezoelectric property of the material and a parallel capacitor representing the capacitance of the metallic contacts on the crystal. Thus the circuit operates with two resonant frequencies, a series resonant frequency f_s with $L_1 - C_1$ and a parallel resonance frequency f_p with L_1 and the series combination of C_0 and C_1. The expressions for estimating the two resonant frequencies are given by Eqs. (6.5) and (6.6), respectively. Note that $f_s < f_p$. Also, since $C_0 \gg C_1$, their series combination will be close to C_1, making the difference $f_p - f_s$ small.

$$f_s = \frac{1}{2\pi\sqrt{L_1 \cdot C_1}} \tag{6.5}$$

$$f_p = \frac{1}{2\pi\sqrt{L_1 \frac{C_0 \cdot C_1}{C_0 + C_1}}} \tag{6.6}$$

Note that quartz crystals or ceramic resonators do not make oscillators by themselves. These devices need an excitation source and some form of positive feedback to make a sustainable oscillator circuit from them, as is also the case of plain RLC components. Microcontrollers that support external quartz crystal connections provide internal circuitry to complete the oscillator. In most practical applications, the quartz crystal is biased with a—parallel—voltage source which excites the equivalent resonant circuit, making the fundamental frequency of the crystal tend to f_p. In practice, the crystal's fundamental frequency does not reach f_p due to limitations imposed by stray capacitances in the leads and the interconnection.

The two most commonly used oscillator topologies in MCUs are the Pierce and Colpitts configurations. Some processors are hardwired to use only one of them, while others accept being configured to operate with either of them through the connection of external components. Each one offer advantages and disadvantages that should be considered when an MCU allows for either of them to be used.

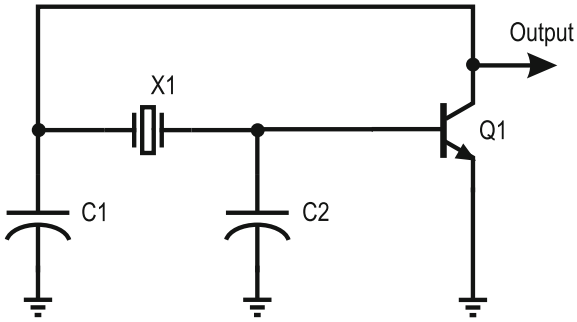

Fig. 6.12 Simplified topology of a Pierce crystal oscillator

Pierce Crystal Oscillators

A Pierce crystal oscillator is a series resonant mode circuit that places the crystal as part of the feedback path. Figure 6.12 illustrates the fundamental configuration of a Pierce crystal oscillator, excluding the bias network for clarity.

External capacitors C_1, C_2 and stray capacitances from the interconnect and the transistor determine the load capacitance seen by the crystal. The combination of these capacitors must closely match the equivalent capacitive load expected by the crystal to be tuned at its fundamental frequency. Moreover, these capacitors also establish the amount of feedback reaching the amplifier. Too little and the circuit might not oscillate, too much and the crystal gets overdriven, causing excess RF emissions, increased power dissipation, and physical wear in the crystal, accelerating aging. For these reasons, the selection of the values for C_1 and C_2 is critical.

The Pierce configuration is the most used topology in microcontrollers. A common configuration is illustrated in Fig. 6.13. This configuration includes two additional resistors: R_S, inserted to reduce the overdrive on the crystal; and R_B, which acts as feedback resistor for the inverting gate, stimulating oscillation. Some MCUs include these resistors on-chip, while others require them as external components. The Pierce topology offers the advantages of less sensitivity to noise and stray capacitances, and starts-up faster. It however, tends to consume more power and if the circuit might experiment accelerated aging the overdrive condition is not prevented.

Colpitts Crystal Oscillator

A Colpitts crystal oscillator is a parallel resonator circuit that removes the crystal from the amplifier's feedback path. Figure 6.14 shows a circuit schematic of basic amplifier and feedback loop for this topology.

In the circuit, the series combination of capacitors C_1 *and* C_2 in parallel to the transistor input capacitance form the load capacitance seen by the crystal. Thus, the selection of the values of these capacitors needs to be carefully done to match the expected crystal load. This configuration produces a DC voltage over the crystal that contributes to aging. Figure 6.15 shows the external connection of a crystal to an

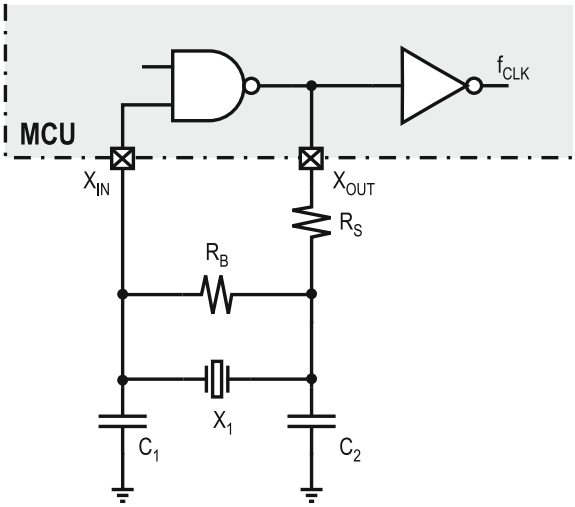

Fig. 6.13 Crystal-based Pierce configuration for digital MCU clock generation

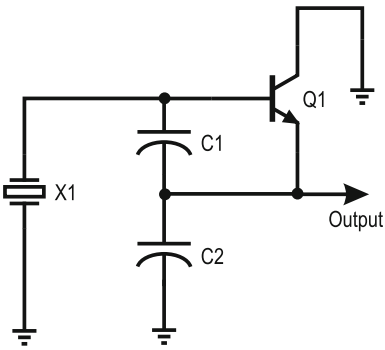

Fig. 6.14 Simplified topology of a Colpitts crystal oscillator

MCU crystal port supporting a Colpitts configuration. Note the insertion of capacitor C_{DC} with the purpose of blocking this DC component. This configuration is also more susceptible to noise and stray components since they appear across the crystal itself. One of the attractive features of the Colpitts oscillator is that when coupled with a branch to limit the output amplitude, it reduces the RF emissions and lowers the power consumption.

Crystal Startup Time

One last note regarding crystal oscillators is about startup time. When initially energized, a crystal oscillator circuit will only produce noise. The frequency component of this noise that matches the phase condition for oscillation will be amplified and fed back to the input. As the oscillator operates with positive feedback and gain greater than one, the amplitude will grow until the amplifier gain is reduced to one by an

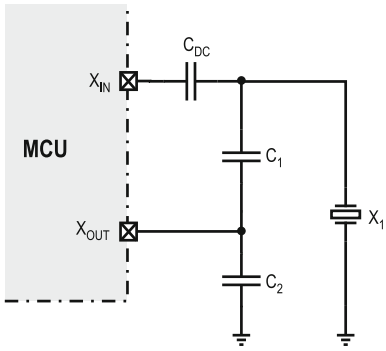

Fig. 6.15 Crystal connection to MCU pins for a Colpitts configuration

automatic gain control loop in the oscillator (not illustrated in the figures above). The time elapsed from power-up until the oscillation amplitude reaches its nominal value is called the *oscillator startup time*.

The startup time in a crystal oscillator can be anywhere from several hundred milliseconds to several seconds. Factors influencing the actual startup time for a particular crystal include the frequency value, quality factor Q, and load capacitance seen by the crystal, among others. The lower the frequency of a crystal, the longer its startup time. High Q-factor crystals or those having a large load capacitance will also have a slow startup. In contrast, RC-based oscillators with their low Q-factors exhibit a quick startup.

Startup times play an important role when the MCU is operated in a low-power mode that shuts down the clock generator. Upon wake-up, the oscillator needs to be restarted going through its startup process. As we will see later, there are ways to reduce the startup time for cases like this. A note of caution here: it is possible to reduce the startup time by overdriving the crystal. But be aware, this would increase power consumption in the clock generator and would also accelerate the aging process of the crystal.

6.4 The MSP430 System Clock

The system clock in MSP430 devices provides synchronization signals for the CPU and embedded peripherals. The system is designed for providing low cost clocking alternatives and supporting the processor's low-power modes.

Throughout the successive generations of MSP430 controllers launched by Texas Instruments, the system clock topology has evolved, growing in the number of clocking options and their capabilities. Three fundamental topologies have been introduced for the six generations of controllers in the MSP430 family. Each topology has been used through the generations of MCUs as listed below.

- The Frequency Locked Loop (FLL) Clock Module used in MSP430x3xx devices and its "plus" version used in MSP430x4xx generation.
- The Basic Clock Module and its "plus" version used in generations MSP430x1xx and MSP430x2xx, respectively.
- The Unified Clock System (UCS), used in the fifth and sixth generation MSP430x5xx and MSP430x6xx devices.

The next sections provide insight into the evolution of these topologies, highlighting key features and differences among them.

6.4.1 The FLL and FLL+ Clock Modules

The first two generations of MSP430 devices, series x3xx and x4xx, both use the same basic clock generation design, the *Frequency Locked-Loop (FLL) Clock Module*. This fundamental design was introduced with x3xx devices, and an improved version included in x4xx devices. The discussion below provides details on these clock modules.

The FLL Clock Module

The earliest generation of the MSP430 device family, the MSP430x3xx uses a *Frequency-Locked Loop* clock module as system clock generator. This clocking system consists of two oscillators: a Pierce crystal oscillator and a frequency stabilized, RC-based, Digitally Controlled Oscillator (DCO). The DCO is locked to a multiple of the crystal frequency, forming a frequency-locked loop (FLL). This clocking system fundamentally provides two clock signals: the *Main Clock* (MCLK), taken from the DCO output and an *Auxiliary Clock* signal (ACLK), taken out from the crystal oscillator. A buffered, software selectable output XBUF is also available, that provides as output either MCLK, ACLK, ACLK/2, or ACLK/4. The XBUF clock can also be turned off via software. The MCLK signal is used to drive the CPU, while ACLK and XBUF can be software selected to drive peripherals. Figure 6.16 shows a simplified diagram of the FLL Clock Module.

The crystal oscillator in the FLL module is designed to support an external 32.768 KHz quartz crystal with no need for additional external components. Load capacitors are internally provided and set to 12 pF. The oscillator can be turned off via control bit OscOff. Note that without an external crystal, there would be no ACLK signal, nor its derivatives through XBUF. ACLK can also be fed by an external square waveform, connected through pin Xin. In such a case the crystal oscillator would be bypassed and its internal circuitry deactivated.

The DCO is a software programmable, on-chip ring oscillator with RC-like characteristics. The DCO frequency is set via a multiplying factor K obtained as $K = N + 1$, where N is a value set via software. Upon a valid reset (PUC),

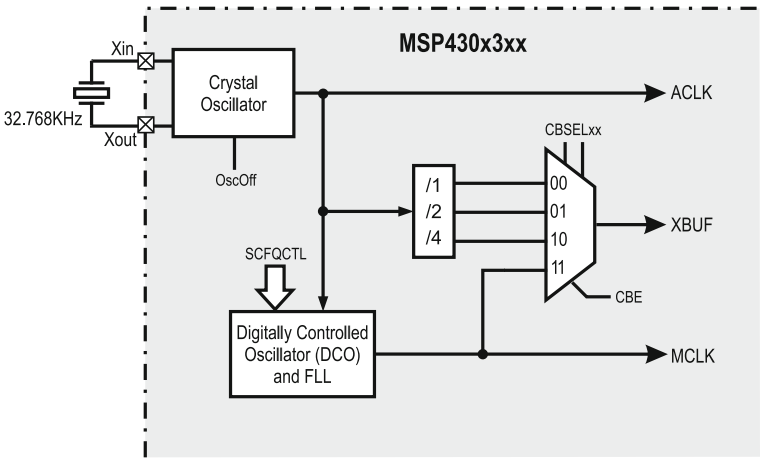

Fig. 6.16 Simplified block diagram of FLL clock module in MSP430x3xx devices

N is preset to 31 causing the system to begin operation at a frequency 32 times the crystal oscillator frequency. That is, if the external crystal frequency were 32.768 KHz, then, upon a PUC, MCLK would have a frequency of 1.048576 MHz. Under software control, the multiplying factor can be set to any value $1 \leq (N+1) \leq 128$ via the system clock frequency control register (SCFQCTL). The least significant seven bits in SCFQCTL allow setting N to any value between 1 and 127. Note that as with any RC oscillator, the DCO frequency will vary with temperature and voltage. However, the inclusion of a FLL that tracks the crystal oscillator output also allows stabilizing the DCO frequency. The stability achieved brings the frequency variability to that of the external crystal connected to the chip. In addition, by using a digital FLL and DCO, the system clock exhibits a short startup time similar to that of an RC oscillator.

In the absence of the ACLK signal, an internal oscillator fault condition would be created. This fault could be caused by a crystal failure or by just the absence of an external crystal in the chip interface. This condition would trigger a non-maskable interrupt to the CPU if the oscillator-fault interrupt enable (OFIE) bit in the interrupt enable register 1 (IE1) was enabled. This event however, does not prevent the DCO from generating an unlocked frequency on the MCLK output. In fact, OFIE is by default clear upon PUC which allows the processor to operate without an external crystal. Take into consideration that under this condition the FLL would be unable to compensate for the inherent frequency variations of the RC oscillator.

The FLL+ Clock Module

The second generation of MSP430, the MSP430x4xx, features an improved clock generator designated the *FLL+ Clock Module*. This improved module, in addition

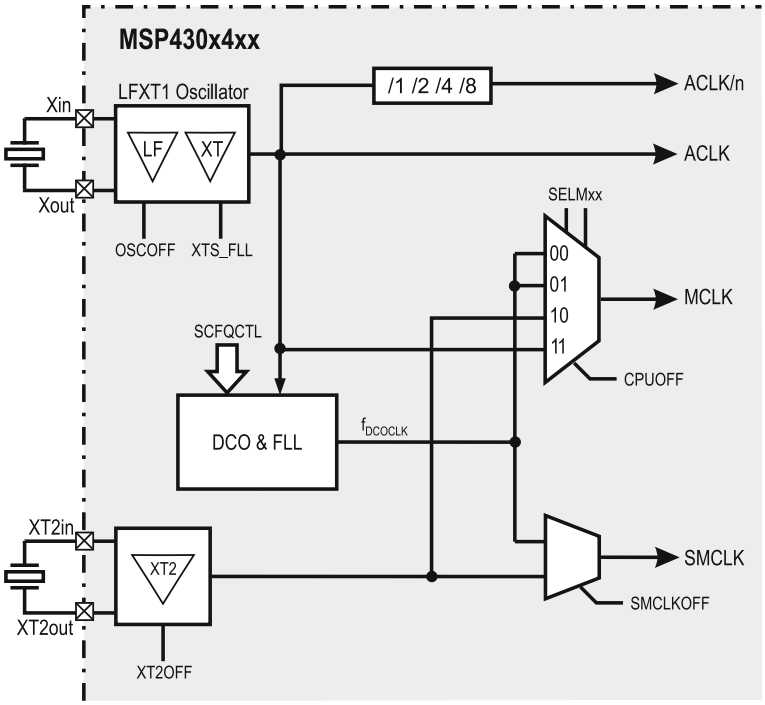

Fig. 6.17 Simplified block diagram of FLL+ clock module in MSP430x4xx devices

to LFXT1, DCO, accommodates an optional secondary oscillator, XT2, included in selected family members. The FLL+ module can provide four clock signals to the system: ACLK, ACLK/n ($n = 1, 2, 4$, or 8), MCLK, and SMCLK. Figure 6.17 shows a simplified block diagram of the FLL+ clock module.

Oscillator LFXT1 is an improved version of the Pierce crystal oscillator of x3xx devices. It can now accept crystals of frequency higher than 32.768 KHz by including an additional high-speed feedback amplifier (XT) capable of operating with crystals in the range from 400 KHz to 8 MHz. Control bit XTS_FLL allows for choosing the correct amplifier for the kind of external crystal connected. The value of the internal load capacitors can now be selected via software to match the connected crystals, or optionally, externally provided. The LFXT1 oscillator feeds clock signal ACLK. As before, LFXT1 can be bypassed by an external clock fed via pin Xin. ACLK/n is a submultiple of ACLK software selectable to be divided by a factor of either 1, 2, 4, or 8.

The DCO is essentially the same as in the previous generation. As before, it operates in an FLL tracking a multiple of ACLK, and providing the same advantages of frequency stability and quick startup.

Oscillator XT2 is a new addition in this generation, but is present only in selected family members. It has the same capabilities as the high-frequency portion of LFXT1,

except that it does not incorporate on-chip load capacitors, which must be externally provided.

The main clock (MCLK) can now be fed either from ACLK, the DCO, or XT2 (if present), and provides the same failsafe capabilities as in the previous generation. FLL+ allows for the CPU to be operated from the DCO without external crystals. Clock signal SMCLK (sub-main clock) replaces the old XBUF, and can now be fed from either the DCO or XT2 (if present).

6.4.2 The Basic Clock Module and Module+

Generations x1xx and x2xx featured a modified clock module with respect to the design included in earlier generations. This new clock generator is designated as *Basic Clock Module*. The basic clock module design is introduced in x1xx devices, and an improved version, the *Basic Clock Module+* featured in x2xx devices.

The Basic Clock Module

The basic clock module in MSP430x1xx devices is a revised design of the FLL+ used in the earlier generation. This somewhat simpler, and yet improved version of the FLL+ still includes three oscillators (LFXT1, XT2, and DCO). However, the DCO is now redesigned to operate in open loop (no FLL), while offering a better frequency stability without the need of an external crystal. In addition, only three clock signals are made available: ACLK, MCLK, and SMCLK. Figure 6.18 shows a simplified block diagram of the basic clock module.

The structure of oscillator LFXT1 in this generation of the MSP430 retains most characteristics from its predecessor in series x4xx. It supports both low (32.768 KHz) and high frequency (450 KHz to 8 MHz) external crystals, selectable with the XTS control bit. Although the LF amplifier still includes on-chip load capacitors, this time fixed at 12 pF, its high-frequency counterpart, XT, does not. Load capacitors for high-frequency crystals must be externally provided. Bypass capability with an external clock is maintained.

Secondary oscillator XT2, available in pins XT2in/XT2out, maintains the same structure as before: high-frequency only, crystal-based, requires external load capacitors, and is available only on selected devices.

The major change of this generation system clock is in the DCO. The new DCO uses a current-controlled ring oscillator with RC-type characteristics. It is redesigned to sustain a stable programmed frequency output without the need of an external crystal and to provide an even faster startup time. This new DCO has eight frequency taps and does not include an FLL feedback loop. Note in Fig. 6.18 that now the DCO is independent from ACLK. The DCOCLK frequency now depends on three factors:

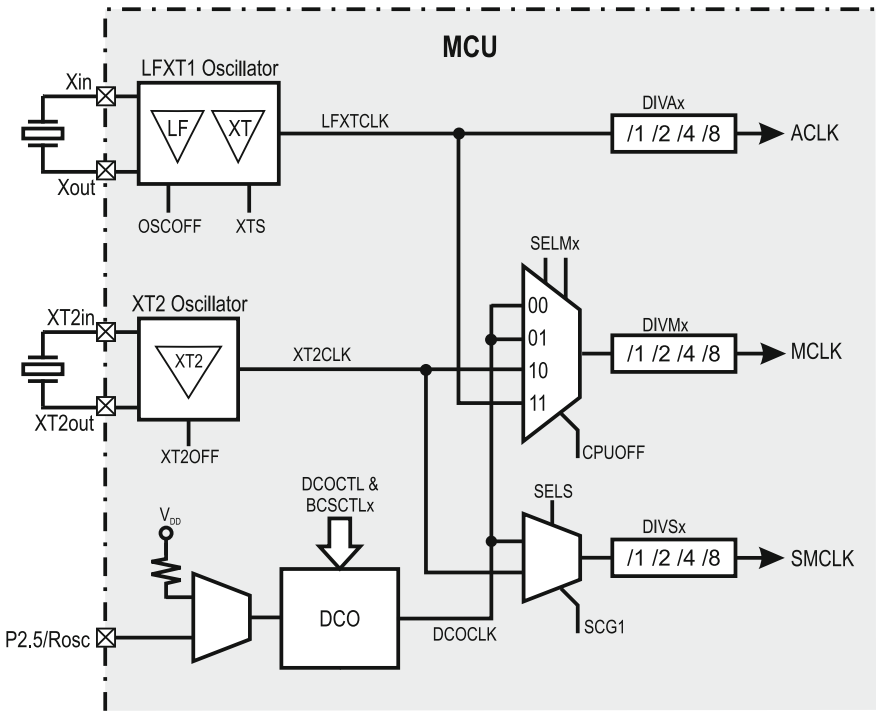

Fig. 6.18 Simplified block diagram of basic clock module in MSP430x1xx devices

- A resistor-controlled current injection. Internally, the DCO incorporates a DC generator whose current output is controlled by either an internal or externally provided resistor. Control bit DCOR allows choosing either. Using an external resistor provides better tolerance to temperature and supply voltage changes.
- The value assigned to the DCO frequency select bits in the DCOCTL register. These three bits select which of the eight frequency taps in the DCO internal modulator will be used for the output frequency. Each successive tap has a frequency which is approximately 10 % higher than the previous one.
- The adjustment introduced by the modulator specified in the MODx bits of DCOCTL register. Each modulator can automatically adjust the selected frequency tap to maintain the clock frequency within bounds.

The specific combination of values for the modulator and DCO frequency varies for each desired frequency. The MSP430 header file of the specific device used in a particular application contains pre-defined values to accurately obtain a set of pre-calibrated frequencies with the DCO. The designer is referred to these values in the corresponding data sheets for a straightforward use of this resource in the basic clock module. General formulas for a wider range of frequencies are provided in the MSP430x1xxx user's guide.

The outputs from the three oscillators can be distributed through the clock signals ACLK, MCLK, and SMCLK as follows. The auxiliary clock ACLK can be fed exclusively from LFXT1. Remember that ACLK can be used to clock on-chip peripherals such as timers, baud rate generators, and data converters. The main clock (MCLK) has the most flexibility as it can be fed with either of the three oscillators. ACLK, as explained before, is the CPU clock and can be assigned to some peripherals as well. SMCLK, the sub-main clock can have as sources either the DCO or XT2, if present in the particular chip being used. All three clock signals can be divided via software control through respective frequency dividers located at the output of each selector, as illustrated in Fig. 6.18. Failsafe capabilities allow detecting failures in either of the high-frequency oscillators. There is no fault detection for LFXT1 in LF mode.

The Basic Clock Module+

The fourth generation of the MSP430, the MSP430x2xx, expands the basic clock module design by introducing several new features that include a fourth oscillator, and increasing the maximum clock frequency for the system. This new clock module, designated the *Basic Clock Module+*, inherits most features of the basic clock module. Figure 6.19 shows a simplified diagram of the basic clock module+.

The improvements can be summarized as follows:

- Addition of a new clock source, the very-low-power, low-frequency oscillator (VLO). With a typical frequency of approximately 12 KHz, the VLO is an internal RC type clock source that can replace the LFXTCLK in ACLK. Control bits LFXT1Sx allow for selecting between VLOCLK or LFXTCLK.
- Inclusion of minimum pulse filters (MPF) at the outputs of all clock generators to prevent spurious pulses propagating through the clock system. Observe the LPF blocks at the outputs of DCO, LFXT1, and XT2 oscillators.
- The internal resistor to set the DCO frequency is replaced with a current source, for added precision in the internal setting of the DCO DC signal.
- The high-frequency oscillators and DCO are now capable of operating up to 16 MHz. This is twice the maximum frequency achievable in the original basic clock module.
- Addition of a fault detection indicator for failure in the LFXT1 oscillator.
- Oscillator LTXF1 now has internal programmable load capacitors on-chip.

Not all these features are included in every x2xx device. All other remaining characteristics from the original basic clock module remain the same in these new devices.

6.4.3 The Unified Clock System

The last MSP430 generation reviewed, the MSP430x5xx/x6xx, uses the most comprehensive system clock designed so far for these controllers. The *Unified Clock*

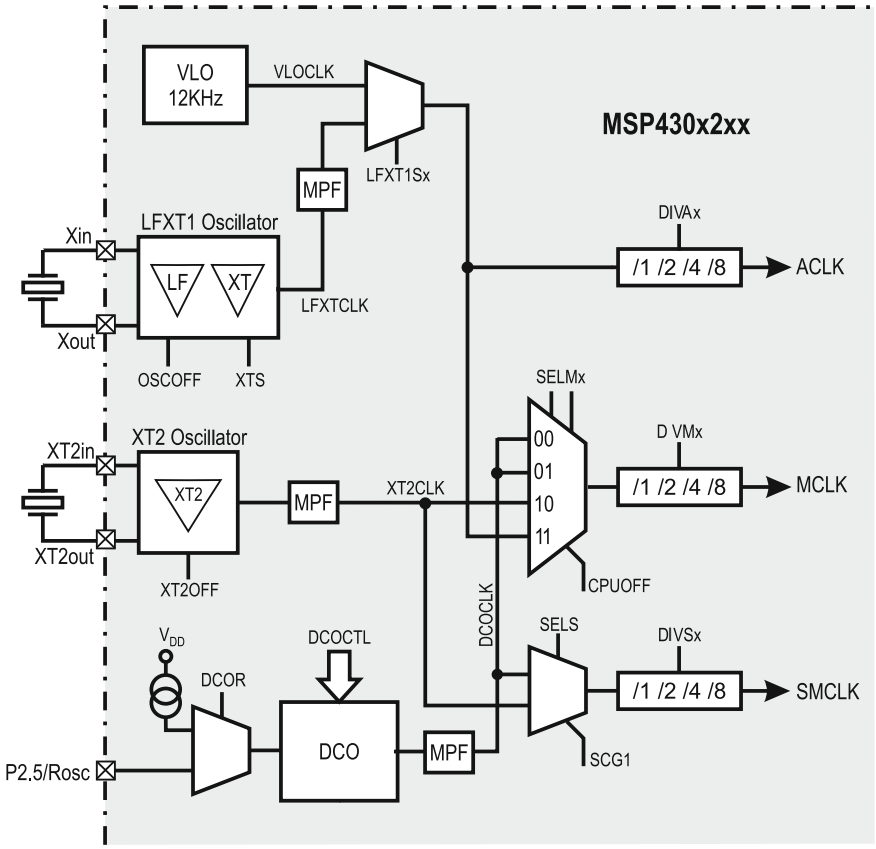

Fig. 6.19 Simplified block diagram of basic clock module+ in MSP430x2xx devices

System (UCS) features functional attributes developed in previous generations now consolidated in one clock system. It incorporates five general oscillators, XT1, VLO, REFO, DCO, and XT2 that can be equally used for clocking the CPU or embedded peripherals through three main clock signals ACLK, MCLK, and SMCLK. A sixth dedicated oscillator, MODOSC and its output MODCLK, are reserved for clocking flash memory and certain peripherals. Any of the five general oscillators can be assigned to any of the clock outputs, as denoted in the simplified block diagram of the unified clock system shown in Fig. 6.20.

As in previous generations, oscillator XT1 maintains its capability of handling either low-frequency (LF) or optionally, high-frequency (HF) crystals through pins Xin, Xout. Control bit XTS for selecting the type of crystal connected. LF remains intended for 32.768 KHz crystals, while HF is now redesigned to accept crystals or clock sources in a range from 4 to 32 MHz. On-chip software selectable load capacitors make unnecessary to use additional external components to run from LF.

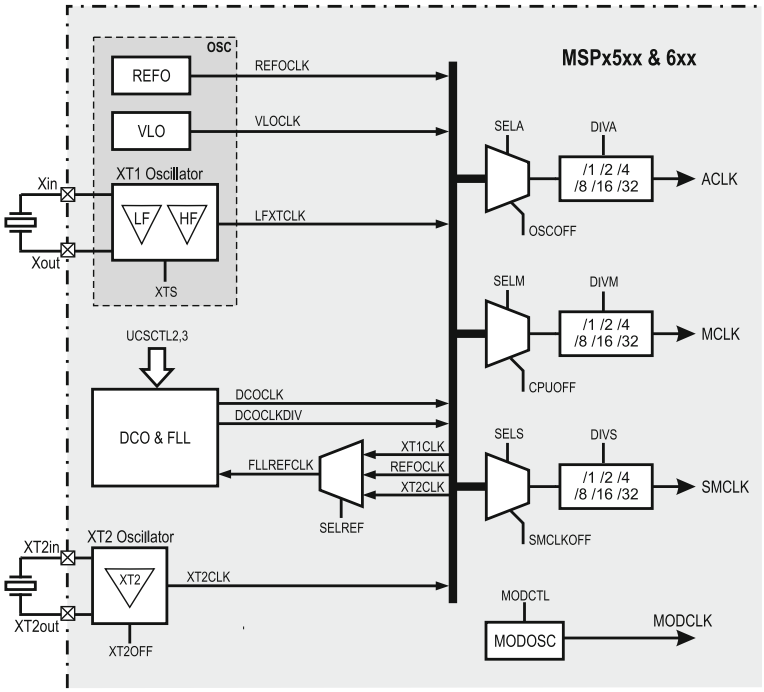

Fig. 6.20 Simplified block diagram of the unified clock system in MSP430x5xx/x6xx devices

HF requires external capacitors. Xin maintains its capability to be bypassed with an external clock signal.

As in previous generations, optional oscillator XT2 runs with high-frequency crystals. Its frequency range is improved to match that of XT1-HF.

The very-low-power, low-frequency oscillator (VLO) introduced in the BCM+ is retained in this generation, now with a nominal frequency of 10 KHz. This RC-type oscillator is optimized for low power, with typical current requirements as low as 10 nA.

A new oscillator appearing in the UCS is the REFO, an internal trimmer low-frequency reference oscillator. REFO is trimmed to 32.768 KHz and intended to be used as a crystal free reference frequency for the DCO, although it can be used as any of the other oscillators.

The DCO in the UCS returns to its roots, with the ability to operate in a frequency-locked loop (FLL). In this revised version, the DCO frequency can be adjusted by software or stabilized by the FLL to a multiple of FLLREFCLK/n, for $n = 1, 2, 4, 8, 16,$ or 32. The FLL reference clock, FLLREFCLK, can be software configured to be either XT1CLK, REFOCLK, or XT2CLK. The DCO frequencies DCOCLKDIV and DCOCLK are configured through the UCS control registers UCSCTL2 and UCSCTL3, assigning factors D, $N + 1$ and n to conform Eqs. (6.7) and (6.8).

Table 6.4 Summary of clock configurations for MSP430 generations x1xx through x6xx

MSP430	Name	Oscillators	Clock signals	Max freq. (MHz)	DCO type
x3xx	FLL	XTAL, DCO	MCLK, ACLK,XBUF	0.032	FLL
x4xx	FLL+	LFXT1, DCO, XT2	MCLK, SMCLK, ACLK, ACLK/n	8.000	FLL
x1xx	BCM	LFXT1, DCO, XT2	MCLK, ACLK, SMCLK	8.000	DCO
x2xx	BCM+	LFXT1, DCO, XT2, VLO	MCLK, ACLK, SMCLK	16.000	DCO
x5xx	UCS	LFXT1, DCO,	MCLK, ACLK	32.000	FLL/DCO
x6xx		XT2, REFO, VLO, MODOSC	SMCLK		

$$f_{DCOCLKDIV} = (N+1) \times \left(\frac{f_{FLLREFCLK}}{n} \right) \tag{6.7}$$

$$f_{DCOCLK} = D \times f_{DCOCLKDIV} \tag{6.8}$$

When the FLL is disabled, achieved by deactivating either the internal DC generator or frequency integrator, the DCO continues to operate as it did in the basic clock system of series x1xx/x2xx. This implies that f_{DCOCLK} is defined by the DCO frequency range selection and divider (DCORSEL) and the DCO modulator pattern bits (MOD). When enabled, the FLL is controlled by the DCO frequency tap and MOD specified in UCSCTL0. The specific frequency values that can be obtained depend on the particular device being programmed, as specified in the corresponding data sheets.

The clock distribution system allows assigning any of the six oscillator outputs REFOCLK, VLOCLK, LFXTCLK,XT2CLK, DCOCLK, or DCOCLKDIV to any of the clock signals ACLK, MCLK, or SMCLK. The clock signals are served divided by a factor $n = 1, 2, 4, 6, 8, 16, 32$, as specified in the corresponding DIVx control bits.

6.4.4 Using the MSP430 Clock

The clock options in the different generations of MSP430 can be summarized as illustrated in Table 6.4. There we can see that the low-frequency crystal oscillator and the DCO are a common denominator in all MSP430 generations. We can also see that the architecture has converged to providing MCLK, SMCLK, and ACLK as the clock signals for all the synchronization needs in the MCU.

Despite the number of different clocking alternatives provided by the MSP430, probably more than 90 % of the design needs can be satisfied with simple topologies and setup as illustrated in the examples below.

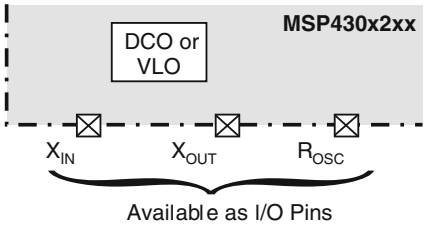

Fig. 6.21 Hardware setup for internal clock generation in an MSP430F2274

Example 6.4 *Provide the hardware setup and initialization sequence to have the basic clock module in an MSP430F2274 operating on the internal DCO at 1 MHz with internal current source. Route ACLK from the VLO and turn off unused internal oscillators.*

Solution: *This configuration is one of the simplest that can make the MSP430 quickly usable. The hardware setup requires no external components, leaving all oscillator related pins (Xin, Xout, and Rosc) available for use as I/Os. Figure 6.21 shows the hardware setup for this configuration.*

Note that although this example calls for using the DCO, the above hardware setup would also be valid if we were using the VLO as the internal clock.

To internally configure the oscillator, we first note that the default configuration upon power-up (PUC) has clocks MCLK and SMCLK fed by the DCO and running with its internal current source at a frequency of approximately 1.15 MHz. This is because upon PUC, the DCO is configured to use its internal current source (DCOR = 0) with DCOx = 3, MODx = 0, RSELx = 7. The auxiliary clock ACLK, is sourced from LFXT1, but in this case it would be inactive since there is no external crystal connected. Therefore, to avoid energy waste it would be necessary to deactivate LFXT1. In summary, all that is needed to satisfy the requirement is configuring ACLK to be fed from VLO and turn off LFXT1. The only action required for achieving both objectives would be setting LFXT1Sx = 10b. This would not only route ACLK from the VLO but will also disable LFXT1. This can be achieved by executing the instruction

```
MOV.B #LFXT1S0,&BCSCTL3 ;Set ACLK feed from VLO and deactivate LFXT1
```

Example 6.5 *Provide a hardware setup and initialization sequence to have the BCM+ in the MSP430F2274 operating from the DCO at 2.0 MHz. Use an external resistor for a temperature tolerant DCOCLK frequency. Assume no peripheral will use ACLK.*

Solution: *The hardware setup illustrated in Fig. 6.22 shows a connection to use an external resistor to set the DCO reference current. Such a setup reduces the temperature sensitivity of DCOCLK.*

By default, after a PUC DCOCLK sources MCLK, LFXT1 is on, and DCOR is set to the internal current source. A PUC also makes DCOx = 3, MODx = 0, and RSELx = 7. These settings make $f_{DCO} \cong 1.15\,MHz$. To get $f_{DCO} = 2\,MHz$ with an external resistor we need to change the default configuration. Since LFXT1 is not going to be used, we should turn it off to save power. Consulting the MSP430F2274 data sheet (SLAS504F), the DCO characteristic with external resistor shows that an appropriate setting to get $f_{DCO} \cong 2.00\,MHz$, with $V_{DD} = 3.3\,V$ would be making

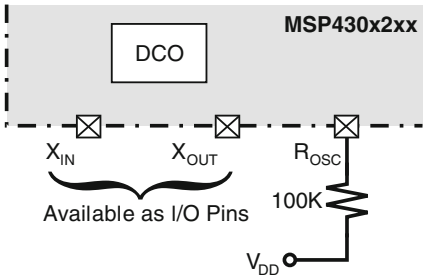

Fig. 6.22 Hardware setup for resistor sourced DCO in an MSP430F2274

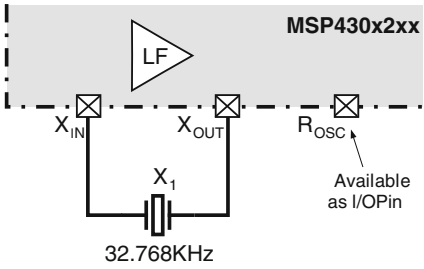

Fig. 6.23 Hardware setup for clocking an MSP430F2274 with a low-frequency crystal

$R = 100\,K$, $RSELx = 4$, and leaving the default values on DCOx and MODx. The following code fragment accomplishes the objective:

```
; Assumes Vdd = 3.3 V and reset values in MODx and DCOx
BIS.B #DCOR,&BCSCTL2        ; Rosc set to external
BIC.B #RSEL1+RSEL0,&BCSCTL1 ; RSEL = 4
BIS.W #OSCOFF,SR            ; XTAL not used
```

Example 6.6 *Use an external 32.768 KHz crystal to run the MSP430F2274 ACLK at the crystal frequency and the CPU at 8 MHz. Set SMCLK at 2 MHz.*

Solution: *In this case, we can connect an external 32.768 KHz crystal through XTAL1 in LF mode, set the DCO to feed MCLK at 8 MHz, and run SMCLK from the DCO, dividing the output via DIVSx by a factor of 4. The hardware setup is quite simple, as illustrated in Fig. 6.23. Since no external resistor is being used, Rosc pin remains free to be used as I/O.*

Upon power-up, XTAL1 in LF mode becomes the source of ACLK, so we can leave it with its default configuration. The DCO frequency desired in this case can be obtained by configuring DCOx, MODx, and RSELx. All we need is to step-up the DCO frequency from its default value of 1.15–8 MHz. This could be achieved with DCOx and MODx in their default values of three and zero, respectively, and changing RSELx to 13. With a 3.3 V supply voltage it would provide approximately 8.1 MHz. However, $f_{DCO} = 8\,MHz$ is one of the calibrated values for this chip, so we will instead resort to this setup. Calibrated values configure DCOx, MODx, and RSELx bits, clearing all other bits in DCOCTL and BCSCTL1. Note that this action will leave XT2 on, which for this example needs to be explicitly turned off. We'd

also need to have SMCLK sourced from DCOCLK and divided by a factor of 4 to get 2 MHz. After PUC, SELS = 0, which sets SMCLK = DCOCLK, so no change needed there. DIVSx is by default 00b and needs to be changed to 10b. The code fragment below would achieve the desired effect.

```
; Assumes Vdd = 3.3 V and reset values in MODx, DCOx, and BCSCTL2
MOV.B &CALBC1_8 MHZ,&BCSCTL1 ; Set range for calibrated 8 MHz
MOV.B &CALDCO_8 MHZ,&DCOCTL  ; Set DCO step + modulation for 8 MHz
BIS.B #XT2OFF,&BCSCT1        ; XTAL2 not used
BIS.B #DIVS_2,&BCSCTL2       ; SMCLK divided /4 = 2 MHz
```

6.5 Power-On Reset

The sequential nature of the CPU's control unit, besides a clock, also requires a RESET signal for taking its finite state machine (FSM) to its initial state upon power-up. All sequential circuits have such a requirement. The reset signal in a microprocessor-based system causes several actions in the CPU and its surrounding logic. The specific list of actions changes from one processor to another, but there is a list of tasks that are common to most architectures. These include:

- Loading the program counter register (PC) with the address of the first instruction to be executed. This causes execution of the first instruction in the booting sequence.
- Disabling the reception of maskable interrupts by the CPU.
- Clearing the status register (SR). The specific value loaded into the SR changes from one processor or MCU to another.
- Initializing some or all system peripherals (list changes for specific devices). For example many MCUs set all their I/O pins to input mode, timers are initialized to zero, and the default CPU operating mode is selected.
- Canceling any bus transaction in progress and returning control to the default bus master (details of bus Arbitration in Chap. 7).

The RESET signal in an embedded system is generated through a specialized circuit called a *power-on reset circuit* or POR for short.

6.5.1 Power-On Reset Specifications

The specifications of a reset signal change from one chip manufacturer to another and even for the same manufacturer, among device families. Despite this, there are a few requirements than can be generalized to virtually any processor. These include the assertion level of the reset pulse, and the minimum pulse width of the signal to be accepted as a valid reset. Figure 6.24 illustrates a typical reset timing for a generic CPU.

When power is applied to a system, the supply voltage ramps-up until reaching its nominal value V_{DD}. During this period, the oscillator providing the clock signal

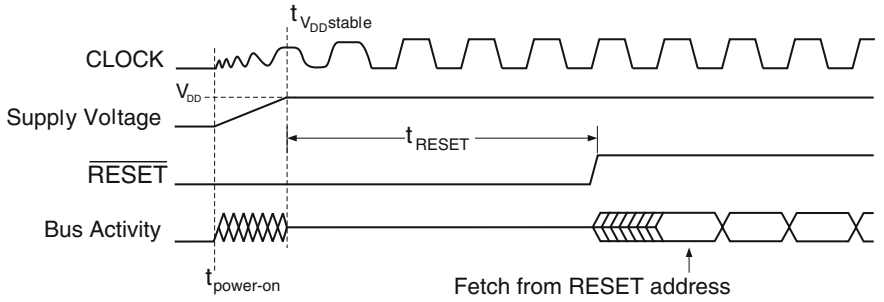

Fig. 6.24 Typical CPU RESET timing

begins its startup time. The reset signal, in this example assumed to be asserted-low, must remain active at least for t_{RESET} time after V_{DD} has stabilized for the CPU recognizing it as a valid reset. Bus activity might show some noise, but usually the busses are held inactive until the reset signal is de-asserted. Upon RESET de-assertion, the CPU begins fetching instructions from program memory, starting at the reset address. The function of the POR circuit is to provide such a timing to the reset signal.

6.5.2 Practical POR Circuits

There are different alternatives for supplying a POR signal to a microprocessor or microcontroller. Many microcontrollers have their internal reset source, requiring minimal or no external components. Sometimes a custom-built circuit using discrete components, able of providing the minimal reset requirements would suffice. In other cases, the application requirements would call for dedicated reset supervisor ICs. We discuss these cases below.

Embedded POR Circuits

Contemporary microcontrollers frequently include some form of on-chip POR circuit, reducing the count of external components needed to implement an application. When they do, they also provide the option of feeding an externally generated reset signal to the CPU. The MSP430 includes an on-chip POR circuit that requires no external components for operation. Section 6.7 provides a detailed discussion of the MSP430 reset structure.

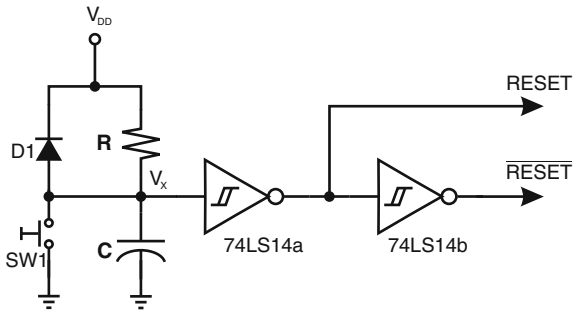

Fig. 6.25 A POR circuit based on a one-shot topology

One-Shot-Based POR Circuit

Most solutions for POR circuits are based on the structure of a monostable multivibrator, also known as a one-shot circuit. This circuit operates between two states: one stable and one unstable. The unstable state is entered when a perturbation to the stable state is triggered. The duration of the unstable state depends on the time constant of some energy storing component in the circuit. In most cases this is implemented with a simple RC circuit. When using a one-shot as a POR circuit, its time in the unstable state is used to produce the desired reset timing. Figure 6.25 shows a practical implementation of a power-on reset circuit based on a one-shot topology. Resistor R and capacitor C determine the circuit time constant that defines the reset pulse width.

Before power is applied to the circuit we assume capacitor C is discharged. At power-up, C begins to charge through R, with time constant RC. When Vx reaches the positive threshold of the Schmidt triggered inverter (V_T^+), this causes an abrupt change in the inverter's output. The second inverter provides the correct level of assertion for an active low reset signal. Figure 6.26 shows the waveforms in the critical circuit nodes. The width of the reset pulse is determined by the time constant RC, the supply level V_{DD}, and the positive threshold voltage of the inverter V_T^+, as given by (6.9).

$$t_{reset} = RC \cdot \ln\left(\frac{V_{DD}}{V_{DD} - V_T^+}\right) \tag{6.9}$$

Note that for a given requirement of t_{reset}, Eq. (6.9) is undetermined. What is commonly done to resolve the equation is to assign a suitable value of C and calculating the value of R. In most cases a capacitor around 1 μF should work, as pulse widths for reset are at most in the range of milliseconds.

Figure 6.25 also features a couple of additional devices: D1 and SW1, which provide additional functionality to the basic POR circuit.

When the supply voltage drops below its nominal value, it might reach a level at which correct device functionality might not longer be guaranteed. Such a condition is called a *Brownout* event. Diode D1 in Fig. 6.25 acts as an elemental brownout detector in the POR circuit. It detects voltage drops in the supply line, triggering

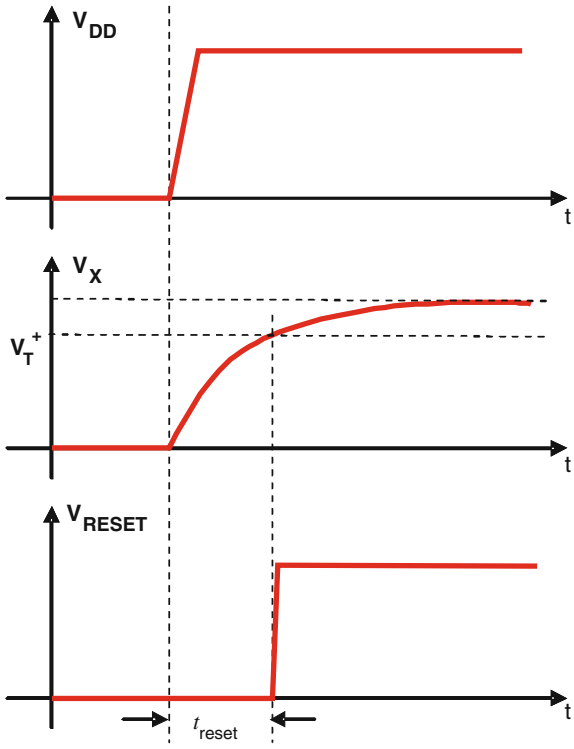

Fig. 6.26 Waveforms in the critical nodes of the one-shot based POR circuit

a POR if a threshold is reached. Its operation can be explained as follows: After a sufficiently long time of operation, Vx shall have reached V_{DD}, which would reverse bias D1. If V_{DD} falls at least one diode drop below its nominal value, D1 would become forward biased, providing a low-impedance discharge path for C1. This would trigger the circuit into its unstable state, producing a valid reset pulse.

Sometimes it might result convenient to manually trigger a system reset without having to remove power from the system. For such situations, switch SW1 provides for manually discharging C1. Note that when SW1 is closed it creates a short-circuit to the capacitor terminals, making it possible to manually reset the system without the need of removing or dropping the supply voltage.

Example 6.7 *A microcontroller with no internal POR logic requires an asserted-low reset pulse of at least 0.1 ms, applied to pin* $\overline{RST}$ *to accept a valid POR signal. Provide a suitable circuit to satisfy the chip requirements. Assume the MCU operates from a 3.3VDC power supply.*

Solution: *In this case, a solution could be provided with the circuit in Fig. 6.25. Here, we essentially need to determine the values of R and C to satisfy this requirement. In this case we will use a 74HC14 inverter, which can be readily operated at 3.3 V and features* $V_T^+ = 1.82\,V$. *From Eq. (6.9), we readily obtain*

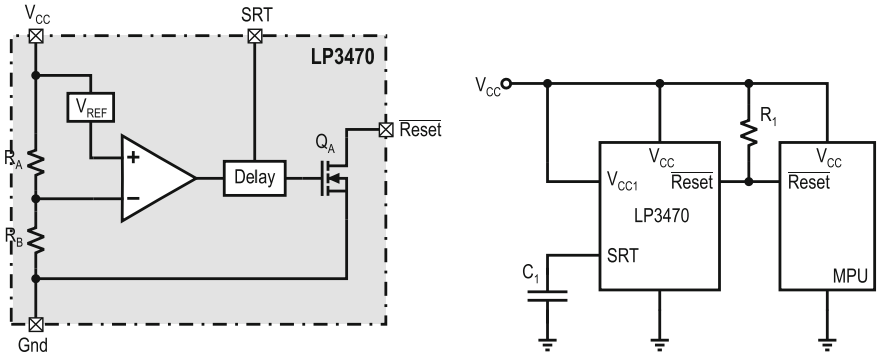

Fig. 6.27 Structure and typical application of LP3470 (*Courtesy of Texas Instruments, Inc.*)

$$t_{reset} = RC \cdot \ln\left(\frac{3.3\,V}{3.3\,V - 1.82\,V}\right)$$
$$t_{reset} = 0.80RC$$

Using the specification of $t_{reset} \geq 100\,\mu s$ *and assigning* $C = 0.1\,\mu F$ *(10 % tolerance) yields* $R = 1.24\,K\Omega$*. In this case, we can choose a standard resistor of* $1.5\,K\Omega$*, 5 %, which would produce an actual reset pulse width of 0.120 ms. This is a comfortable value that covers the component tolerance and still ensures a pulse wider than the minimum required.*

Reset Supervisory Circuits

General purpose processors rarely provide on-chip POR circuitry. This function is typically served by an external, dedicated POR generator circuit. In most microprocessors the POR signal is served by a dedicated chip that also serves as power monitor and provides other supervisory functions that change from one processor to another.

An example of such a circuit is National Semiconductor's LP3470. This power reset supervisor has the capability of providing an externally programmable reset timeout period using an external capacitor. It also monitors the V_{DD} line for undervoltages, triggering the reset signal if the supply level falls below a established threshold value. Figure 6.27 shows a functional block diagram and typical application of this circuit.

For additional details on how to configure the length of the reset pulse and the supply voltage threshold to trigger a POR, the reader is referred to the chip data sheets [40].

6.6 Bootstrap Sequence

The bootstrap sequence, frequently shorthanded as the *Boot Sequence* refers to the sequence of instructions that a CPU executes upon reset. What does it have to deal with boots and straps? The term bootstrap seems to be a metaphor of the phrase "pull oneself up by one's bootstraps" denoting those with the ability of helping themselves without external means. In a sense, that is what the bootstrap sequence does for a computing system: starting it when no other software is yet running. No basic microprocessor or MCU interface is complete without a properly designed boot sequence.

The boot sequence, when compared to the MCU interface elements described earlier in this chapter, can be perceived as an abstract component. After all, it refers to a piece of software. This software, however, requires a physical memory where it can reside and be accessed by the CPU. Thus, in this discussion the ability to provide a booting sequence presumes the existence in the system of a non-volatile program memory where such a code is stored and accessible to the CPU. Note however that we do not call the program memory an element of the basic interface. Just having memory in place does not imply that there is a valid boot sequence in the system. At this point, and for the rest of our discussion, we will assume that such a memory exists in the system, mapped in the right place within the processor's memory map. Section 3.5 provides a discussion on memory.

Specific tasks performed in an MCU boot sequence are influenced by multiple factors that include: specific processor used, system architecture, application, and even the programming style. However, it is possible to identify typical operations taking place in this sequence. These include:

- Identifying and initializing system peripherals. This includes configuring I/O ports, initializing timers, setting-up communication interfaces, etc.
- Setting-up the stack. In order to be usable, a stack needs to be initialized by setting the stack pointer register to point to the top of the memory area designated for stack operations.
- Initializing system-wide variables in memory. Many tasks in embedded software require global variables initialized to specific values for proper software operation. This is a task frequently handled in the boot sequence.
- Performing diagnostics and system integrity check-up. All systems at startup need to verify the integrity of its components to provide some degree of reliability. The boot sequence is the preferred place to run self-tests, particularly on system critical components.
- Loading an operating system or other type program. The boot sequence in these cases is also called the *boot loader*. In some systems this would load the kernel and other components of an operating system or the system run-time environment. In MCUs, for example, with the aid of a JTAG interface, user programs get uploaded into the MCU non-volatile memory.

6.6.1 Boot Sequence Operation

In a computer system the boot sequence begins to be executed when a valid reset signal is accepted by the CPU. Many processors also allow for critical system events other than a power-up detection to trigger the reset sequence as well. Examples include low supply voltage conditions, bus error faults, system integrity violations, and memory access faults, among others.

Another mechanism frequently provided in many systems is the ability to invoke software calls to the reset sequence from within programs. Regardless of the specific trigger, the execution of the boot sequence causes the CPU's control unit to load into the program counter (PC) the address where the first instruction resides. This address is called the reset address. Note that the advantage of starting from a known location in program memory is the main characteristic that makes the boot sequence so important for a computer system. It makes possible not only to start the system upon power-up, but also to recover the status of the system upon a runaway situation.

6.6.2 Reset Address Identification

There are two major modalities in which the reset address can be provided in MCUs: as part of a jump table or as a fixed vector also called *auto-vector*.

Systems using a jump table have a hardwired memory location where the CPU begins execution upon reset. A microcontroller using this type of scheme, upon a valid reset, loads the program counter (PC) with this physical address, making possible to begin the fetch-decode-execution cycle right from the reset location. This type of scheme would also apply to all system interrupts in the same way: a fixed address where execution begins when a qualified events triggers them.

The initial addresses for the reset code as well as for other system interrupts are located in a particular place in memory, each separated from the other by only a few memory words. Typical separations range from four to six bytes away depending on the device. This amount of memory is, in the best of the cases, enough for storing only one or two instructions, insufficient to accommodate a long list of tasks such as those we have identified in a typical reset sequence.

A way to handle this limitation is by placing at the reset address just an unconditional jump instruction that takes program control to the rest of the code. A similar arrangement is made for all other interrupts served by the system. When the entries for reset and the other events are provided, the memory locations for these events become populated with only unconditional jump instructions to the corresponding service routines. For this reason this memory area is called a jump table. MCUs like the classical MCS8051 from Intel and many of its derivatives use a jump table to serve reset and other interrupts. The pseudocode sequence below illustrates how a jump table-based reset address can be configured.

```
;=======================================================================
; Addresses RESET, EVENT1_ISR, etc. are declared in header file
;-----------------------------------------------------------------------
#include "headers.h"
;-----------------------------------------------------------------------
StartUp:    Instruction 1      ; Startup code begins here
            Instruction 2
            ...
Main:       Instruction a      ; Main program (infinite loop)
            Instruction b
            ...
            do_forever
;-----------------------------------------------------------------------
EVENT1_ISR: ---                ; Service routine for Event 1
            ...
            IRET
;-----------------------------------------------------------------------
; Jump table begins below
;-----------------------------------------------------------------------
            ORG RESET_ADDRESS  ; Aligns linker to reset address
            JMP StartUp        ; Unconditional jump to reset code
            ORG INT_EVENT1     ; Aligns linker to event1 address
            JMP EVENT1_ISR     ; Unconditional jump to Event 1 ISR
            ---                ; Other jump table entries
            ...
            END
;=======================================================================
```

Fixed vector systems have a hardwired memory location containing an address pointer to the location where the CPU is to begin execution upon reset. This entry is also called an *auto-vectored* entry. Microcontrollers using an auto-vectored scheme, upon a valid reset, load the PC with the contents of the fixed address location, defining in this way where the instruction fetch-decode-execute cycle must begin. Systems handling a power-up reset through this scheme, also handle interrupt servicing in the same manner. When all vectors are defined, the memory area where they are stored becomes a *fixed vector table*. This is a convenient mechanism for locating the reset code since no jump instruction needs to be executed.

MSP430 devices use an auto-vectored mechanism to support reset and interrupt events. The demo program below by A. Dannenberg illustrates a code organization to support a reset event in an MSP430 device [41].[1] The program toggles I/O port P1.0, producing a square wave signal that could be used to toggle on and off an external device like an LED or a buzzer. The waveform period is controlled with the value loaded into R15. Observe the declaration of the reset auto-vectored entry and the label identifying the startup code.

```
;=======================================================================
;MSP430F22x4 Toggle P1.0 Demo by A. Dannenberg. - 04/2006
;Copyright (c) Texas Instruments, Inc.
;-----------------------------------------------------------------------
```

[1] See Appendix E.1 for terms of use.

```
#include "msp430x22x4.h"                    ; Header file
;-------------------------------------------------------------------------
            RSEG    CSTACK                  ; Stack declaration
            RSEG    CODE                    ; Code into flash memory
;-------------------------------------------------------------------------
STARTUP     mov.w   #SFE(CSTACK),SP         ; Initialize stack pointer
            mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer
            bis.b   #001h,&P1DIR            ; Configure P1.0 as output

Main        xor.b   #001h,&P1OUT            ; Toggle P1.0
            mov.w   #050000,R15             ; Use R15 as delay counter
Repeat      dec.w   R15                     ; Decrement count
            jnz     Repeat                  ; While not zero repeat
            jmp     Main                    ; Do it again
;-------------------------------------------------------------------------
            COMMON  INTVEC                  ; Auto-vector table
;-------------------------------------------------------------------------
            ORG     RESET_VECTOR            ; Power-on reset auto-vector
            DW      STARTUP
            END
;=========================================================================
```

The invocation to the reset sequence is very similar to that of processing an interrupt, except that there is no return address involved: the PC is loaded with the reset address and the status register is cleared.

Microprocessor-based systems in their vast majority handle interrupt events through a vectored system. In such a system, all interrupt events are issued through a single interrupt request pin. An interrupt acknowledgment signal from the interrupting device issues a pointer to an entry in a memory held vector table whose entries are the addresses of the service routines for the supported events. Even in these system it is common to find that a reset event is auto-vectored, requiring no interrupt acknowledgment signal. On example of such an arrangement is found in Intel 80x86 and Pentium devices. A more detailed discussion of vectored methods in microprocessor system is included in Sect. 7.1.

6.7 Reset Handling in MSP430 Devices

The MSP430 provides a flexible on-chip system initialization hardware in all its devices. This means that the required power-on-reset signal necessary to initialize the system can be internally generated without the need of external components. The internal reset circuitry also supports an externally triggered reset signal ($\overline{RST}$) through external pin $\overline{RST}/NMI$. As implied by its name, this pin is shared with the non-maskable interrupt (NMI) input. The default function of the $\overline{RST}/NMI$ pin upon power-up is for accepting an asserted-low reset signal. Although it could nominally be tied to V_{DD} if not being used for either of its functions, adding an external RC (R = 47 K, C = 10 nF) is generally recommended to reduce the chance of false triggering due to power supply noise.

The functionality of $\overline{RST}/NMI$ can be changed via software from its default setting to become an NMI input. All generations of the MSP430, except x5xx/x6xx devices, contain in the Watchdog Timer Control Register (WDTCTL) the bits for configuring this pin. In series x5xx/x6xx the functionality is moved to a Special Function Register (SFR).

The reset circuitry in all MSP430 devices has a fundamental structure that provides for two internal reset signals: Power-on Reset (POR) and Power-up Clear (PUC). Figure 6.28 shows a simplified block diagram of the base reset structure.

POR is the MCU reset signal. This signal is generated when the device detects a power-up or brownout event in the main voltage supply line (V_{CC}) or when the external $\overline{RST}/NMI$ line, in its reset mode, is held low for at least t_{reset} time. The minimum length of t_{reset} is typically 2 μs for most family members, although the specification is left as a device dependent parameter. Another system event capable of triggering a POR in MSP430 devices equipped with a Supply Voltage Supervisor (SVS) is a low-voltage detection in the analog supply voltage line (AV_{CC}) or in a selected external voltage. An SVS circuit is a handy addition to enable monitoring subtle voltage variations in the supply line that could escape the brownout detection circuit and compromise the correct device functionality.

The PUC signal is like a soft reset for the device in the sense that it can be configured to be triggered by additional system events than those triggering a POR. A PUC is always triggered when a POR is triggered, but not viceversa. Conditions triggering a PUC include the expiration of the watchdog timer, a password violation when modifying the WDT control register or FLASH memory, and an invalid access caused by fetching instructions outside the valid range of program memory addresses (bus error).

6.7.1 Brownout Reset (BOR)

Every MSP430 device features a brownout reset circuit that triggers a POR signal when power is applied or removed from the supply voltage pin or when voltage fluctuations capable of compromising the integrity of internal registers or memory

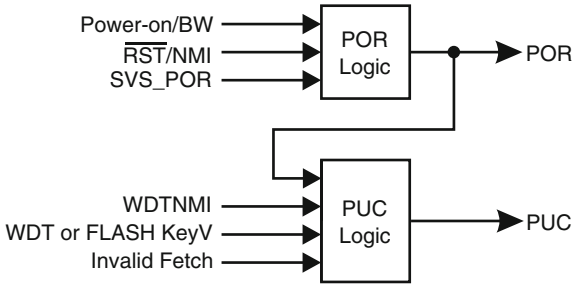

Fig. 6.28 Basic power-on reset module in MSP430 MCUs

occur. The conditions for triggering a BOR event require the supply level V_{CC} to cross the $V_{CC(start)}$ level. The BOR event remains active until V_{CC} crosses the $V_{(B\_IT+)}$ threshold and lasts for at least $t_{(BOR)}$. The length $t_{(BOR)}$ is adaptively determined by the slope of the supply voltage fluctuation, being longer for a slow ramping V_{CC} changes. Levels $V_{(B\_IT+)}$ and $V_{(B\_IT+)}$ define dual hysteresis thresholds that prevent successive BOR events unless the supply voltage drops below $V_{(B\_IT)}$. It deserves to mention that the level of $V_{(B_{IT})}$ is well above the minimum voltage level necessary to activate the POR circuit. Thus, it reacts earlier than the POR circuit upon power failures. Figure 6.29 shows the voltage levels leading to the triggering of a BOR event.

6.7.2 Initial Conditions

After a POR, the initial conditions in an MSP430 device are the following:

- The functionality of the $\overline{RST}/NMI$ pin is set to $\overline{RST}$.
- GPIO pins in all ports are configured as inputs.
- The processor status register (SR) is loaded with the reset value, which clears the V, N, Z, and C flags, disables all maskable interrupts (GIE = 0), and the CPU is set to active mode, cancelling any low-power mode previously set.
- The watchdog timer is activated in watchdog mode.

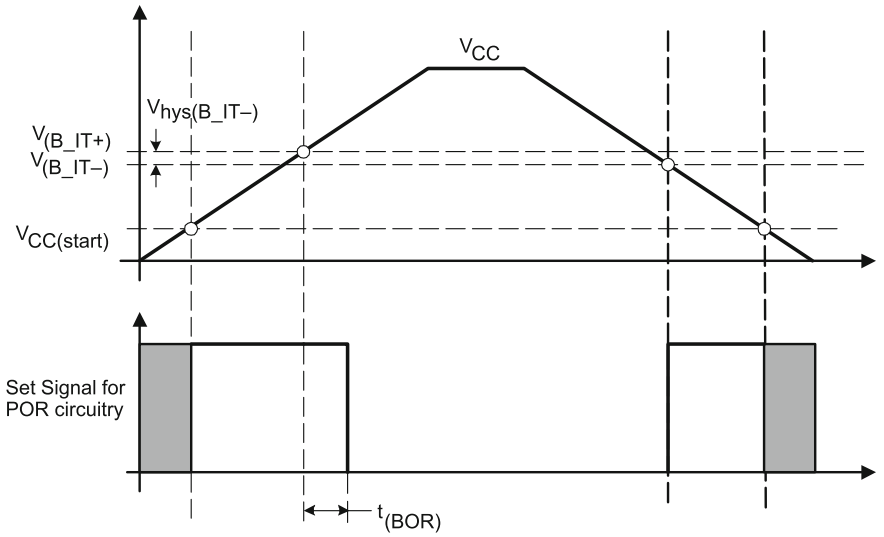

Fig. 6.29 Voltage levels and timing of an MSP430 BOR event (*Courtesy of Texas Instruments, Inc.*)

- The program counter (PC) is loaded with the address pointed by the reset vector (0FFFEh). Setting the reset vector contents to 0FFFFh, disables the device, entering a low power mode.
- Particular peripheral modules and registers are also affected, depending on the specific device being used.

Considering the particular conditions induced by the POR sequence, there are specific actions that must be performed by any user software running in the MSP430 upon reset. These include:

- Initializing the stack pointer (SP) to the top of the designated area in RAM.
- Deactivating or reconfiguring the watchdog timer to the mode required by the application.
- Configuring the peripheral modules necessary to support the particular application.
- Configuring desired low-power modes.
- Optionally, to cope with potential threatening situation, examining particular flags in the watchdog timer, oscillator fault, and/or flash memory status registers to ascertain the cause of the reset.

6.7.3 Reset Circuitry Evolution in MSP430 Devices

The fundamental MSP430 reset structure has exhibited several improvements as its design has evolved through the different device generations from series x3xx through x6xx.

Early MSP430x3xx had a basic system reset circuit with power-up/brownout detection, external RST/NMI support, and internal detection of watchdog timer expiration and key violations. POR could be triggered by a power-up/brownout or by the reset pin $\overline{RST}$ being driven low. PUC could be triggered by a POR or by a watchdog timer expiration or a WDT key violation.

As a measure to prevent accidental modifications to the on-chip flash memory, the MSP430 requires that any read or write access to the on-chip flash controller be word-size, containing in the upper byte the value “0A5h” as keyword. Failure to do so triggers a security key violation condition, denoted by the KEYV flag. This event triggers a PUC system reset. Accesses to the WDT control register are similarly protected. The failure to include the 0A5h keyword will cause a WDT key violation. Additional details on the operation of both the WDT and the on-chip flash memory are discussed in Chap. 7.

Series x4xx shared the same reset functionality as its predecessor, with the added functionality of including FLASH memory key violations among the sources for triggering a PUC.

The reset structure of MSP430 series x1xx and x2xx devices inherit the characteristics of x4xx devices plus a few additions. Series x1xx introduces the ability of configuring the SVS to trigger a POR. This improves device reliability in terms of a better ability to detect a wider range of potentially harmful power conditions.

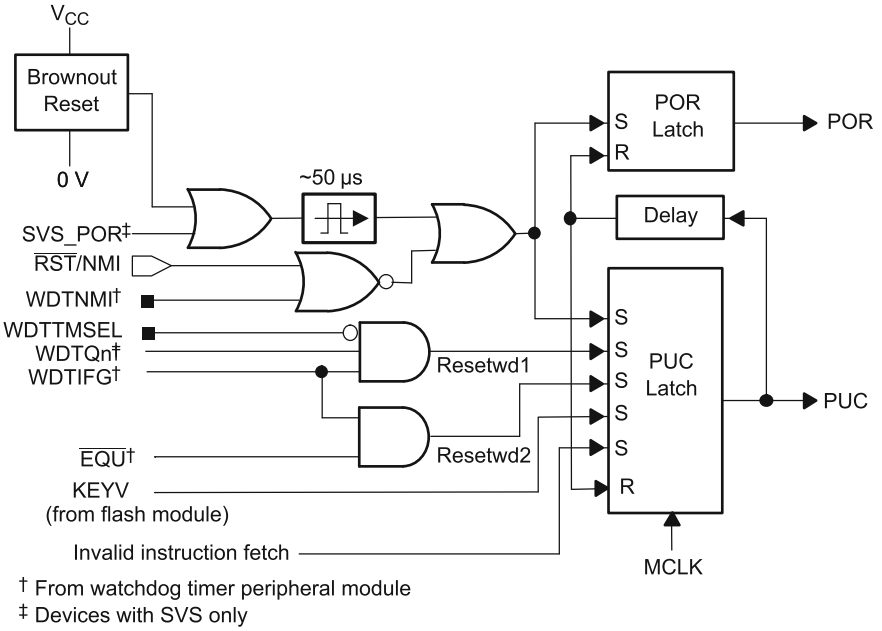

Fig. 6.30 Circuit schematic of POR and PUC logic in MSP430x2xx devices (*Courtesy of Texas Instruments, Inc.*)

The introduction of x2xx devices adds the ability to trigger a PUC by an invalid instruction fetch. Figure 6.30 shows a schematic of the logic governing the behavior of signals POR and PUC in MSP430x2xx devices.

The latest generations of MSP430 devices, series x5xx/x6xx feature an expanded reset and initialization circuit as part of the System Control Module (SYS). The reset logic now provides three internal signals: brownout reset (BOR), power-on reset (POR), and power-up clear (PUC). The BOR signal is separated from the POR to denote the expanded nature of a POR event in the MCU architecture.

The BOR signal is now the device reset, generated by either a power-up event/brownout condition, a low pulse in the $\overline{RST}/NMI$ pin when in reset mode, a wake-up from the low-power modes LPM3.5 or LPM4.5, or a software triggered BOR event. Note that this generation of devices now supports specific software triggered BOR events. From the list of events triggering a BOR, only a power-up/brownout event cannot be masked through software-controlled enable flags. This gives an absolute highest priority to this event.

The POR signal now acquires a softer functionality, since all sources triggering it (excluding a BOR) can be software masked. A POR is triggered whenever a BOR goes off (not the opposite). In addition, voltage levels below the system voltage supervisor high and low levels can be enabled to trigger a POR, as well as a software triggered POR event.

In the MSP430x5xx/x6xx reset chain, PUC is the last signal link, being triggered whenever a POR is generated. As in previous generations, a PUC is also triggered by watchdog timer expiration or key violation events, by Flash memory violations, and an invalid fetch. In this new generation a PUC is also triggered by a password violation in the power management module.

6.8 Summary

This chapter provided concepts and applications of developing interfaces for microcontroller-based systems, from the basic interface necessary to start-up an MCU to specific interfaces that allow to integrate user interfaces, large loads, and motors. Section 6.1 introduced the fundamental components that make an MCU work: Power, clock, reset, and bootstrapping.

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. was 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.

6.9 Problems

6.1 Enumerate the elements of a basic microcontroller interface and indicate two major criteria for the selection of each.

6.2 Why should a designer ensure that an MCU in an embedded application does not work at its absolute maximum ratings?

6.3 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 a non-regulated power supply and a suitable regulator.

6.4 Assume the system described in Problem 6.1 is to be fed from a 4.0 V, 2,400 mAh lithium battery. Estimate the expected battery life, assuming LEDs are driven dynamically at 30 % duty cycle. What would be the regulator effi-

ciency? Determine the MCU thermal dissipation for the given loading conditions and verify if it is operating within a safe temperature range.

6.5 A crystal oscillator with a nominal frequency of 7.3728 MHz exhibits an aging induced drift of 12 PPM per year. If the system where it is being used has an accuracy requirement of 5 %, after how long shall the crystal be replaced to maintain compliance with frequency accuracy requirements?

6.6 Perform a comparative analysis of the advantages and disadvantages of using Colpitts versus Pierce oscillators in microcontroller-based systems. Identify at least one instance where one type would be more convenient than the other.

6.7 An MSP430F5438 is driven by an external crystal oscillator of 32.768 KHz. Devise an initialization sequence to produce a frequency of 11.0592 MHz in the ACLK line to drive a communication channel. Use the external crystal to stabilize the VLO generated frequency.

6.8 A certain microcontroller has an asserted-low Reset input that requires at least 25 clock cycles of assertion to be accepted as a valid reset input. Using the basic POR circuit illustrated in Fig. 6.25, determine suitable values of R and C, considering tolerances of 5 % in each to assure the circuit will always provide a valid reset when the MCU is running at 8 MHz.

6.9 An MSP430F2274 is operated from a 3.3 V power supply. What should be the power supply stability requirements to prevent that a normal voltage fluctuation in the supply voltage might trigger a brownout reset in the device? What would be the lowest safe voltage level at which the chip might operate?

6.10 Provide an initialization sequence for the MCU in Problem 6.1, 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.

6.11 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.

6.12 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.

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

6.14 Design a dumb HEX terminal using a 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.

6.15 A 120VAC fan is to be controlled by an MSP430F2273. Assuming the maximum fan current is 2 A, 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.

6.16 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 1 mA when pulsed, and the motor consumes anywhere between 8 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°.

6.17 A 12VDC, 1.5 A 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.

6.18 Provide an enhanced interface for the motor in Problem 6.10 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.