**Please note that Cypress is an Infineon Technologies Company.**
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

**Continuity of document content**
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

**Continuity of ordering part numbers**
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

# CE97311 - PSoC® 4 M: CAN Simplex Communication with CapSense®

## Objective

This code example demonstrates how to send and receive data over the CAN bus. The status of the CapSense® Gesture Pad is sent by the CAN transmitter to control LEDs at the CAN receiver.

## Overview

This code example demonstrates the configuration and use of the CAN Tx and CAN Rx mailboxes. Five CapSense buttons on the CapSense® Gesture Pad in CY8CKIT-044 are configured to control the ON/OFF status, color, and brightness of an RGB LED. This data is transmitted over the CAN bus; at the CAN receiver, the RGB LED is configured to reflect the received data.

## Requirements

**Tool:** PSoC Creator™ 3.3 SP1 or later

**Programming Language:** C (ARM® GCC 4.9.3 and ARM MDK compilers)

**Associated Parts:** PSoC 4200M and PSoC 4100M

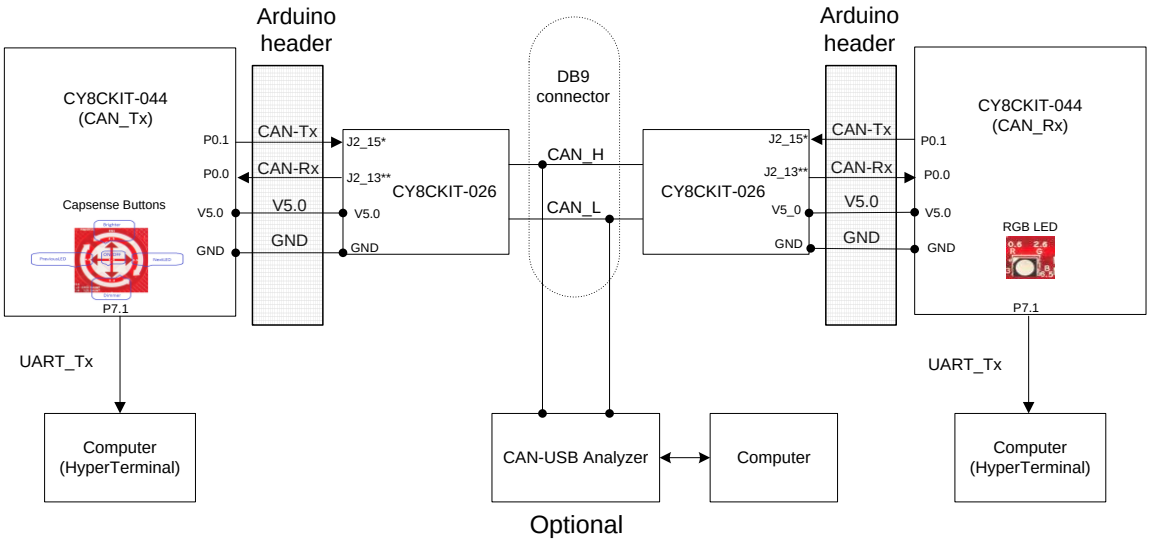
**Related Hardware:** Two of CY8CKIT-044, two of CY8CKIT-026, DB9 cable with male connector on both ends, Jumper wires[1].

[1]Wires that are used to connect from CY8CKIT-026 Arduino headers to CAN Tx and CAN Rx pins on the same board.

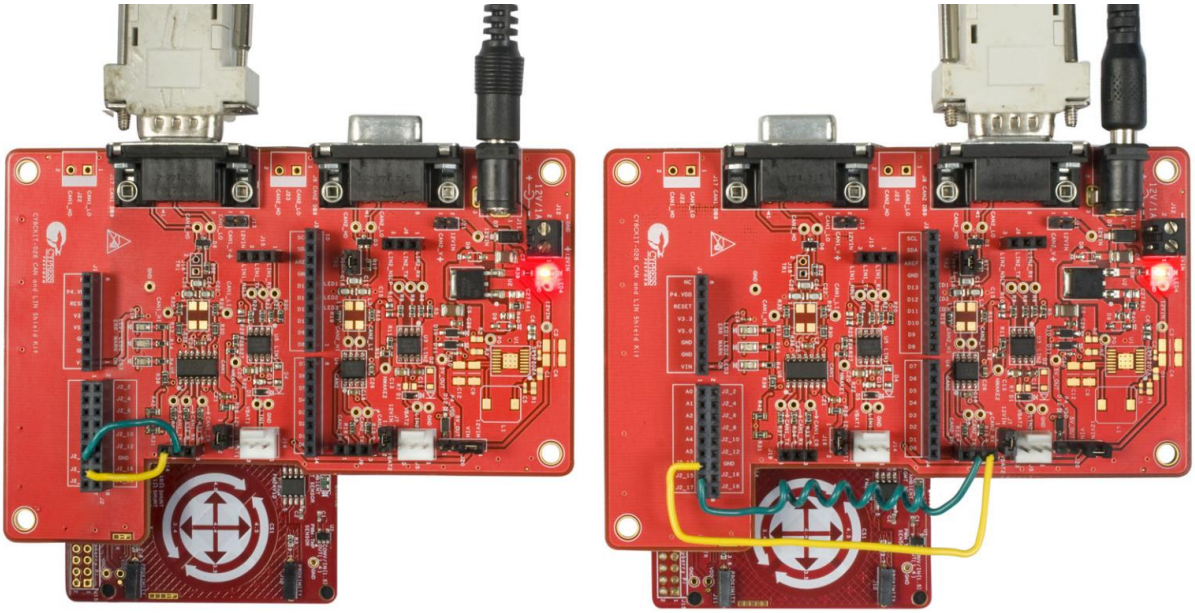
## Design

This code example consists of two projects: one for the CAN transmitter, and one for the CAN receiver. Both projects are used together to test the operation. Figure 1 shows the hardware block diagram of the CAN Tx and Rx project. Figure 1 also shows the hardware connections that need to be done on CY8CKIT-026 (not all connections are shown as there are multiple sets of connections).

Figure 1. CAN Network Topology and Hardware Connections

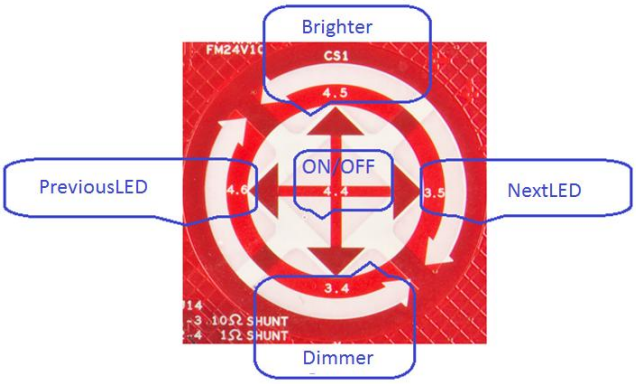



* Connect J2_15 to CAN1_Tx (J19_1) or CAN2_Tx (J9_3)
** Connect J2_13 to CAN1_Rx (J19_2) or CAN2_Rx (J9_2)



The transmitting CY8CKIT-044 board (CAN_Tx) has a CapSense Gesture pad. Each button on the Gesture pad controls the RGB LED on the receiving CY8CKIT-044 board (CAN_Rx) as shown in Figure 2.

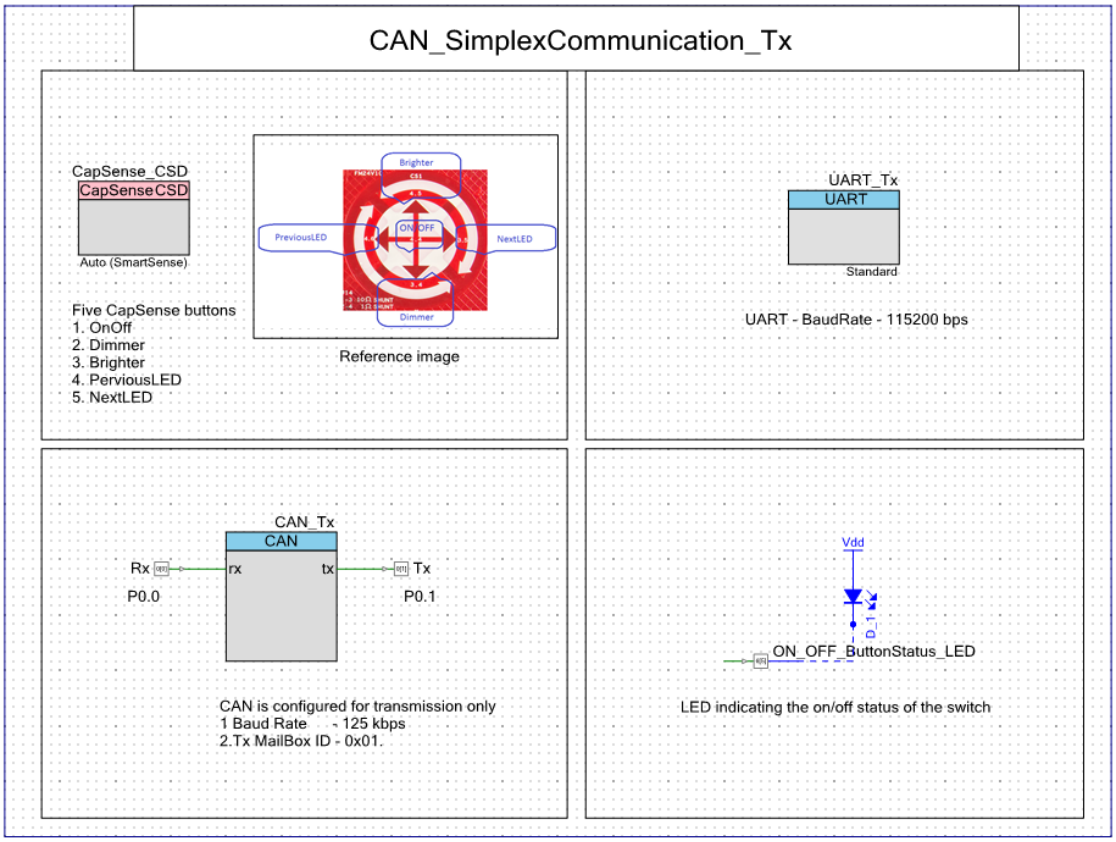
Figure 2. CapSense Gesture Pad with Functionality




## CAN_SimplexCommunication_Tx Project

In this example, PSoC 4 acts as a simple CAN transmitter as shown in Figure 3. Here, the CapSense button status is scanned and stored as a packet of three bytes consisting of the ON/OFF status, color, and brightness of the RGB LED. This data is transmitted over the CAN bus.

Figure 3. CAN Transmitter Design with PSoC 4200M




This document concentrates only on the CAN Component and its configuration.

## Design Considerations

- CAN is a fixed-function Component. Dedicated pins must be used. In this project, P0.0 (Port 0, Pin 0) is used for CAN_Rx, and P0.1 is used for CAN_Tx.
- The UART Component is used to display the transmitted CAN messages on a serial terminal. The UART is used as Tx only, and the UART Tx pin is assigned to P7.1. This pin is connected on CY8CKIT-044 to a built-in USB-Serial Bridge so that the UART data can be displayed using any terminal emulator program when the kit is connected to the USB port. The UART settings are: baud rate – 115200, parity – none, stop bit – 1. See the CY8CKIT-044 kit user guide for additional details on the USB-serial Bridge functionality.

## Components

Table 1 lists the PSoC Creator Components used in the CAN_SimplexCommunication_Tx project, as well as the hardware resources used by each.

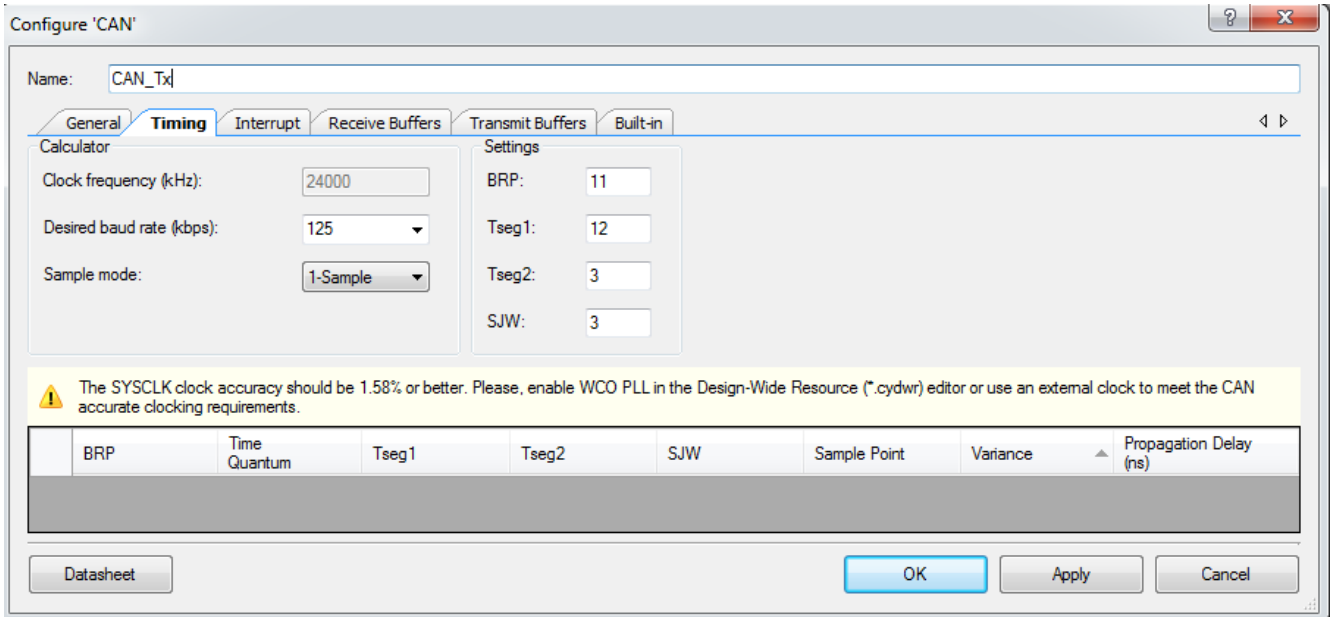
Table 1. PSoC Creator Components – CAN_SimplexCommunication_Tx

| Component | Hardware Resources |
|---|---|
| CAN | 1 CAN block |
| CSD | 1 CapSense CSD block |
| UART | 1 SCB UART |
| Pins | 2 CAN: Rx and Tx<br>5 CapSense buttons (part of the CapSense Component)<br>1 Cmod (part of the CapSense Component)<br>1 UART: Tx<br>2 WCO (used to generate an accurate clock)<br>1 LED |

## Parameter Settings

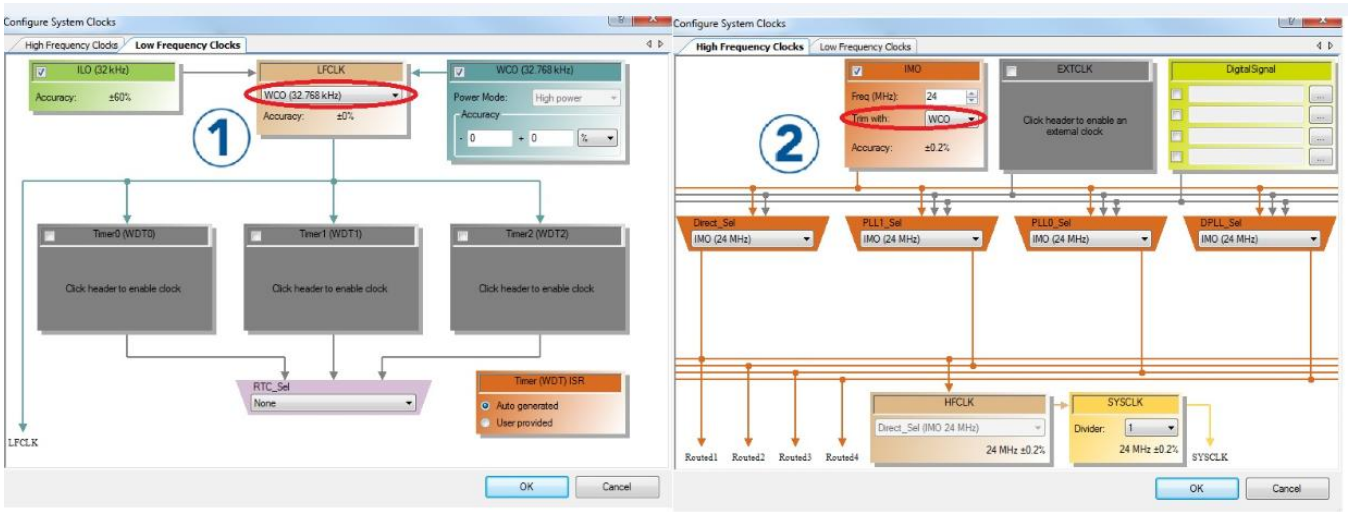
The CAN Component needs a clock with an accuracy of ±1.58 percent or better. The internal main oscillator (IMO) in PSoC 4 M is only ±2 percent accurate; therefore, you must either trim the IMO or use an external clock. These settings must be done before the CAN Component is configured because the timing section in the Component configuration will be updated based on this clock. If the clock is not selected before the Component is configured, the Component displays a warning to do so as shown in Figure 4.

Figure 4. CAN Component Clock Accuracy Warning




In the clock configuration wizard of the *CAN_SimplexCommunication_Tx.cydwr* file, the IMO can be trimmed to an accuracy of 0.5 percent using the watch crystal oscillator (WCO) in PSoC 4 M. To trim the IMO, first select the LFCLK as the WCO and select **Trim with WCO** in the **IMO** section, as shown in Figure 5.

Figure 5. Clock Configuration

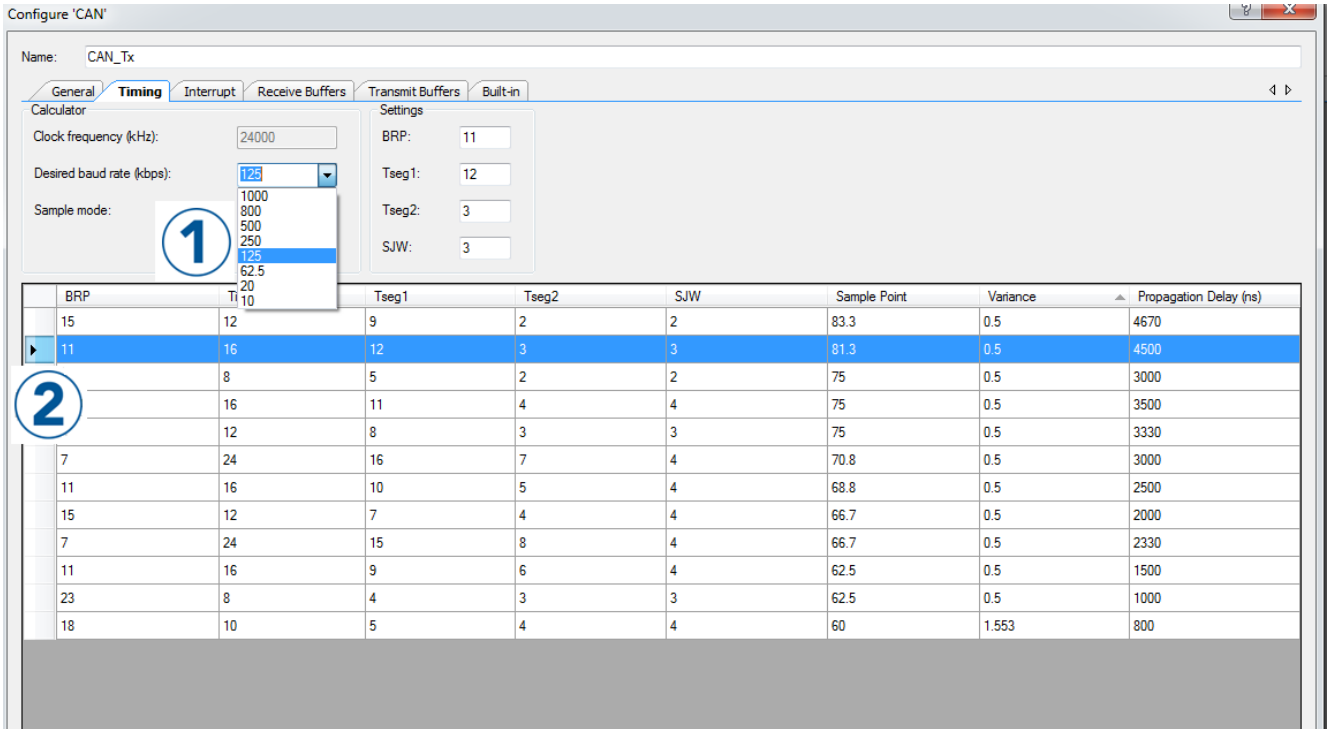



Now, you can go back to the TopDesign to configure the CAN Component.

The settings in the **General** tab are left with the default selections.

In the **Timing** tab, select the **Desired baud rate**. The table with different timing parameters will be updated based on the selected baud rate. Select one of the rows with the criteria that **Variance** is minimal and **Sample point** is around 80 percent, as shown in Figure 6. Double-click on that row to make sure that the values are updated in the settings section.

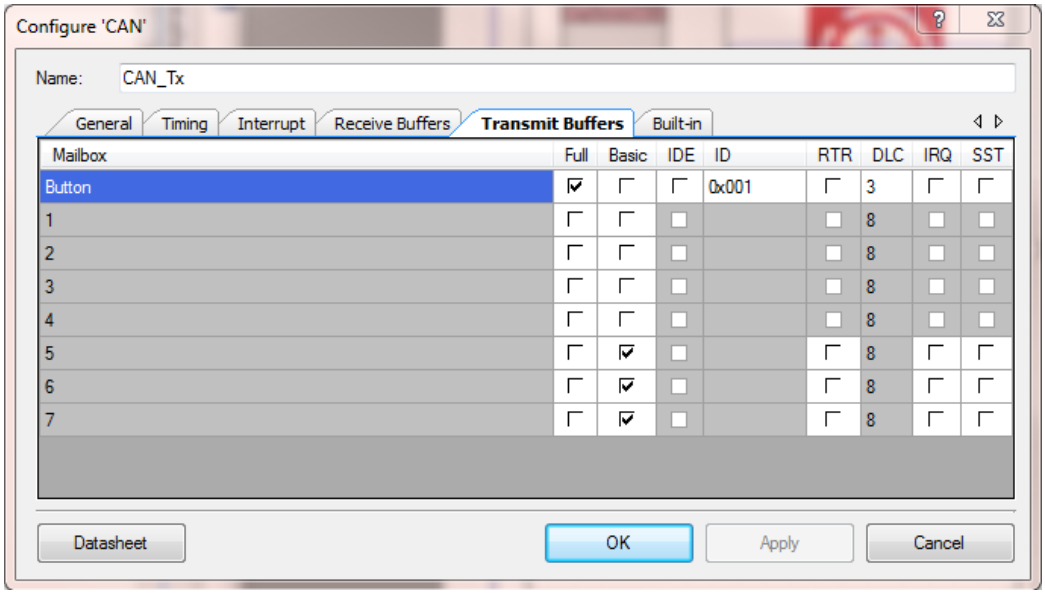
Figure 6. CAN_Tx Configuration – Timing Tab




The **Interrupt** and **Receive Buffers** are left with the default configuration.

In the **Transmit Buffers** tab, add a **Full** Tx mailbox with mailbox **ID** ‘0x01’. Rename the mailbox to “Button” and change the DLC as ‘3’, as shown in Figure 7.

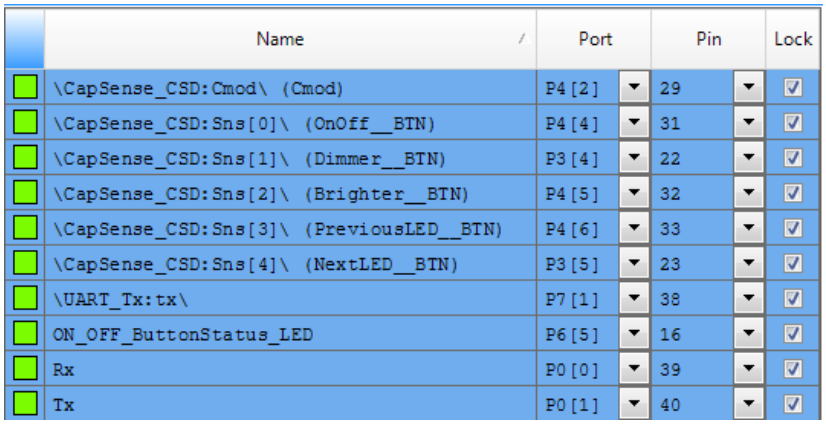
Figure 7. CAN_Tx Configuration – Transmit Buffers Tab



## Pin Assignment

Figure 8 shows the CAN_SimplexCommunication_Tx project pin assignment.

Figure 8. CAN_SimplexCommunication_Tx – Pin Assignment



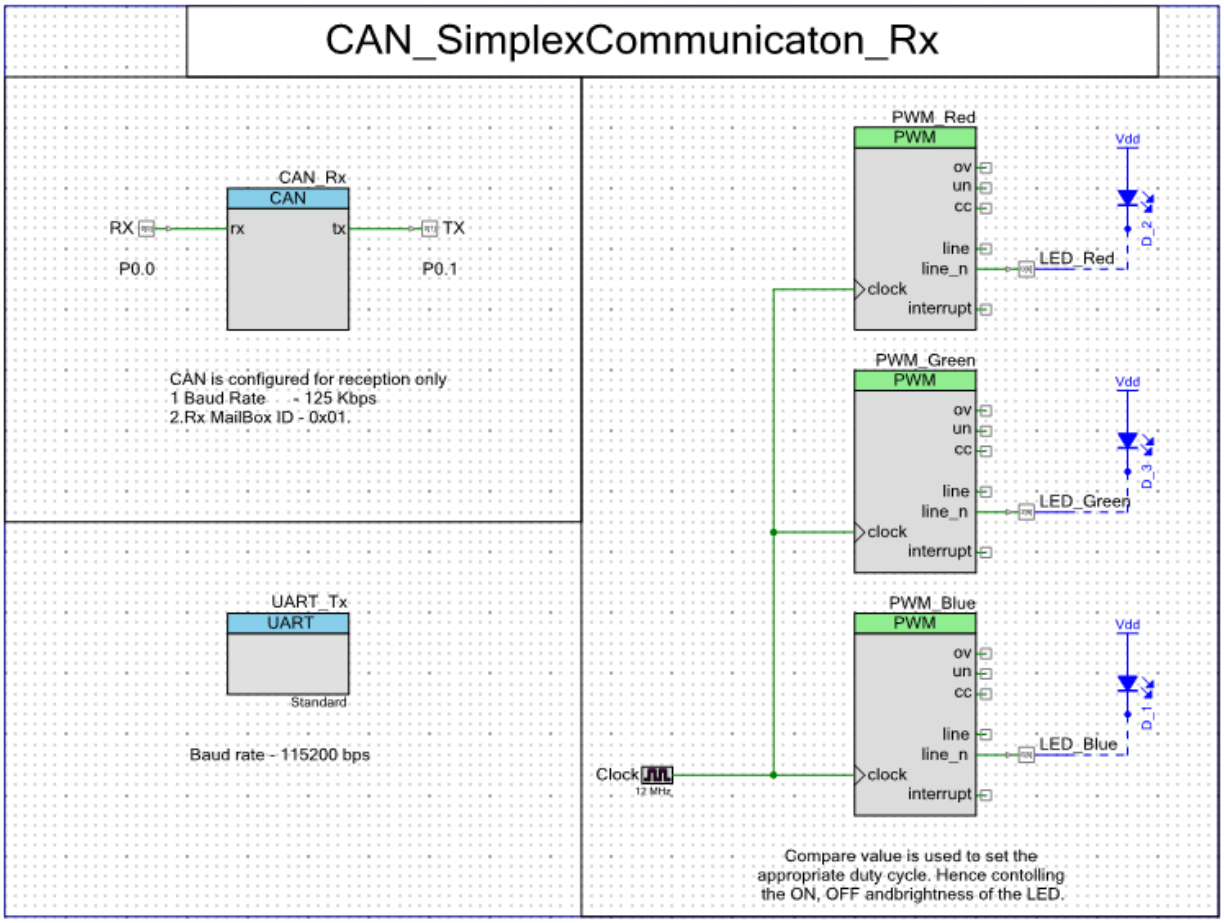
| Name | Port | Pin | Lock |
|---|---|---|---|
| \CapSense_CSD:Cmod\ (Cmod) | P4[2] | 29 | ☑ |
| \CapSense_CSD:Sns[0]\ (OnOff__BTN) | P4[4] | 31 | ☑ |
| \CapSense_CSD:Sns[1]\ (Dimmer__BTN) | P3[4] | 22 | ☑ |
| \CapSense_CSD:Sns[2]\ (Brighter__BTN) | P4[5] | 32 | ☑ |
| \CapSense_CSD:Sns[3]\ (PreviousLED__BTN) | P4[6] | 33 | ☑ |
| \CapSense_CSD:Sns[4]\ (NextLED__BTN) | P3[5] | 23 | ☑ |
| \UART_Tx:tx\ | P7[1] | 38 | ☑ |
| ON_OFF_ButtonStatus_LED | P6[5] | 16 | ☑ |
| Rx | P0[0] | 39 | ☑ |
| Tx | P0[1] | 40 | ☑ |

## CAN_SimplexCommunication_Rx Project

This project receives CAN data sent from the CAN_SimplexCommunication_Tx project. The received data is a packet of three bytes consisting of the ON/OFF status, color, and brightness. The color and brightness of the RGB LED on the receiving CY8CKIT-044 is controlled based on the received data.

Figure 9 shows the PSoC Creator schematic design of the code example.

Figure 9. CAN Receiver design with PSoC 4200M




This document concentrates only on the CAN Component and its configuration.

## Design Considerations

- CAN is a fixed-function Component. Dedicated pins must be used. In this project, P0.0 is used for CAN_Rx, and P0.1 is used for CAN_Tx.
- The UART Component is used to display the received CAN messages on a serial terminal. The UART is used as Tx only, and the UART Tx pin is assigned to P7.1. This pin is connected on CY8CKIT-044 to a built-in USB-Serial Bridge so that the UART data can be displayed using any terminal emulator program when the kit is connected to the USB port. The UART settings are: baud rate – 115200, parity – none, stop bit – 1. See the CY8CKIT-044 kit user guide for additional details on the USB-serial Bridge functionality.
- Three PWM Components are used to control three LEDs.
- The LEDs are connected to the line_n outputs of the PWMs because the LEDs are active LOW in CY8CKIT-044.

## Components

Table 2 lists the PSoC Creator Components used in the CAN_SimplexCommunication_Rx project, as well as the hardware resources used by each.

Table 2. PSoC Creator Components – CAN_SimplexCommunication_Rx

| Component | Hardware Resources |
|---|---|
| CAN | 1 CAN block |
| PWMs | 3 TCPWMs |
| UART | 1 SCB UART |
| Pins | 2 CANs: Rx and Tx<br>3 LEDs<br>1 UART: Tx<br>2 WCO (used to generate an accurate clock) |
| Clock | 1 Clock Component |

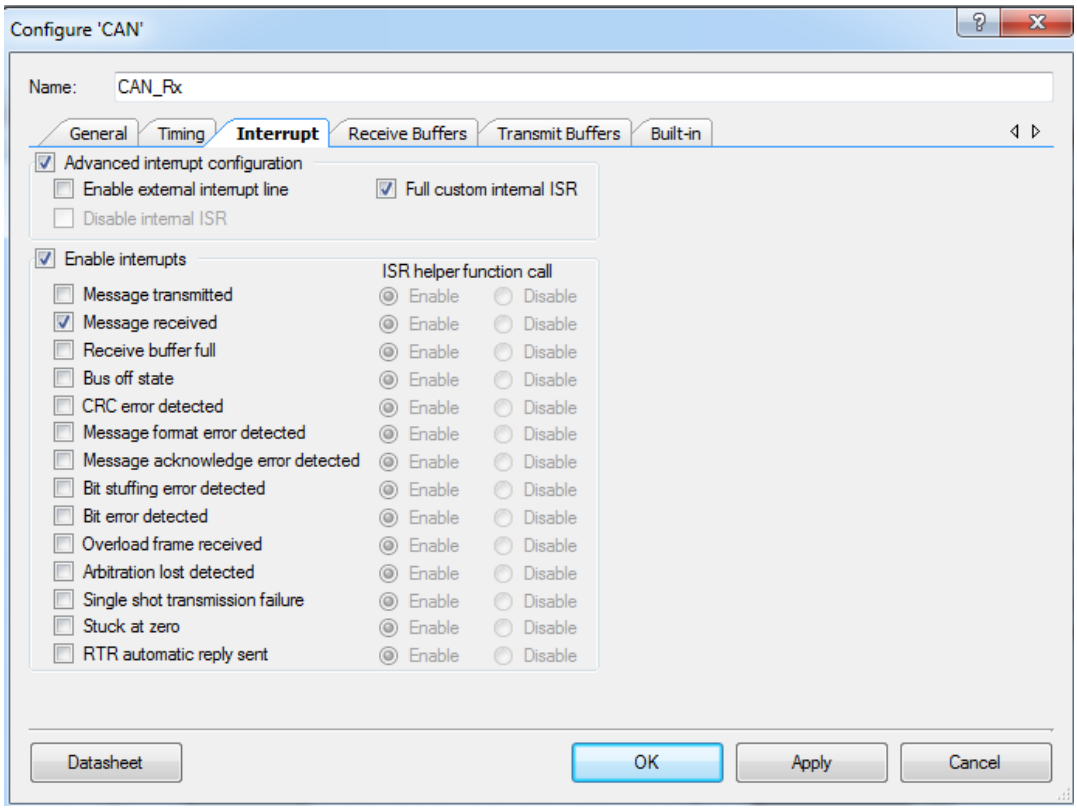
## Parameter Setting

Timing Figure 3 and Clock configuration Figure 4 for the CAN_Simplex_Rx project are same as that of the CAN_Simplex_Tx project.

In the **Interrupt** tab, enable **Advanced interrupt configuration** and select **Full custom internal ISR,** as shown in Figure 10

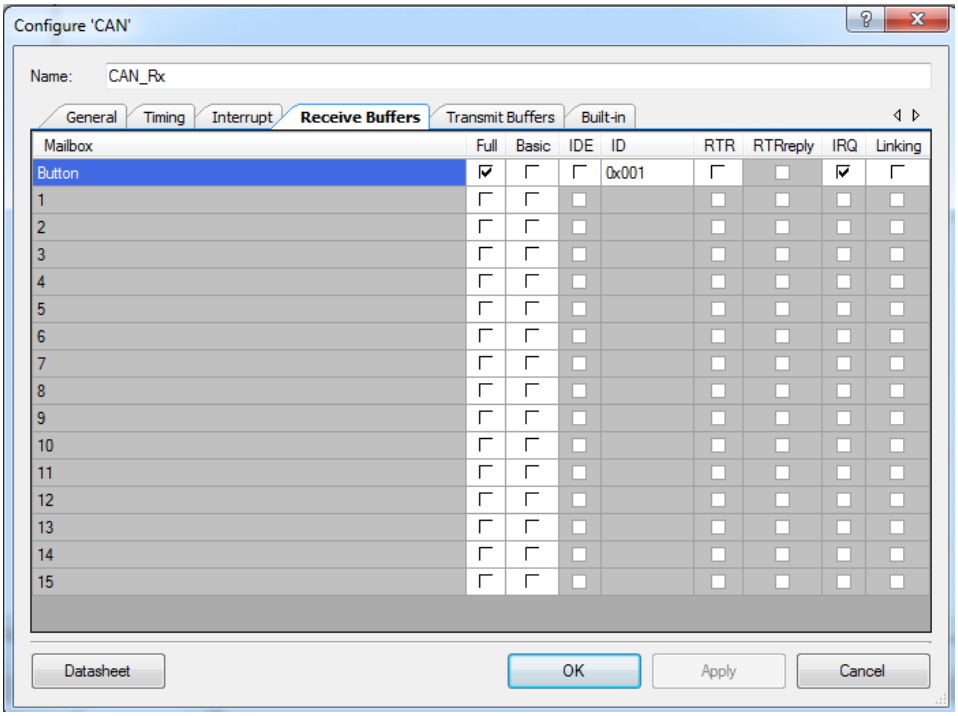
.

Figure 10. CAN_Rx Configuration – Interrupt Tab



In the **Receive Buffers** tab, add a **Full** Rx mailbox with mailbox **ID** '0x01', which is the same as the Tx mailbox ID in the CAN_Simplex_Tx project. Rename the mailbox to "Button," as shown in Figure 11. **IRQ** will be selected by default because the **Message received** interrupt is selected in the **Interrupt** tab.

Figure 11. CAN_Rx Configuration – Receive Buffers Tab

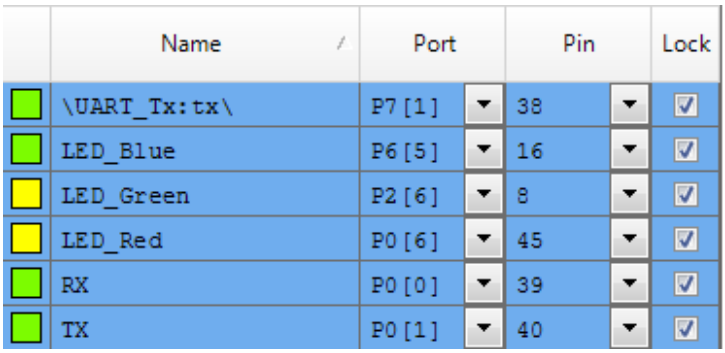



The **Transmit Buffers** tab is left with the default settings.

## Pin Assignment

Figure 12 shows the CAN_SimplexCommunication_Rx project pin assignment.

Figure 12. CAN_SimplexCommunication_Rx – Pin Assignment



## Hardware Setup

To use CY8CKIT-026 with CY8CKIT-044, make the following connections:

- Plug in CY8CKIT-026 to CY8CKIT-044 through the Arduino-compatible connectors.
- Because there are two CAN transceivers on CY8CKIT-026, choose to use one of the CAN transceivers (U6 – CAN1 and U4 – CAN2). Connect J2_13 (CAN_Rx) and J2_15 (CAN_Tx) to the appropriate CAN_Rx and CAN_Tx of the transceiver (J19 – CAN1 or J9 – CAN2) as shown in Table 3.

Table 3. Pin Connection on CY8CKIT-026

| Arduino Header Pins | CAN1 Transceiver | CAN2 Transceiver |
|---|---|---|
| J2_13 | J19_2 (CAN1_RX) | J9_3 (CAN2_RX) |
| J2_15 | J19_1 (CAN1_TX) | J9_2 (CAN2_TX) |

- If you have chosen CAN2, then install a jumper at J10 that populates the CAN termination resistors.
- If you have chosen CAN1, then an external 12-V supply must be connected through J11 or J12 (refer to the CY8CKIT-026 user guide for more details on the power supply connections).
- Connect jumper J20 appropriately as per the power connections.

## Operation

### CAN_SimplexCommunication_Tx

Program the CAN_SimplexCommunication_Tx project to the first CY8CKIT-044 that is used as CAN Tx.

- The center ON/OFF button (can be called a “switch”) is used to switch ON/OFF the LEDs.
- When the switch is ON, brightness and color can be varied.
- If the switch is OFF, then brightness and color inputs do not have any effect.
- A Blue LED is used to show the switch status as either ON or OFF. Brightness and color variation are not shown in this project because three PWMs would be needed. This feature is demonstrated in the CAN_SimplexCommunication_Rx project, which is part of this code example.
- The UART is used to show the values that are sent over the CAN bus. Figure 13 shows the UART data.

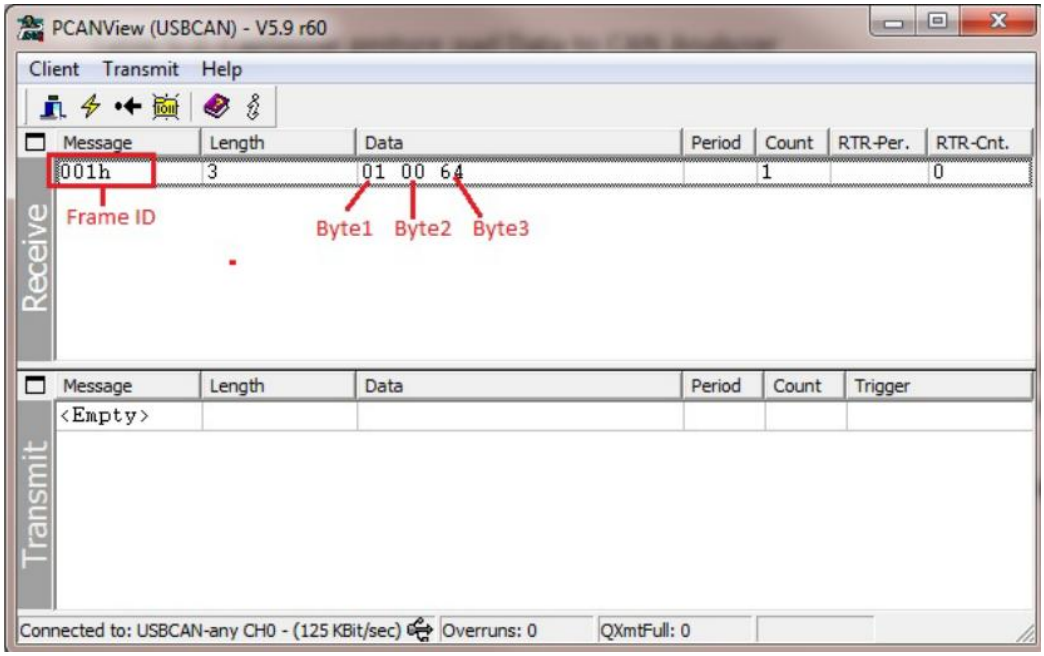
Figure 13. UART Data on HyperTerminal

```
onOffStatus: 0    color:2    brightness:10
onOffStatus: 1    color:2    brightness:10
onOffStatus: 1    color:0    brightness:10
onOffStatus: 1    color:1    brightness:10
onOffStatus: 1    color:1    brightness:40
onOffStatus: 1    color:1    brightness:70
onOffStatus: 0    color:1    brightness:70
```

- “onOffStatus” can be either 0 or 1: 0 – OFF and 1 – ON.
- “color” is a value from 0 to 2, where 0 – Red, 1 – Green, 2 – Blue.
- “brightness” is a value from 10 to 250 in steps of 30. Brightness does not start at zero to differentiate between switch OFF and low brightness.
- onOffStatus, color, and brightness are updated based on user input. A packet with three bytes is sent over the CAN Tx signal.

- Figure 14 shows the CAN data as viewed in a CAN analyzer.

Figure 14. CAN Analyzer Data




## CAN_SimplexCommunication_Rx

Program the CAN_SimplexCommunication_Rx project to the second CY8CKIT-044 that is used as CAN Rx.

- The CAN receiver triggers an interrupt after receives the data in the configured mailbox.
- onOffStatus, color, and brightness values are read after the interrupt is triggered.
- If the onOffStatus is ON, then the RGB LED is configured for the appropriate color and brightness. If the onOffStatus is OFF, then all the LEDs are turned OFF.
- Three PWMs are used for the three LEDs. Because the Blue LED is connected to P6.5, which does not have a connection to the UDB array, a dedicated TCPWM must be used.
- The UART is used to monitor the received data. Figure 15 shows the UART data.

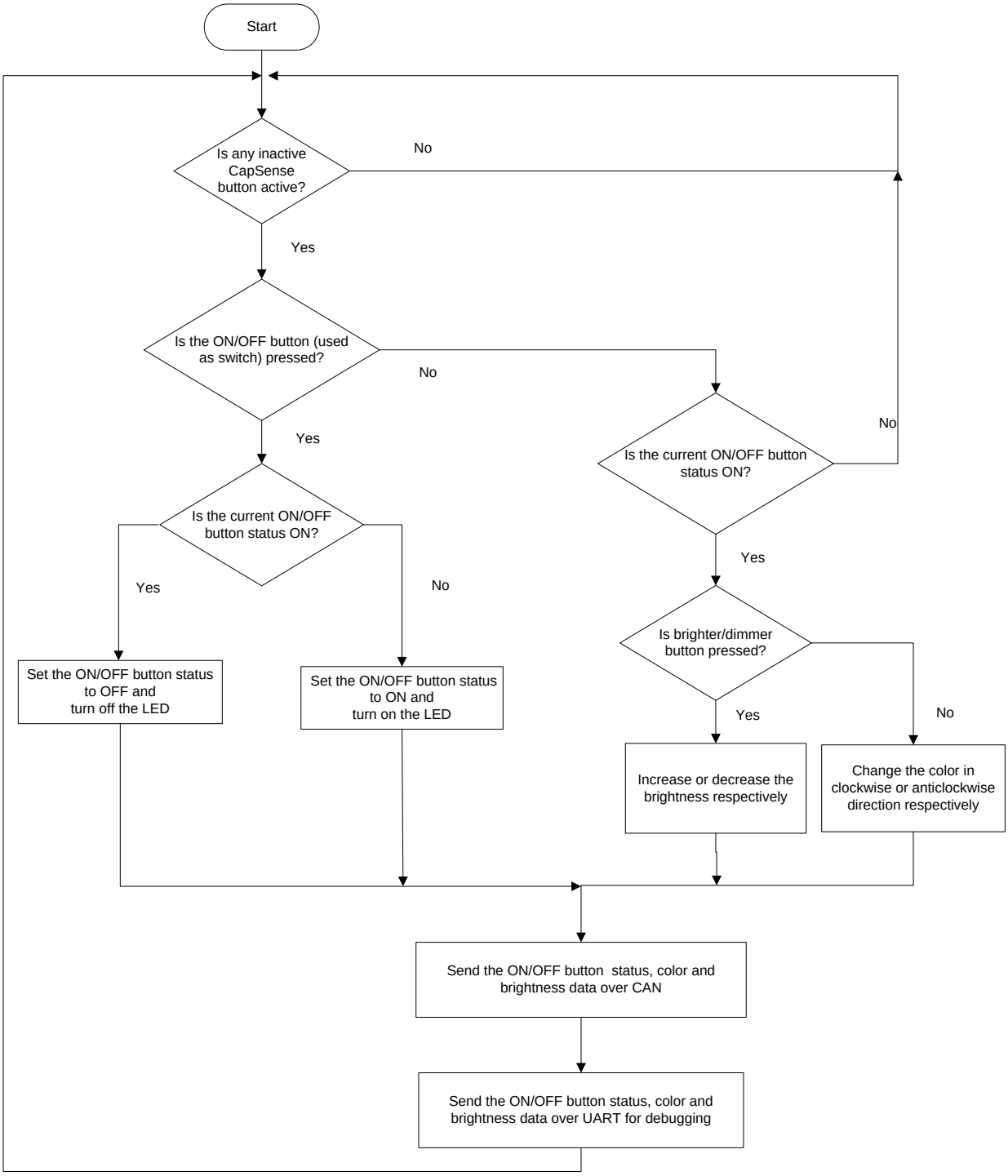
Figure 15. UART Data on HyperTerminal

```
onOffStatus: 0    color:2    brightness:10
onOffStatus: 1    color:2    brightness:10
onOffStatus: 1    color:0    brightness:10
onOffStatus: 1    color:1    brightness:10
onOffStatus: 1    color:1    brightness:40
onOffStatus: 1    color:1    brightness:70
onOffStatus: 0    color:1    brightness:70
```

## CAN_SimplexCommunication_Tx – Firmware Flow Chart

Figure 16 shows the CAN_SimplexCommunication_Tx project's firmware flow chart.

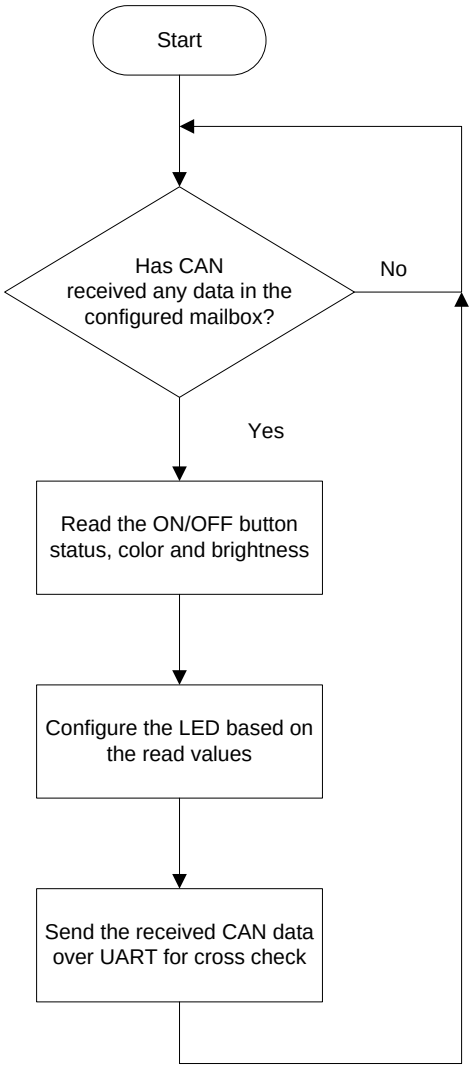
Figure 16. CAN_SimplexCommunication_Tx – Firmware Flow Chart



## CAN_SimplexCommunication_Rx – Firmware Flow Chart

Figure 17 shows the CAN_SimplexCommunication_Rx project's firmware flow chart.

Figure 17. CAN_SimplexCommunication_Rx – Firmware Flow Chart



## Related Documents

Table 3 lists the relevant application notes, code examples, knowledge base articles, device datasheets, and Component datasheets.

Table 3. Related Documents

| |
|---|
| **Application Notes** |
| AN79953 – Getting Started with PSoC 4 |
| **Code Examples** |
| CE95351 – Fixed Function PWM with PSoC 4 |
| **PSoC Creator Component Datasheets** |
| Controller Area Network (CAN) |
| PSoC 4 Controller Area Network v3.0 |
| **Device Documentation** |
| PSoC 4 Datasheets |
| PSoC 4 Technical Reference Manuals |
| **Development Kit (DVK) Documentation** |
| CY8CKIT-044 –PSoC 4 M-Series Pioneer Kit |

## Document History

Document Title: CE97311 – PSoC® 4 M: CAN Simplex Communication with CapSense®

Document Number: 001-97311

| Revision | ECN | Orig. of Change | Submission Date | Description of Change |
|---|---|---|---|---|
| ** | 4773096 | BSKG | 05/21/2015 | New code example |
| *A | 5181058 | BSKG | 03/18/2016 | Replaced CY8CKIT-017 with CY8CKIT-026.<br>Modified the flow of the document. |
| *B | 5740004 | AESATP12 | 05/26/2017 | Updated logo and copyright. |

## Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

## Products

| | |
|---|---|
| ARM® Cortex® Microcontrollers | cypress.com/arm |
| Automotive | cypress.com/automotive |
| Clocks & Buffers | cypress.com/clocks |
| Interface | cypress.com/interface |
| Internet of Things | cypress.com/iot |
| Memory | cypress.com/memory |
| Microcontrollers | cypress.com/mcu |
| PSoC | cypress.com/psoc |
| Power Management ICs | cypress.com/pmic |
| Touch Sensing | cypress.com/touch |
| USB Controllers | cypress.com/usb |
| Wireless Connectivity | cypress.com/wireless |

## PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

## Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

## Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2015-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.