# Why PSoC is Unique

## Introduction

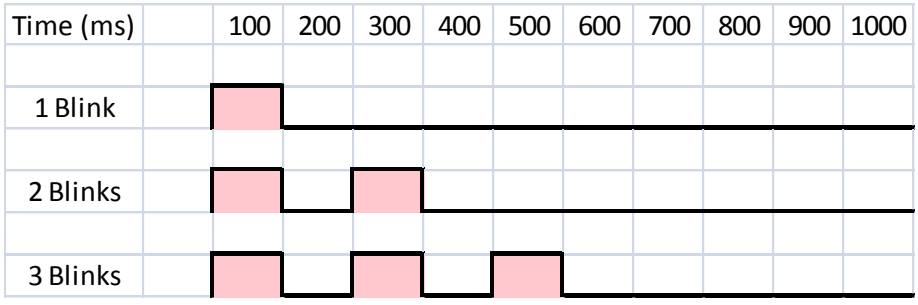
PSoC is like other microcontrollers in many respects –PSoCs have ARM core processors, fixed function peripherals such as I2C, PWMs, Timers, ADCs, Opamps, CapSense, etc. But in other ways, PSoC is different.

It is often these differences that make PSoC most valuable to our customers. However, some of the unique features and their advantages may not be immediately obvious. A few of the ways that PSoC's capabilities have proven valuable in the real world are described below.

## Offloading the CPU

The CPUs in microcontrollers are constantly being asked to perform more and more tasks. This can lead to increased firmware complexity as well as difficulty in managing latency. With PSoC, it is common to use the Universal Digital Blocks (UDBs) to implement functions in hardware so that the CPU isn't overly burdened. In addition to freeing up the CPU, logic that is implemented in hardware is failsafe (it won't stop working if the firmware ends up off in the weeds) and has constant timing (no delays due to firmware or interrupt latency). This is different from fixed function peripherals in that you can customize the UDBs to do practically anything that you want. In fact, many of the pre-configured PSoC components provided in PSoC Creator are made using UDBs.

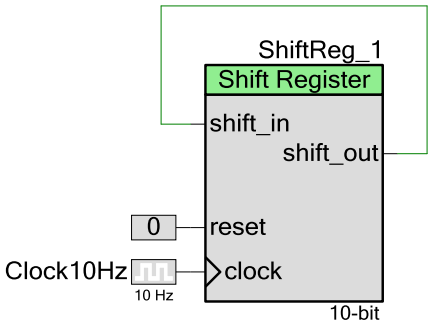
As an example, consider that you want an LED to flash in a pattern that indicates some sort of system status. Assume that the entire cycle is one second and the LED can flash on/off in 100ms intervals anywhere from 1 to 3 times during each cycle. That is, the LED can flash on/off once, twice, or three times to indicate three different states (perhaps 1 blink is normal operation while 2 and 3 blinks indicate two different fail states). The 1 second interval shown in 100ms intervals for each of the three states is:




This could certainly be done in firmware using an interrupt that triggers every 100ms. The ISR would then look at a state variable and decide what to do with the LED. The downside is that while the ISR is taking care of the LED, the CPU can't be doing other (perhaps critical) operations. We have added latency to the system. In addition, if the CPU gets stuck in an error condition, the LED will no longer blink.

As an alternative, we can place a 10-bit shift register in the schematic as shown below. We connect the shift_out pin to the shift_in pin to create a circular buffer and connect the clock pin to a 10Hz clock (100ms period). Now, we just have to load the register with the pattern that we want and the hardware

will take care of the rest. For example, if we load the register with “000000001” we will get one blink per second because we will shift out 1 for the first clock cycle (100ms) and 0 for the next 9 clock cycles (900ms). The pattern will repeat every second since the shift_out connects back to the shift_in which causes a repeat every 10 clock cycles. If we instead load the register with “0000000101” we get two blinks per second (100ms on, 100ms off, 100ms on, 700ms off), and so on. Instead of the CPU having to deal with the LED every 100ms, it just has to load the register when an event causing a state change occurs.




As a more complex example, consider writing to a set of 16 WS2811/12 RGB LED strips to create a billboard that is 16x60 (960 RGB LEDs = 2880 total LEDs). Examples of doing this using traditional microcontrollers can be found online. In those cases, the LED strips are bit-banged using the CPU to drive outputs directly. As you can imagine, writing to 2880 LEDs takes a lot of CPU power and timing for the signals can get tricky.

With PSoC, we create a custom component with an API that allows most of the hard work to be done in the UDB logic. The timing is very simple and the CPU is almost entirely free to do other tasks. The component uses only 2 to 4 UDBs depending on the number of channels required.

A complete example of this example including projects for both PSoC 5LP and PSoC 4 as well as a video of the billboard in action can be found at:

http://www.element14.com/community/message/89756/l/psoc-4-pioneer-kit-community-project100-psoc-4-times-square-led-billboard

## Glue Logic and More

Systems often require logic to interface between the main components. For example, what if a signal needs to be inverted when going from one device to another or it needs to be gated with a second signal? In other cases, there are last minute design changes requiring pinout or logic modifications. PSoC’s UDBs and flexible pins can save both re-work time and board cost.

The UDBs contain both PLD functionality and an 8-bit ALU. With the PLDs you can write Verilog code and have PSoC Creator synthesize the logic for you or you can use the UDB editor to graphically create a state machine. The UDB editor can also be used to configure the ALUs to do a multitude of computational tasks to free up the CPU.

## Voltage Flexibility

Many PSoC devices have multiple power domains. For example, PSoC 5LP has the following six supply voltages:

Analog (Vdda)
Digital (Vddd)
Four separate IO supply banks (Vddio0, Vddio1, Vddio2, and Vddio3).

The analog supply has to be the highest voltage but otherwise any combination is possible.

Some ways in which this is useful:

You may want to have a high analog voltage to provide the range of analog inputs that you require while using a lower voltage for the core logic to save on power consumption. Signals that go from the analog domain to the digital domain and vice versa are voltage level translated automatically.

You may have multiple power domains in your system and you may need devices between domains to communicate. In this case, you can set different IO banks to different voltages and use the PSoC to do the voltage level translation for you. No additional logic is needed. In fact, you can have a signal come in to the PSoC and then go back out on a pin in a different IO bank with no other connection on the chip. This can be done on the schematic in PSoC Creator so that no register writes, firmware, or any interaction with the CPU is necessary.

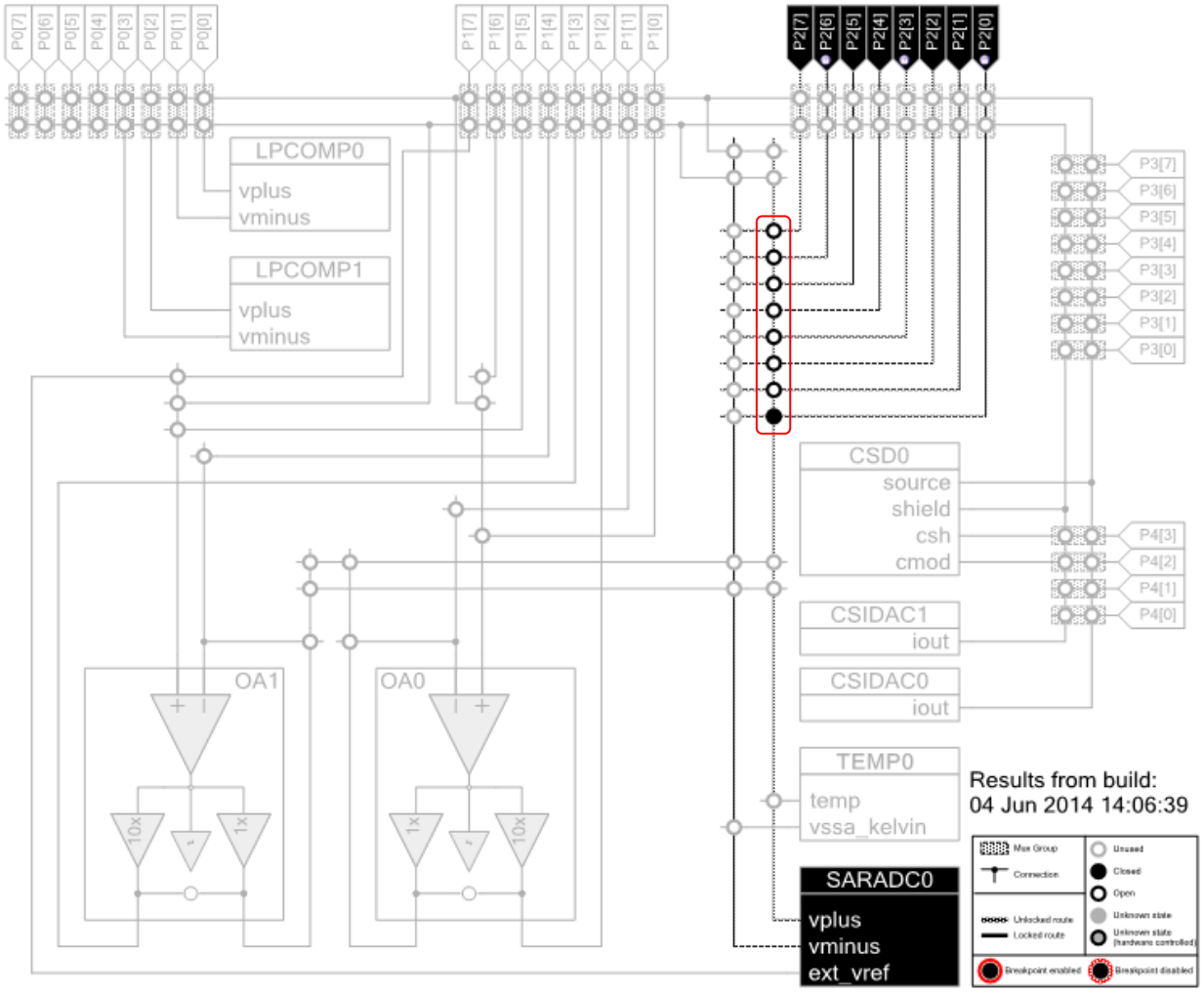
For example, you could have:

Vdda = 5.0V
Vddd = 1.8V
Vddio0 = 1.8V
Vddio1 = 2.5V
Vddio2 = 3.3V
Vddio3 = 5.0V

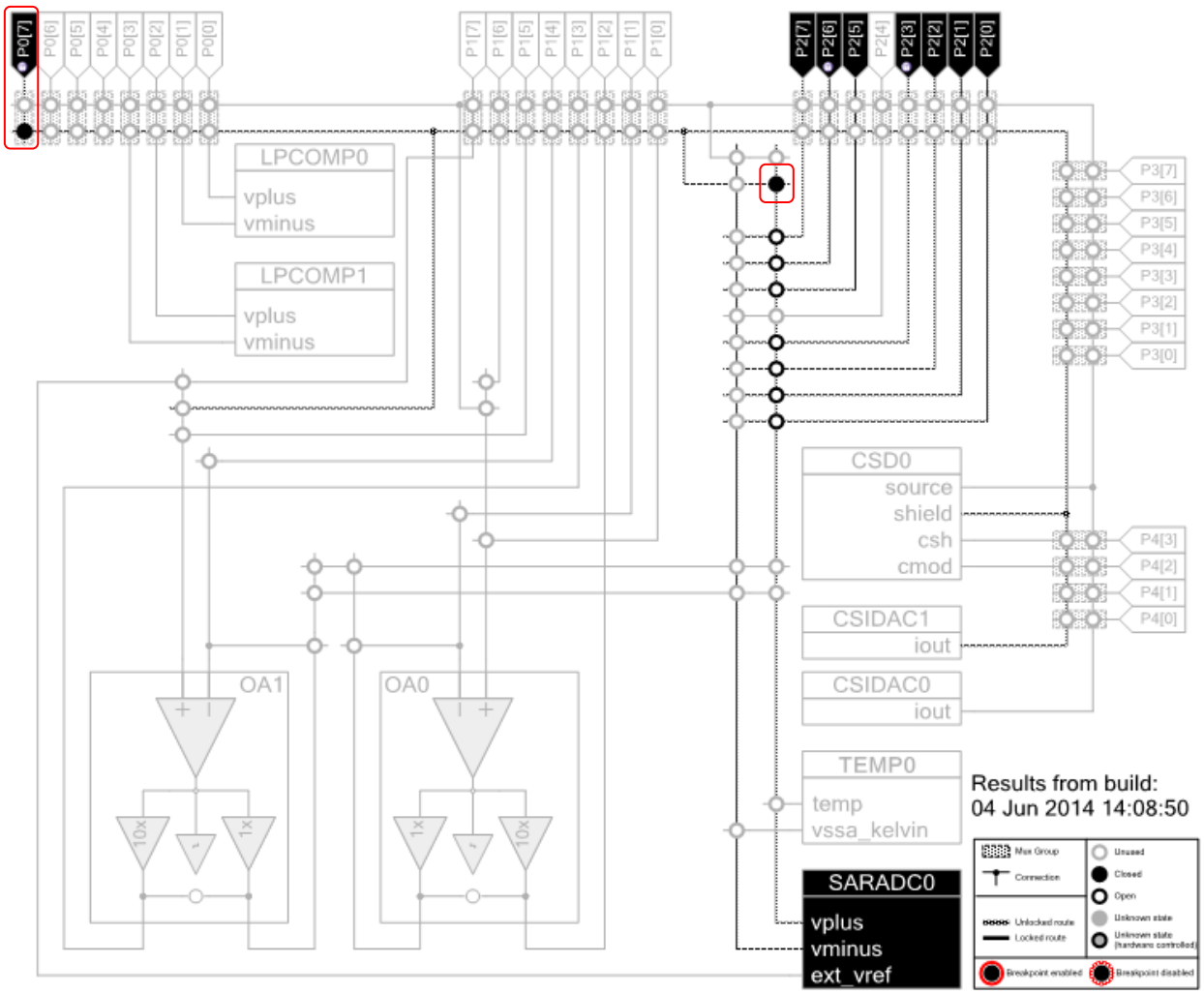
The above combination allows you to seamlessly connect logic from four different supply domains with no discrete voltage translation circuitry. As long as you have PSoC pins available, you can easily translate to/from any voltage domain.

## Pin Flexibility

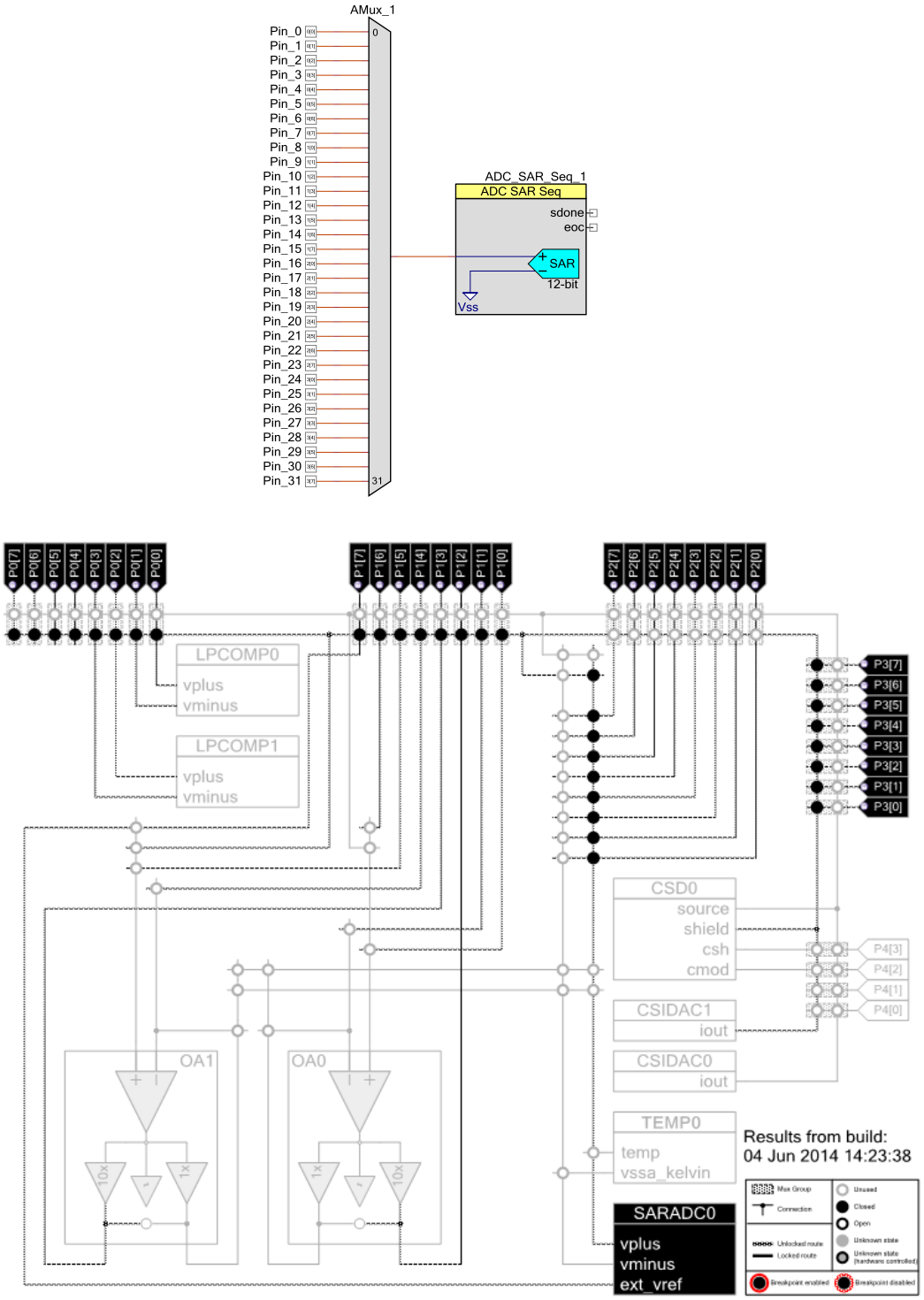
With very few exceptions, all pins on a PSoC can be both analog and digital. As an example, consider the PSoC 4 analog routing diagram shown below. Notice that the SAR has dedicated Mux inputs from eight of the pins (P2[7:0]).




Likewise the sequencing SAR ADC component allows up to eight channels. This may cause you to think that the SAR can only connect to P2[7:0] or that the maximum number of channels that the SAR can sample is 8. However, neither of these are the case. All of the other GPIO pins can connect to the SAR using the analog routing on the device. For example, the figure below shows P0[7] connected to the SAR instead of P2[4].




In fact, in PSoC the only limitation is the number of pins available. You can connect (just about) every pin into a large Mux to the SAR component if that is what your application requires – you are not limited to just 8 pins. As an example, here is a PSoC 4 schematic and analog routing diagram showing a SAR with 32 input channels! The API for the Mux handles turning the appropriate switches on and off to make the connections required for each channel. You don't need to write to registers – just tell the API which channel you want to sample and go.




In addition to pin location flexibility, PSoC pins have:

1. Selectable interrupts (rising edge, falling edge, or both edges)
2. Up to 8 different drive modes
3. Programmable I/O threshold levels

## Components – Ours and Yours

All of the custom logic in PSoC, custom code, APIs, and even customizer GUIs can be encapsulated in components. Components can (and often do) contain other components within them. A wealth of components is built right into PSoC Creator, but it doesn't stop there.

For example, what if you need an I2C interface but you need it to do something slightly different than the standard? You can import the UDB based Cypress I2C component, look at the implementation, modify it to do what you need, and save it as your own custom component.

You can share components with other users and can group them together into custom libraries. In fact, there is a set of "community components" that can be found on www.cypress.com under "Support & Community -> Developer Community -> Community Components".

The Component Author Guide (available from the PSoC Creator Help –> Documentation menu) gives more detailed information. There are also video tutorials online.