Driving Innovation in 32-bit Embedded Designs with Standard MCU Peripherals

Today's 32-bit microcontroller (MCU) designs integrate a wide variety of standard peripherals, such as analog-to-digital converters (ADCs), digital-to-analog converters (DACs), universal synchronous/asynchronous transmitters (USARTs) and timers. While this peripheral integration is intended to address a wide range of general-purpose applications, embedded developers commonly consider these on-chip peripherals to be "check box" items, and the implementation of this peripheral integration usually ends up providing limited value. In addition, the lack of smart interconnectivity among peripherals often forces designers to add significant amounts of glue logic to ensure proper use of the MCU's features, resulting in unnecessary incremental development costs and system complexity.

Standard microcontroller peripherals are intended to support a broad range of general-purpose embedded applications. However, peripherals often are integrated into 32-bit microcontroller designs with little or no thought about how they can enable a more cost-effective or simpler application implementation. Let's take a closer look at implementations of standard microcontroller peripherals that provide system-level value and application benefits that help designers get the most out of every feature in the microcontroller while simultaneously reducing the bill of materials (BOM) and simplifying the embedded design. Innovative implementations of standard microcontroller peripherals, such as I/O with deterministic pulse generation and level shifters, USART input noise-filtering for reliable communications, capacitive-sense tamper detection systems, current-based communication using interconnecting digital-to-analog converters (IDACs) and synchronous current-to-voltage converter systems, can provide significant value for 32-bit embedded designs.

I/O with Deterministic Pulse Generation and Level Shifters

Some 32-bit applications require the generation of one or more pulses with precisely-controlled pulse widths. Examples include a trigger for an external event or a test pulse to stimulate an external device within a specific time window. A typical software-implemented I/O toggle cannot guarantee the pulse width since the software is subject to interruptions and latencies, and using a timer consumes a valuable and typically pin-constrained resource.

An innovative solution to this design challenge implements pulse generation at the I/O level. This system provides the capability to set the initial and final value of the I/O in one 32-bit register (allowing the initial and final value to be set in a single 32-bit word write) along with a mask register to enable multiple pins at any time and a programmable 5-bit delay time. Once initiated, the pulse generator waits for a number of peripheral clock cycles set by the delay time before driving the pre-programmed final value onto the I/O pin, enabling simple yet effective deterministic pulse generation. Figure 1 shows a practical implementation of this feature at the register level.

PBCFG0_CONTROL0: Global Port Control

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name	PGDONEF	Reserve		PGTIMER					Reserve							
Type	R	R		RW					R							
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	INT1EN	Reserve	INT1MD	INT1POL	INT1SEL				INT0EN	Reserved	INT0MD	INT0POL	INT0SEL			
Type	RW	RW	RW	RW	RW				RW	RW	RW	RW	RW			
Reset	0	0	0	0	0				0	0	0	0	0			

PGTIMER holds the time delay between the pin transitions. PGDONEF indicates when the time has expired.

PBSTDn_PBPGEN: Pulse Generator Pin Enable

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name	Reserved															
Type	RW															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	PBPGEN															
Type	RW															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PBPGEN enables the pulse generator in the corresponding port bank pins. Multiple I/Os can provide pulse generation functionality if required.

PBSTDn_PBPGPHASE: Pulse Generator Phase

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name	PBPGPH1															
Type	RW															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	PBPGPH0															
Type	RW															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PBPGPH1 holds the logic level for phase 1. PBPGPH0 holds the logic level for phase 0, when pulse generation enabling writing to this register will trigger the beginning of the pulse. The value in PBPGPH0 will be applied to enable pins immediately, and when the pulse generator counter (PGTIMER) times out, the enable pins will be set to the value in PBPGPH1.

Figure 1. Pulse Generation Register Implementation

3 V-to-5 V Level Shifting

Another common challenge in factory automation and industrial control applications is interfacing with legacy 5 V I/O systems. While a 5 V-tolerant I/O provides a means of receiving input from 5 V outputs, bidirectional communication requires the capability to drive 5 V logic levels. To address this problem, system designers are often forced to add external glue logic, such as external level shifters, increasing PCB layout complexity and BOM cost.

An integrated, versatile level shifting system can be achieved by enabling a 3 V I/O and a 5 V drive I/O that are directly interconnected on the same device, thus enabling a 3 to 5 V level shifting capability. This connectivity enables the developer to connect any output peripheral that does not already go to the 5 V drivers through an external wire that can be converted to a 5 V driver.

Figure 2 provides an example in which a timer output that is typically in a 3 V domain can be level shifted by externally connecting the timer output to one input of the 3 V I/Os that are internally connected with the 5 V drive. The output of the corresponding 5 V drive output provides a 5 V timer output.

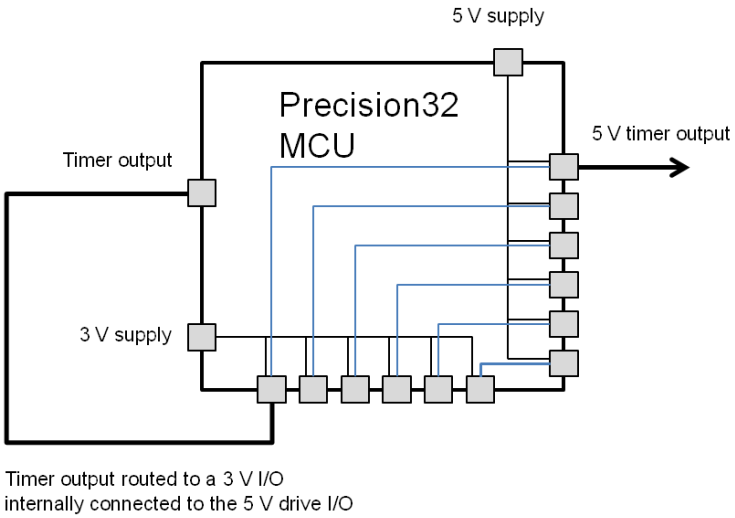

Figure 2. Internal Level Shifter System

Current-Based Communications Using IDACs

Applications developed to support noisy environments, such as factory floors, face the major challenge of providing reliable communications in the presence of high noise levels. Typical communications interfaces, such as USART, SPI and I2C, have proven to offer some level of reliability. However, these applications are always pushed to their limits. One solution is to implement current-based communications using IDACs to enable communications protocols with high levels of noise immunity.

To implement this solution, the developer must use a current-mode IDAC. While a typical DAC provides a voltage output through a current output, an IDAC is controlled by a smart timer that can assist in generating the timing sequence for the communications protocol. In addition, a FIFO buffer on the IDAC can generate an interrupt after the FIFO has sent the last word of data, allowing the software to load the FIFO with the next packet of data to be sent, as shown in Figure 3.

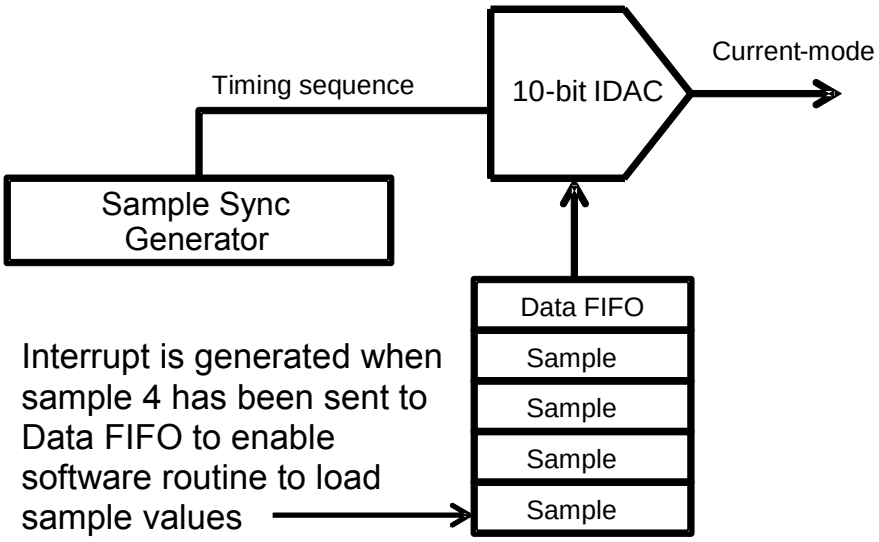

Figure 3. Current-Based Communications Using IDACs

Input Filter for Noisy Communications

In a typical industrial context, it is relatively common to find communication channels affected by noise. For example, a USART is a typical interface that can be impacted by a noisy environment and thus benefits from a comparator threshold filter used to filter noise from the receiver (RX) path.

To implement this system, it is necessary to use a comparator with an N-bit DAC acting as a voltage reference on the negative input. If the device containing this comparator is used in a noisy environment and it has to detect a noisy USART RX signal, the comparator can be used as an RX filter to remove the noise and avoid bad USART data. The USART RX signal coming from the slave device will connect to the selected comparator positive input pin. The N-bit DAC can be used to threshold filter the RX signal, given prior knowledge of the noise in the system. The comparator's asynchronous output is then sent from the device back into the actual USART RX input.

This input filter technique is not limited to a USART RX channel. The same technique can be used on any noisy input signal. Figure 4 provides a flow diagram of the signal path to enable a filter mechanism for a UART implementation.

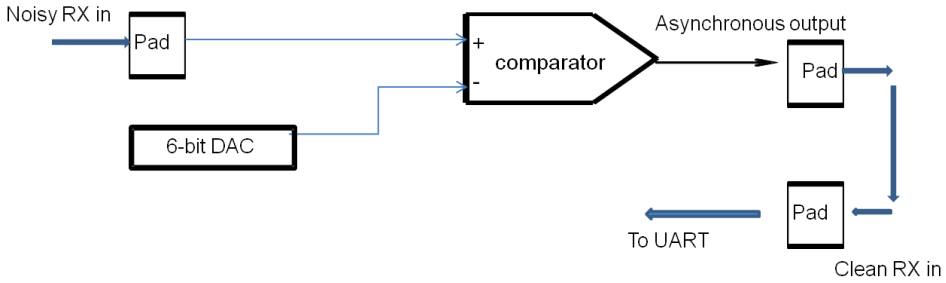

Figure 4. Comparator-as-Filter USART Receiver

Capacitive Touch for Tamper Detection

Many 32-bit microcontroller-based systems used in industrial control or home automation require some level of tamper detection. For example, security cameras use detection of a known number of consecutive dark images as a tamper detection trigger. Other techniques might use external sensors to generate a tamper trigger event. Most 32-bit MCUs have fixed input pins that are either active high or active low and float or switch polarity to indicate a tamper event.

An alternative solution uses a capacitive touch-sense engine as a potential multi-channel tamper detection system. For example, the capacitive touch-sense engine built into Silicon Labs' ARM-based Precision32® MCUs supports channel bonding and has a maximum of 16 possible input channels supporting up to 16 tamper detection channels in a single conversion. Alternately, using a non-channel-bonded mode combined with a programmable threshold, 16 tamper detection channels with independent tamper thresholds on each channel can be implemented with minimal software overhead. This engine can be used to generate a time stamp via an interrupt to the system generating a real-time clock (RTC) trigger event.

A typical untampered system can be implemented with tamper detect channels that have a known capacitance, which will be detected. For example, if the capacitive touch-sense node is connected to a closed door monitored by a security system, you would expect the capacitance to be "X" pico farads (pf). If the door is opened, the capacitance will be "less than X" pf. In this scenario, the capacitive touch-sense module can be configured to generate an interrupt that then triggers the system to time-stamp the event in software by reading the RTC timer register at this time. There will be a few cycles of latency, but given the frequency of the RTC and the operating frequency of the system, the human scale timing of this event

will, for all intents and purposes, look like a zero-time event. The capacitive touch-sense module also supports a “greater than threshold” event trigger, so the opposite of the “closed door” system can also be implemented (i.e., an “open door” system). Given the multi-sampling capability of the capacitive touch-sense module, multiple samples can be taken and accumulated to filter out false triggers.

The capacitive touch-sense capability can also function in sleep mode to enable a low-power system. This can be accomplished by setting up the RTC to trigger a certain number of times per second, wake up the system and perform a capacitive touch-sense port scan. Depending on whether a tamper event was detected or not, the system will either continue with post-tamper-related functions or go back to sleep.

Synchronous Current to Voltage Converter Systems

Many sensors used in factory automation, building control and medical applications, such as pressure sensors, optical sensors, transducers and biosensors, provide a current output instead of a voltage output to achieve greater noise immunity. To properly interface with current output sensors, the design must include a current-to-voltage converter. Typically, an external current sensing system is added to the system, resulting in greater BOM cost, design complexity and challenges in synchronizing measurements based on dynamic excitation references. For example, a biosensor used in a medical glucose meter, outputs tiny current values as a result of a dynamic excitation. A cost-effective implementation of a biosensor interface that requires full synchronization between dynamic excitation and current measurements can be achieved using a synchronized DAC-ADC mechanism. Precision32 MCUs, for example, support this synchronization capability. In addition to providing a full synchronization path between the ADC and DAC, Precision32 MCUs also feature an internal current-to-voltage converter.

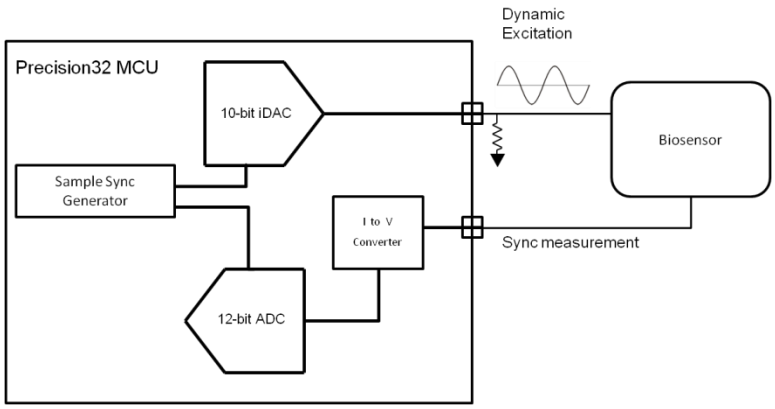

Figure 5. DAC and ADC Synchronization Based on Current-to-Voltage Converter

Figure 5 illustrates the DAC-ADC synchronization mechanism using the current-to-voltage converter. The sample synchronization generator module can generate a synchronous pulse to trigger both the DAC and ADC conversion inputs. The ADC can be configured to either sample synchronously with the DAC output or in a delayed fashion by a fixed number of clock cycles to allow the sensor to process the stimulus input. This enables a fully synchronous measurement system, simplifying the system design and reducing cost by eliminating external components.

Conclusion

Innovative implementation of standard 32-bit microcontroller peripherals coupled with a highly flexible interconnect architecture enables a wide variety of configurations for embedded systems. The synchronous current-to-voltage converter, input filter for noisy communication and 3 to 5 V level shifting

are examples of flexible interconnect architectures. Using versatile I/Os and peripherals, such as capacitive touch, I/O pulse generation and DAC precision enhancement, developers can achieve novel solutions to real-world embedded design problems while reducing cost, simplifying design and accelerating time to market.

# # #