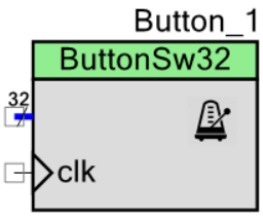
ButtonSw32: Button Switch debouncer
0.0

Features



- Implements software debouncing for button switch.
- Uses vertical counters algorithm.
- Debounces up to 32 buttons simultaneously.
- Does not consume UDB resources.

General description

The ButtonSw32[*] component implements software debouncing using vertical counters algorithm. It can detect button press and release events[†]. Component does not consume hardware resources, and spares very little CPU clocks (~0.01%). The component can simultaneously handle up to 32 inputs without loss of performance. Multiple instances of the component can run simultaneously in the project.

When to use ButtonSw32 component

Component was developed as part of a polyphonic piano project, where multiple low-cost mechanical switches had to be debounced for press and release events. It can be useful whenever a simple button switch event need to be detected by PSoC with high reliability, such as control switches, panel-mounted momentary buttons, mechanical joysticks, etc. Component is useful when PSoC hardware debouncing is not justified for handling simple switch button, or for a system with limited hardware resources, such as PSoC4. Component was tested using CY8KIT-059 prototyping kit (PSoC5LP) and CY8KIT-042 Pioneer Board (PSoC4200). Several demo projects are provided along with the Application Note.

[*] Hereafter referred to as Debouncer
[†] This basic version of the component, which doesn't have many of the advanced options, such as: double-click, long press, key acceleration, random pins assignment, etc.

Input-output connections

inp – button switch terminal

External terminal for connecting to a switch button annotation component (off-chip). The pin is always visible. The pin doesn't have to be connected, it is merely an external terminal to the button annotation component, provided to enhance visibility of the component. Actual assignment of the button pin is performed in the Pins dialog. See **Implementation** section for details.

clk – clock input

Input pins are sampled on rising edge of this signal. The pin is visible when **internal** interrupt option is selected (default). When visible, the pin must be connected to a valid clock source. Recommended frequency of the clock should be from 100 Hz to 1 kHz. Debouncing interval equals 4 clock periods. Very short debouncing time may not be sufficient for debouncing a button; long debouncing time makes buttons unresponsive. See the **Implementation** section for details.

On PSoC4 this pin can't be directly driven by a clock and must be bypassed using either a UDB component or a pin; alternatively WDT timer can be used in conjunction with the **external** interrupt option (Figure 1).

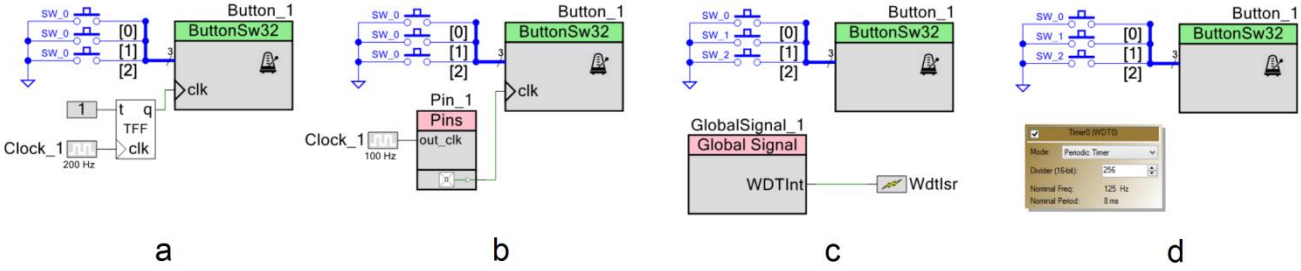
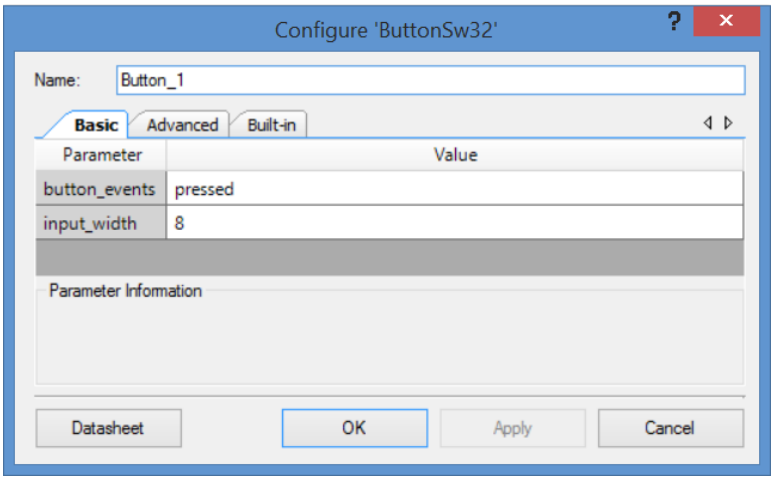


Figure 1. PSoC4 clock options: (a) using TFF, (b) using pin clock, (c) using DWT timer with user-provided interrupt, (d) using DWT timer with auto generated interrupt. Examples (a, b) use internal interrupt, examples (c, d) use external interrupt option.

Parameters and Settings

Basic dialog provides following parameters[*]:



button_events (pressed / released / pressed & released)

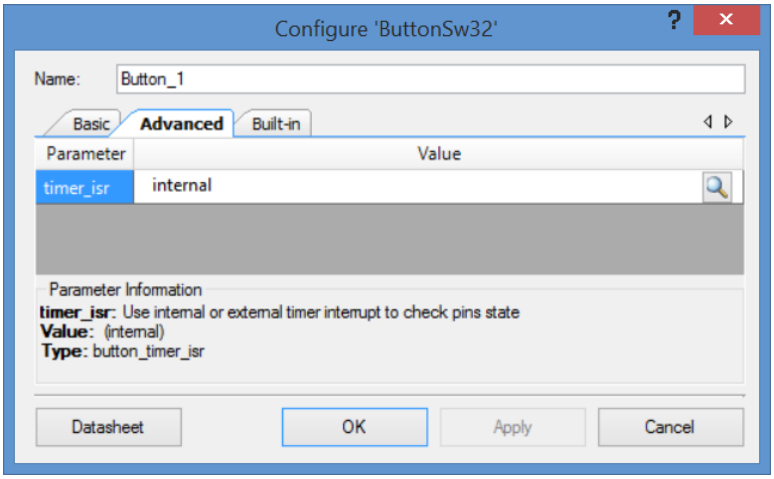
Selects button events being detected (pressed, released or both). This value can't be changed during run time. This option saves CPU clocks by conditionally removing pieces of the unused code. See the **Performance** section for details.

input_width (uint8)

Number of buttons processed simultaneously by the component. Valid range is from 1 to 32. This value can't be changed during run time.

[*] Component was intentionally compiled using Creator 4.0 for compatibility with older versions.

Advanced dialog provides following parameters:



timer_isr (internal / external)

Selects the internal or external timer interrupt for polling pins. The **internal** option automatically configures built-in interrupt on component startup. The **external** option should be used when a user-provider interrupt is configured for pins status check, or when DWT is selected as a polling clock source[(*)]. By default the component is configured for internal interrupt. When **external** option is selected, the clock input becomes hidden. It is user responsibility to handle external polling interrupt.

[*] See DWT examples for PSoC4 in the Application Note.

Application Programming Interface

Function	Description
`Button_Start()`	Initialize and start component
`Button_Stop()`	Stop component
`Button_CheckStatus()`	Process input pins status
Variable	**Description**
`Button_Pressed`	Button pressed flag
`Button_Released`	Button released flag

void Button_Start()

Description: Initializes and starts component. Starts internal polling interrupt if such option is selected. This function has no effect when external option is selected. In that case it is user responsibility to handle external polling interrupt.

Parameters: none

Return Value: none

void Button_Stop()

Description: Stops and disables component. Stops internal polling interrupt if such option is selected. This function has no effect when external option is selected. In that case it is user responsibility to handle external polling interrupt.

Parameters: none

Return Value: none

void Button_CheckStatus()

Description: Samples input pins and processes pins status. This function is called automatically by the internal interrupt when **internal** option is selected. If **external** option is selected, then this function must be called in the user-

specified interrupt routine. It is user responsibility to handle external polling interrupt.

Parameters: none

Return Value: none

uint32 Button_Pressed

Description: Flag indicating button pressed event.

Check this flag in the main() loop to detect button pressed event (a non-zero value indicates that some buttons has been pressed). The flag will rise after debouncing algorithm finds that a button was detected in closed state for the last 4 consecutive clock cycles. The flag will not reset automatically upon read - it must be reset to zero to avoid reentrancy. The flag is latching, and represents all button pressed events since the time it has been reset to zero. See **Implementation** section for details.

Return Value: Each bit in the 32-bit word corresponds to a single button, starting from LSB (1 – button pressed, otherwise return value is 0).

uint32 Button_Released

Description: Flag indicating button released event.

Check this flag in the main() loop to detect button released event (a non-zero value indicates that some buttons has been released). The flag will rise after debouncing algorithm finds that a button was detected in open state for the 4 consecutive clock cycles. The flag will not reset automatically upon read - it must be reset to zero to avoid reentrancy. This flag is latching, and represents all button released events since it has been reset to zero. See **Implementation** section for details.

Return Value: Each bit in the 32-bit word corresponds to a single button, starting from LSB (1 – button released, otherwise return value is 0).

Functional Description

Mechanical switches typically require some signal conditioning due to the bouncing of their contacts [1]. In result, the signal transition between the logical states occurs erratically before settling to a final value. Depending on switch type, the bouncing time may vary anywhere from microseconds to over than 100 ms [1].

Typical switch connection to the microcontroller input and signal traces are shown on Figure 2. Configuring PSoC input pin for resistive pull-up allows for direct switch connection without using extra parts.

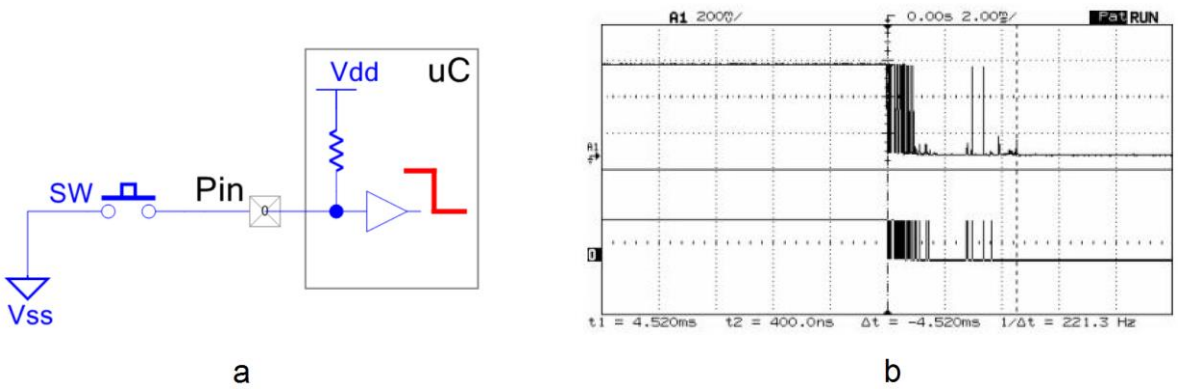


Figure 2. (a) Standard switch connection to the microcontroller; (b) typical signal traces: top – analog signal generated by closing a switch, bottom – digital pattern observed by microcontroller [1]. Timescale 2ms/div.

General observations of the bouncing phenomenon:

- Bouncing time correlate with size of the switch: tactile micro-switches have shortest bouncing time, while panel-mounted “industrial control” switches have the longest one.
- Bouncing time for releasing switch contacts is usually several times longer than for closing them.
- Capacitor-smoothed input signal may still result in the bouncing of the digital signal during voltage transition through CMOS threshold region.

Numerous hardware and software approaches exist to remedy switch transition uncertainty problem [2]; there is no single solution that fits all cases. Current implementation of the Debouncer utilizes processing algorithm implemented entirely in software code. Such approach saves PSoC valuable hardware resources, but is limited to low frequencies of operation due to the CPU involvement. Fortunately, it is totally sufficient for hand-operated manual switches, where response time of 10-50 ms is usually acceptable.

Implementation

Component implements switch debouncing using the vertical counters algorithm [4]; see **Appendix 1** for details.

The state of input pins is being polled by CPU on each sampling clock rising edge. The 2-bit counter algorithm is looking for the 4 (four) consecutive persistent states to confirm a valid transition, rising Pressed or Released flags correspondingly. These flags are latching, and any subsequent press or release events are ignored until the flag is cleared. The flags are being evaluated in the application main loop. These flags are not reset automatically and must be reset to zero to avoid reentrancy. Latching the flags helps capturing of button events when controller is busy executing other tasks and can't respond instantly to the button change of state. This can happen, for example, when pressing the button starts some lengthy task, which can mask consecutive button release event.

Sampling clock selection

Component debouncing time ($T_{DEBOUNCE}$) is set by the frequency of the sampling clock. For a 2-bit counter the debouncing takes 4 clocks, therefore sampling clock frequency:

$$F_{CLOCK} = 4/T_{DEBOUNCE} \quad (1)$$

Typically, polling frequency of about 100 Hz (debouncing time 40 ms) is sufficient for normal manual operation of a button switch and provides good responsivity. Using polling clock slower than 50 Hz makes controller response feel sluggish. Fast polling rate produces faster response, but consumes more CPU resources and is limited by the performance of the switch.

Maximum rate of the polling clock is limited by the bouncing time of a switch:

$$F_{CLOCK} < 4/T_{MIN}\ , \quad (2)$$

where T_{MIN} - is minimal acceptable time interval, which is sufficient for debouncing the switch. For example, for a switch described on Figure 2b, minimum debouncing interval for pressed event is about 5 ms. Thus, maximum acceptable polling clock frequency for debouncing button press events is $F_{CLOCK} < 4/5.10^{-3}$ s = 800 Hz. Note that bouncing time for button release event is typically longer then for press event. Since the component utilizes single clock for both pressed and released events, the longer time should be taken in consideration when selecting the clock frequency.

Input pins configuration

To parse buttons state, the component utilizes buried pins. By default, the pins are configured as resistive pull up on startup; only job left to user is to assign inputs in the Pin Configuration window.

Decoder pins configuration is shown on Figure 3. The pin array must be contiguous (belong to same port and be consecutive). The off-chip button switch component is provided merely for annotation purpose; its presence on the schematic does not affect operation of the component.

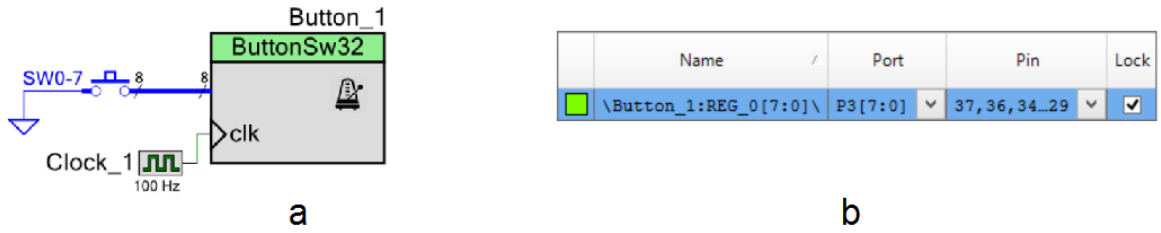


Figure 3. Input Pins configuration: (a)- component appearance on schematic, (b)- pins configuration (assignment is contiguous).

Example of pins assignment in case of multiple instances of the component is shown on Figure 4. The pin groups can be assigned to any available ports, including same port as long as there is enough space available for pin placement.

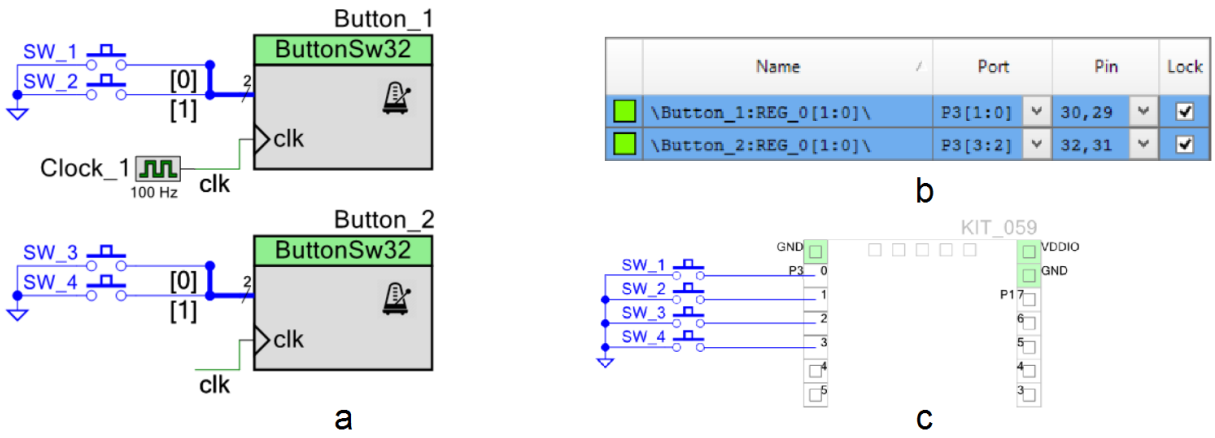


Figure 4. Pin configuration for multiple instances of the Decoder: (a) components appearance on schematic, (b) pin configuration window, (c) connection schematic for CY8KIT-059.

Performance

Component was tested using PSoC5LP (CY8KIT-059) and PSoC4200 (CY8CKIT-042 Pioneer Kit). The component doesn't use UDB, performing all operation entirely by CPU. The interrupt code consumes less or about 50 clocks, making total CPU load very small (~0.01%). Typical results for PSoC5LP are shown below. PSoC4 performance is slightly lower. Component performance decreases linearly with number of ports occupied by input pins.

Table 1. PSoC5LP typical CPU clocks consumption by interrupt code.

Option	1x8 pins		4x8 pins	
	press	press & release	press	press & release
debug(*)	31	38	47	55
release(†)	26	37	37	47

(*) data collected in debug mode with compiler optimization turned off

(†) data collected in release mode with compiler optimization set to speed

Resources

Component resources consumption is provided below. The component does not consume UDB resources. Component does not have built-in DMA capabilities.

Table 1. PSoC5 and PSoc4 resources consumption.

Resource	PSoC5 / PSoC4		PSoC4 (WDT)	
	internal	external	user provided	auto generated
interrupts	1	1	1	0
clocks	1	1	0	0

Sample Firmware Source Code

Basic use of the component is shown in **Appendix 2**. Demo projects are provided.

Component Changes

Version	Description of changes	Reason for changes/impact
0.0	Version 0.0 is the first beta release of the component	

References

1. J. Ganssle, A Guide to Debouncing, http://www.ganssle.com/debouncing.htm
2. M. Szczys, Debounce code, https://hackaday.com/2010/11/09/debounce-code-one-post-to-rule-them-all/
3. E. Williams, Debouncing noisy buttons, https://hackaday.com/2015/12/09/embed-with-elliot-debounce-your-noisy-buttons-part-i/
4. Debouncing switches with vertical counters, https://www.compuphase.com/electronics/debouncing.htm

Appendix 1

The 2-bit vertical counters algorithm

In the vertical counters algorithm the inputs are sampled and processed on timer event [4]. The code is called by the polling interrupt, and checks for 4 consecutive persistent button state to confirm change of state:

```
uint32 sample;                          // raw (undebounced) pin state
static uint32 state;                    // debounced pin state
static uint32 cnt0, cnt1;               // vertical counters

sample  = ~Pin_Read();                  // read raw pin state (undebounced)
                                        // 1-pin pressed, 0-pin released

delta = sample ^ state;                 // get difference from debounced state
cnt1 = (cnt1 ^ cnt0) & delta;           // increment bit 1
cnt0 = ~cnt0 & delta;                   // increment bit 0

toggle = delta & ~(cnt0 | cnt1);        // detect change of the debounced state
                                        // after both counters are cleared

state ^= toggle;                        // get debounced state

Button_Pressed  |= toggle & state;     // latch button pressed event and
Button_Released |= toggle & ~state;    // latch button released event
                                        // for further processing in main loop
```

The press and release events are latched for further processing in the main loop. These flags must be cleared to allow further detection of the button events. This latching mechanism helps detecting buttons events even when CPU is busy with other tasks and can't respond instantly to the button state change. This can happen, for example, when pressing the button starts some lengthy task, masking consecutive button release event. Main loop code:

```
for(;;)                                 // main loop
{
  if (Button_Pressed != 0)              // check for button pressed event
  {
     uint32 status = Button_Pressed;    // capture buttons state
     Button_Pressed = 0;                // clear flag

     if (status & 1u) {some action}     // process button #0 pressed event
     if (status & 2u) {some action}     // process button #1 pressed event
     if (status & 4u) {some action}     // process button #2 pressed event
     . . .
  }
}
```

Timing diagram for a single button press and release operation is shown on Figure 5. When button is pressed (sample=HIGH), counters cnt0 and cnt1 continue to increment while the input reading differs from the debounced state. The debounced state toggle after both counters are cleared. Four (4) consecutive sample=HIGH readings are necessary to toggle the state. Sequence of HIGH samples, which is shorter than 4 clocks resets the counters. Since button press and release operation is not intrinsically synchronized with the sampling clock, the actual response time may vary between 3 to 4 clock periods.

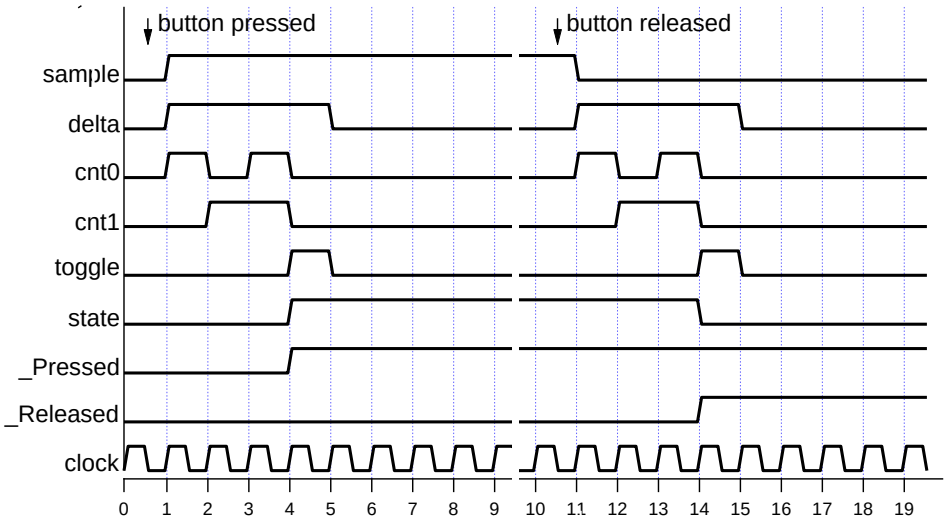


Figure 5. Timing diagram of the algorithm response to button press and release events.

Appendix 2

Basic examples using the component

Several demo projects provided along with the Application Note, showing component use with PSoC4 and PSoC5. Basic example for PSoC5 is shown on Figure 6. The component is configured for internal interrupt for processing input from 8 buttons simultaneously.

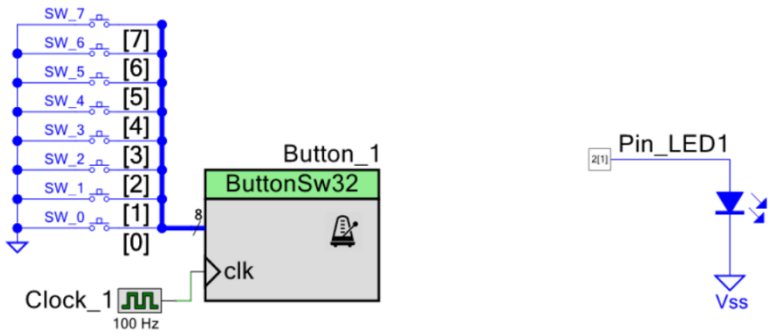


Figure 6. PSoC5 Button debouncing demo using internal polling interrupt.

Since PSoC4 has limited hardware resources it is convenient to use WDT timer as a slow polling clock source. On PSoC4 the clock input can't be directly driven by the Global Signal, as internal interrupt is not capable of clearing the interrupt source. In this case the component should be configured for external interrupt option (Figure 7), and pins status processed inside user-provided interrupt using component API CheckStatus() function.

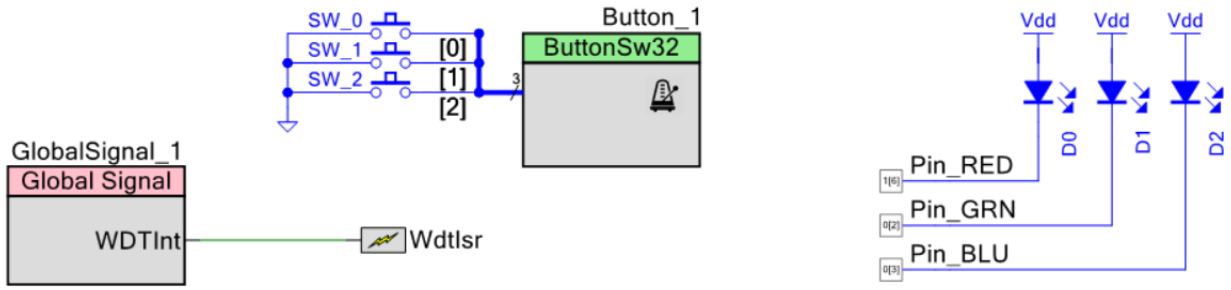


Figure 7. PSoC4 button debouncing demo using external WDT timer.

Appendix 3

Off-chip annotation components

The Debouncer component is accompanied with few off-chip annotation components facilitating schematic drawing and enhancing visibility of the Debouncer component (Figure 8). They can be also used in conjunction with Annotation library for kits CY8KIT-059 and CY8KIT-042.

Figure 8. Off-chip annotation components: (a) small button switch, (b) wide button switch.

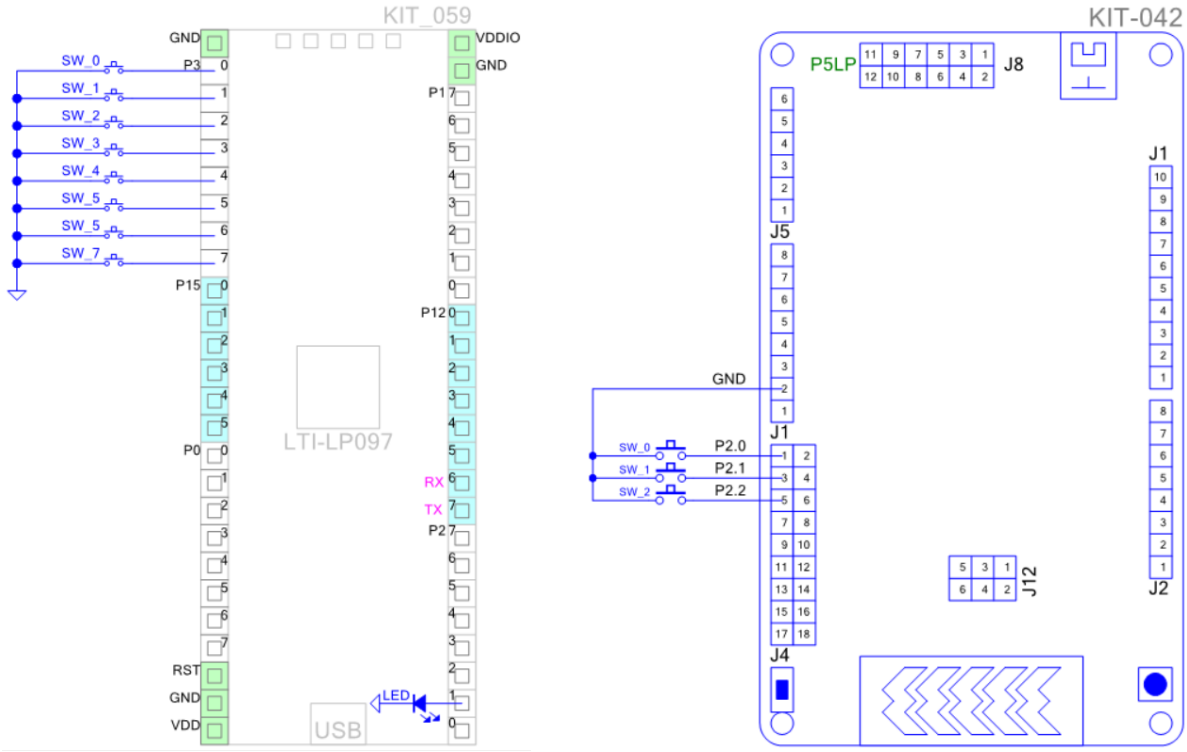


Figure 9. Schematic examples using CY8KIT-059 off-chip annotation library(*) and KEES PioneerBoard annotation library.

* CY8KIT-059 annotation library community component: http://www.cypress.com/forum/psoc-community-components/annotation-library-cy8ckit-059-prototyping-kit