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.

AN221886

Using Smart I/O to Generate a Digital FSK Transmit Signal

Author: Dennis Seguine

Associated Part Family: PSoC 4000S, PSoC 4100S, PSoC 4100S Plus, PSoC 4100PS, CY8C4Axx, PSoC 6 MCU

More code examples? We heard you.

To access an ever-growing list of hundreds of PSoC® code examples, please visit our code examples web page. You can also explore the PSoC video library here.

AN221886 shows how to use PSoC 4 Smart I/O™ to implement the connections and logic necessary to build an FSK transmitter. The design uses no interrupts and no DMA; run-time firmware is limited to startup code. Step-by-step methods show how to build a narrow-band FSK transmitter with minimum phase distortion.

Contents

1 Introduction....1
2 System Structure....2
3 Basic Structure, Step-by-Step....2
4 Improved Structure, Step-by-Step....8
5 Summary....12
6 Related Documents....13
Document History....14
Worldwide Sales and Design Support....15
Products....15
PSoC® Solutions....15
Cypress Developer Community....15
Technical Support....15

1 Introduction

Smart I/O in some PSoC 4 and PSoC 6 MCUs enables logical combination of internal block signals and pin inputs. This is particularly useful in MCUs that do not have Universal Digital Blocks (UDBs) for random logic functions. In devices with Smart I/O, only a subset of GPIO ports support the feature; check the specific part family datasheet to identify ports with Smart I/O.

Smart I/O consists of eight Look-Up-Tables (LUT) and a Data Unit (DU). The LUTs are configurable three-input gates with register capability on the output and on one of the inputs. The DU is an 8-bit data path capable of performing increment, decrement, shift, and AND/OR operations.

The input connections for Smart I/O come from the block signals routed to the pins to which the Smart I/O is assigned, from other LUTs, and from the GPIOs on the assigned port. The LUT outputs can be directly connected to GPIOs or routed back to other LUTs and the DU.

Frequency Shift Keying (FSK) is commonly used in modems, transferring UART data to a modulated frequency. Base-band examples include Bell 102 (300 baud) and Bell 202 (1200 or 2400 baud). Narrow-band FSK is typically used for Power Line Communications (PLC). When the data from the UART is logic level '1,' the output is at the upper modulating frequency. When the data from the UART is at logic level '0,' the output is at the lower modulating frequency.

The waveforms for a typical base-band FSK transmission are shown below. When UART data is low, FSK Out follows f_{LOW}; when UART data is high, FSK Out follows f_{HIGH}.

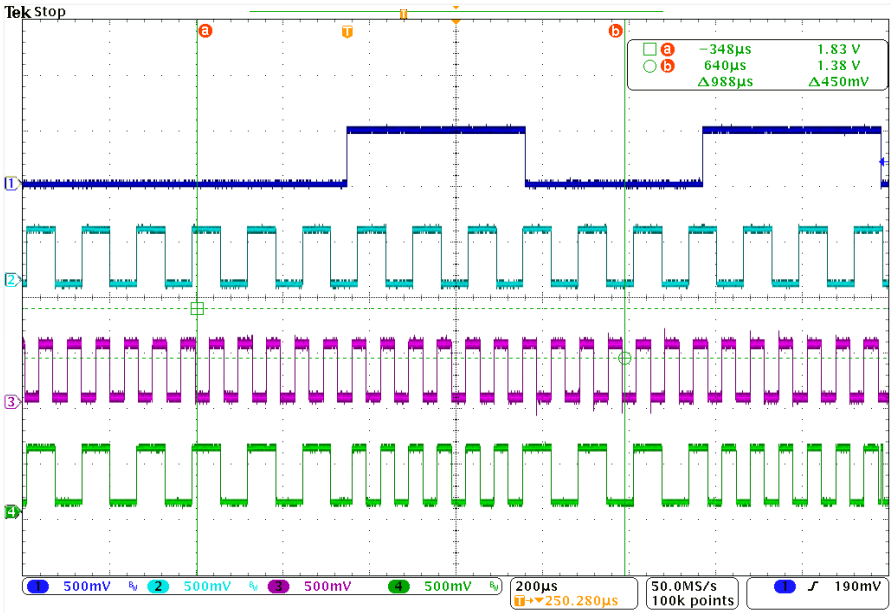
Figure 1. FSK Waveforms

Trace 1: UART data

Trace 2: f_{LOW}

Trace 3: f_{HIGH}

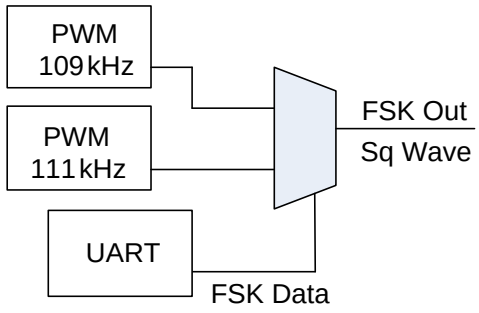
Trace 4: FSK Out



2 System Structure

Digital FSK signal generation can be done by simply using a pair of pulse width modulators (PWM) and a digital multiplexer. This can be done in Universal Digital Blocks (UDBs) in some PSoC MCUs. In PSoC MCUs without UDBs, the multiplexer can be implemented using the Smart I/O block. The Smart I/O version is shown here.

Figure 2. FSK Block Diagram



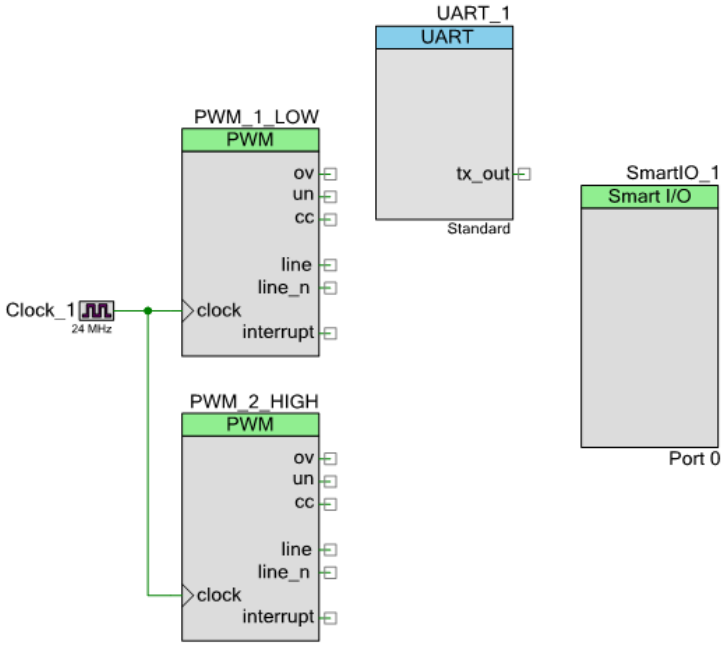
3 Basic Structure, Step-by-Step

This application note assumes that the user has some familiarity with PSoC MCUs and PSoC Creator, including starting projects, instantiating components in the design, writing C code, and programming the part. If this is not the case, review AN79953 – Getting Started with PSoC 4 or AN221774 – Getting Started with PSoC 6 MCU. Note that this application note only demonstrates building the design in PSoC Creator™, and not the new ModusToolbox for PSoC 6 MCU. The same principles can be applied in a ModusToolbox™-based design.

1. **Start Design**
   A. Start PSoC Creator.
   B. Select suitable part with Smart I/O, e.g., CY8C4A45xx.
   C. Create project.

2. **Place Blocks. See Figure 3.**
   A. Place PWM_1 and change its name to PWM_1_LOW.
   Configure for 109 kHz.
   In PWM **Basic** tab, set **Period** = 219, and set **Compare** = 109.
   B. Place PWM_2, and change its name to PWM_2_HIGH.
   Configure for 111 kHz.
   In PWM Basic tab, set **Period** = 215, and set **Compare** = 107.
   C. Place UART_1.
   In the **UART Basic** tab, set **Baud rate** to 2400.
   In the **UART Advanced** tab, scroll to bottom and check **Enable SmartIO support**.
   D. Place clock, and configure it for 24 MHz.

Figure 3. Block Placement



Now that the Components are placed, you can configure the Smart I/O. Inputs to come from several sources: TCPWMs, SCBs through data lines, and an array of signals present on the port where Smart I/O is located; in this case, Port 0.

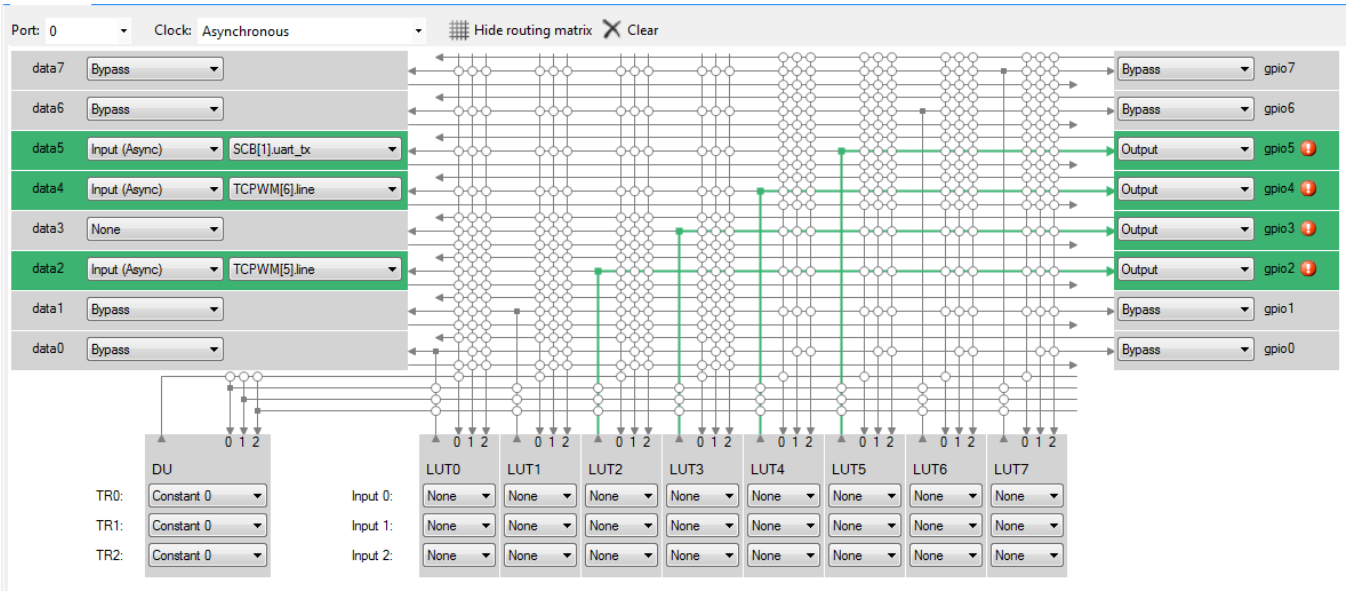
3. **Configure Smart I/O Inputs.**
   A. Right-click on **Smart I/O** and select **Configure**.
   B. Click **Data2** drop-down and click **Input(Async)**.
   C. Click input select pull-down and click **TCPWM(5).line**.
   D. Click **Data4** drop-down, click **Input(Async)**, click input select pull-down, and click **TCPMW(6).line**.
   E. Click **Data5** drop-down, click **Input(Async)**, click input select pull-down, and click **SCB(1).uart_tx**.
   F. Click Ok.

   PWM and UART signals are now shown enabled on the components.

   The output is the logical combination of the signals on the data lines, but it may be helpful to look at the raw signals from the PWMs and the UART. To do this, the data inputs are passed through LUTs to get to the output.

4. **Select output connections.**
   A. Right-click on **Smart I/O** and select **Configure**.
   B. Click the **gpio3** drop-down, select **Output**. This will be the FSK signal output.
   C. Click the **gpio2**, **gpio4**, and **gpio5** drop-downs. Set all to **Output**. These will follow the input signals.

Figure 4. Smart I/O Input and Output Placement



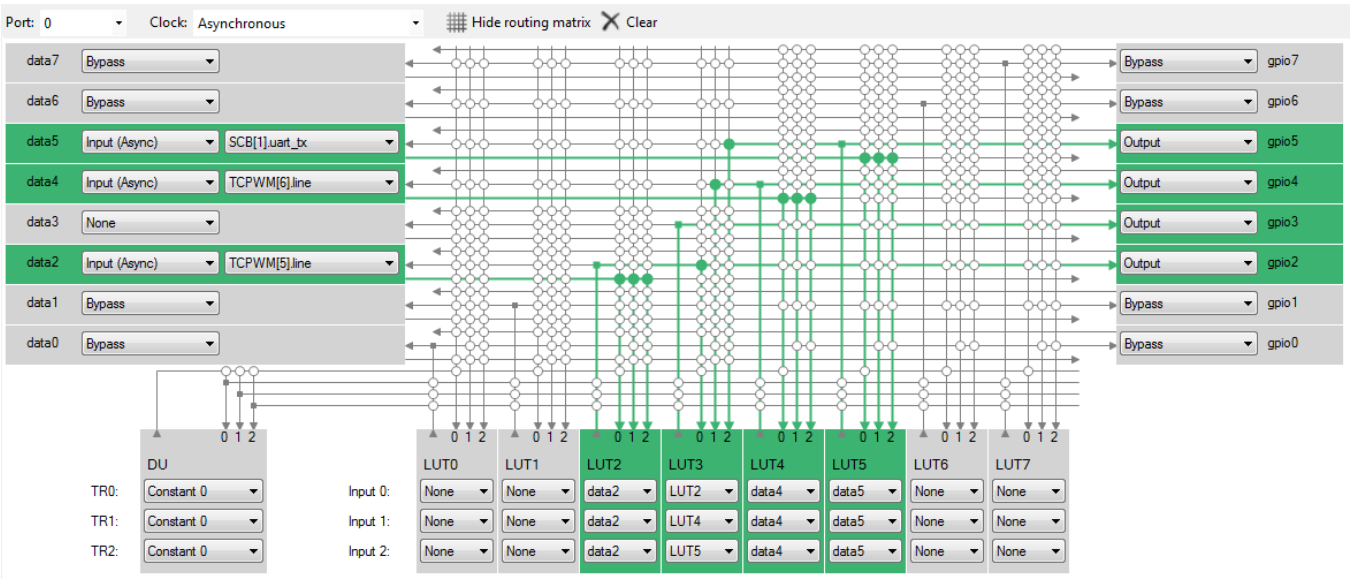
The outputs all display errors because the LUTs sourcing the outputs are not configured.

D. Click **Ok**.

5. **Configure pass-through LUT inputs.**
   A. In **LUT2 Input0, 1,** and **2** drop-downs, set all three LUT2 inputs to **data2**.
   B. In **LUT4 Input0**, **1,** and **2** drop-downs, set all three LUT4 inputs to **data4**.
   C. In **LUT5 Input0**, **1,** and **2** drop-downs, set all three LUT5 inputs to **data5**.
6. **Configure LUT3 input connections.**
   This selects inputs for the LUT-implemented multiplexer
   A. In the **LUT3 Input0** drop-down, set Input0 to **LUT2**.
   B. In the **LUT3 Input1** drop-down, set input1 to **LUT4**.
   C. In the **LUT3 Input2** drop-down, set input2 to **LUT5**.

   This Smart I/O routing is now complete.

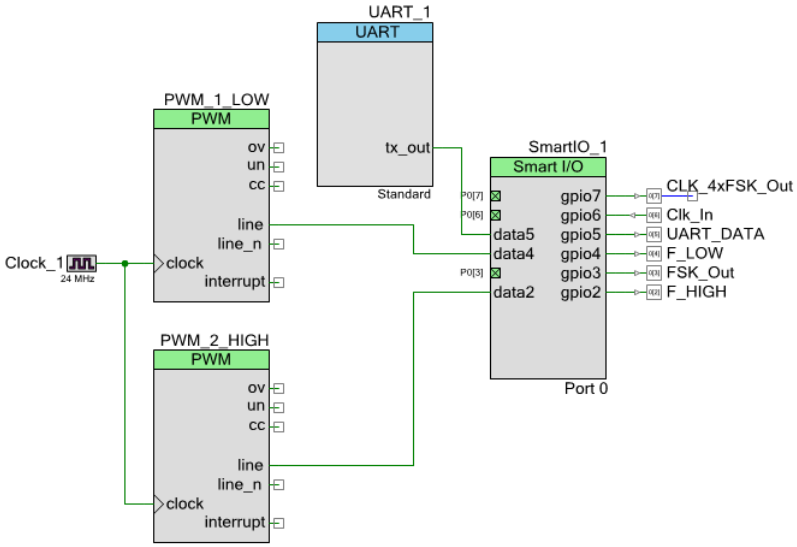
Figure 5. Completed Smart I/O Routing



7. **Add pin connections.**
   A. Route signal from PWM_1_LOW to SmartIO_1 data4 input.
   B. Route signal from PWM_2_HIGH to SmartIO_1 data2 input.
   C. Route signal from UART_1 tx_out to SmartIO_1 data5 input.
   D. Place digital output pins on Smart IO_1 P0[2], P0[3], P0[4], and P0[5]. Assign suitable signal names to Smart I/O outputs.

   The pins and block interconnects are now completely routed.

Figure 6. Blocks with Routed Signals



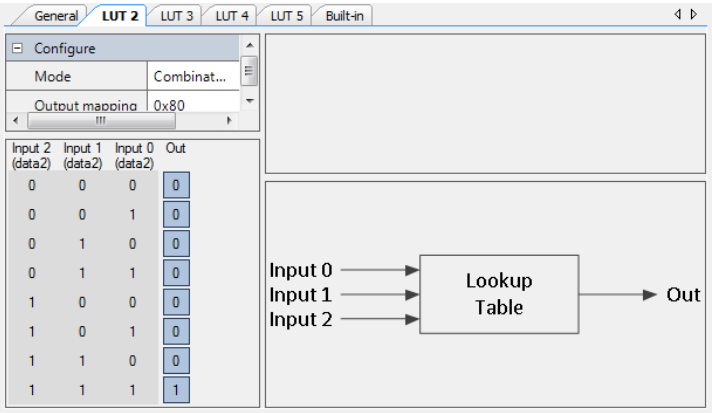
8. **Configure LUT2, LUT4, and LUT5 as followers.**
   A. Click **LUT 2** tab.

B. Click on each output state in **Out** column to change state as necessary.

Because the inputs of each of these LUTs are wired together, **Out** will go high when all three (Input 0, Input 1, and Input2) are high.

C. Click **Apply**.

Figure 7. Inverters in LUTs

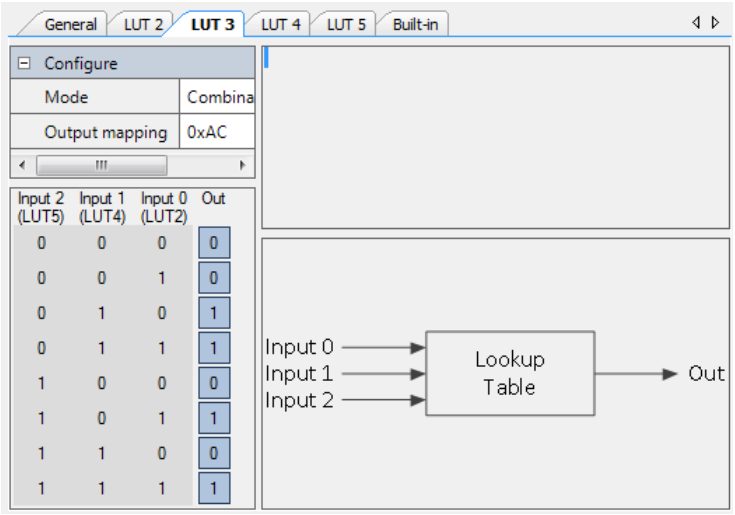


D. Repeat for LUT4 and LUT5.

9. Configure LUT3 as mux.

A. Click the **LUT3** tab.

Figure 8. Mux in LUT



When UART data (input2 on LUT3) is high, the output follows input 0 (fHIGH).
When UART data is low, the output follows input 1 (fLOW).
Thus when UART data is high, the output on LUT3 is the square wave from PWM_2_High.
When UART data is low, the output on LUT3 is the square wave from PWM_1_LOW.

B. Click on output state in the **Out** column to change state as necessary.

Configuration of Smart I/O is complete.

C. Click **Ok** to return to schematic view.

This arrangement using Smart I/O allows the FSK generation to be completely in hardware; it takes zero run-time code except start commands and consumes no interrupts.

10. Generate code.

All this takes is a few start commands. For demonstration purposes the UART is fed a continuous stream of 'AA:

Code 1

```
    PWM_1_LOW_Start();
    PWM_2_HIGH_Start();
    UART_1_Start();
    SmartIO_1_Start();
    while(1)
        UART_1_UartPutChar(0xAA);

    while(1);
```

11. Build and program.

Sync the scope on the UART data channel, P0.5. The image is best captured with single shot and normal trigger.

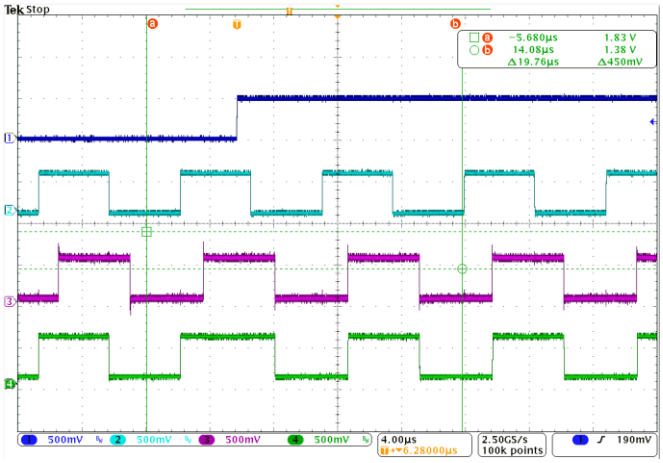
Figure 9. FSK Output

Trace 1: UART data

Trace 2: f_{LOW}

Trace 3: f_{HIGH}

Trace 4: FSK Out



FSK Out aligns with f_{LOW} when UART data is low; FSK Out aligns with f_{HIGH} when UART data is high.

Because the waveforms are asynchronous, there is no control over the phase relationship between the two output frequencies. This shows up as an abrupt phase discontinuity at the transition. This phase shift is randomly up to 180 degrees, which may cause a long period at the transition. This transition results in a signal that is not in the transmitted band.

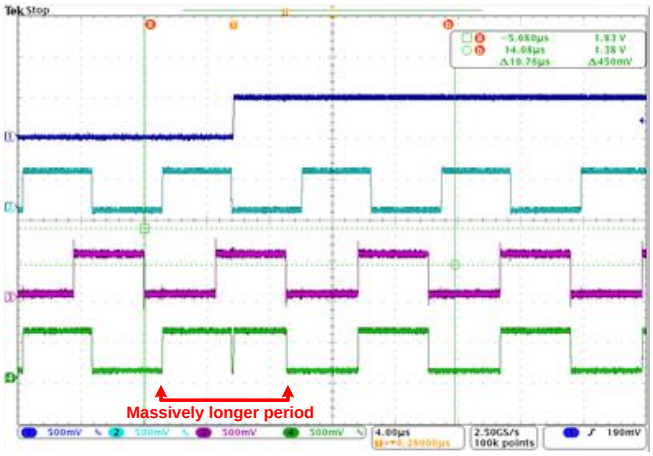
Figure 10. FSK with Phase Misalignment

Trace 1: UART data

Trace 2: f_{LOW}

Trace 3: f_{HIGH}

Trace 4: FSK Out

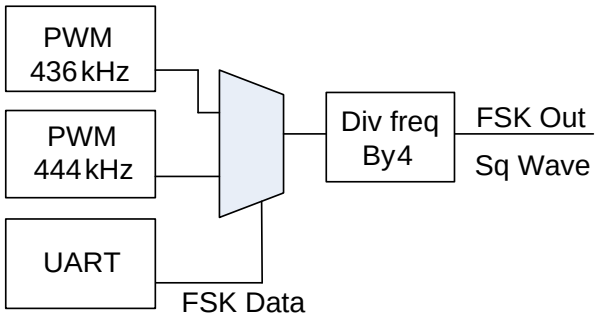


As a result, considerable energy is generated out of band; the in-band energy of the FSK waveform is substantially but randomly reduced. This ultimately causes a problem (delay) in detection of the FSK signal.

4 Improved Structure, Step-by-Step

There is a method for reducing the phase error. Reducing the phase error requires generating the FSKJ signal at a higher frequency, then dividing it down to the desired output frequency range. The phase error is reduced by the amount of the division. PWMs in PSoC 4 and PSoC 6 MCUs will accept clocks only from a defined clock source, not from another block or multiplexer output, so the post-multiplexer divide by 4 must be realized another way.

Figure 11. FSK Block Diagram for Reduced Phase Error



1. **Adjust PWMs for f_{LOW}=436 kHz (4*109 kHz) and f_{HIGH} = 444 kHz (4*111 kHz).**
   A. Set PMW_1_LOW Period = 54, PMW_1_LOW Compare = 27.
   B. Set PWM_2_HIGH Period = 53, PMW_2_HIGH Compare = 27.

   The 4xFSK waveform is not symmetrical, that is, not 50% duty cycle. This doesn't matter because it will be squared-up in the divider. We'll make some Smart I/O routing adjustments to enable the frequency divider.

   LUTs 2 and 4 are still the PWM signals, controlled by LUT5 (UART). These are logically combined in LUT7 instead of LUT3 (because we want the same pin to have the FSK output as in the earlier setup).

2. **Move multiplexer to LUT7.**
   A. Click **gpio7** drop-down, select **Output**.
   B. Click **gpio6** drop-down, select **Input**.
      LUT7 drives a GPIO which is externally connected to GPIO6.
   C. In **LUT7 Input0** drop-down, set Input0 to **LUT2**.

D. In **LUT7 Input1** drop-down, set input1 to **LUT4**.

E. In **LUT7 Input2** drop-down, set input2 to **LUT5**.

F. Click **Ok** to return to schematic view.

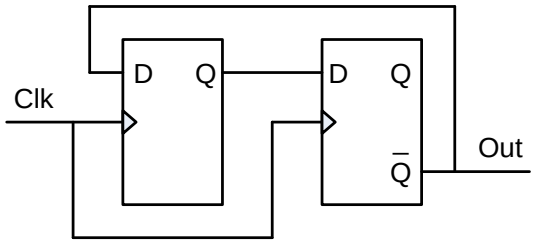
3. Route external clock.

A. Add digital output pin to gpio7

B. Add digital input pin to gpio6

The DU is configured as a ring counter, a pair of chained flip-flops. The $\overline{Q}$ output of the second flip-flop is routed back to the first flip-flop's input.

Figure 12. Divide by 4 Ring Counter



4. Configure Ring Counter in Smart I/O.

The DU's output is the Q of the flip-flop, there is no $\overline{Q}$ output. There is no mechanism in the DU to invert a signal, so LUT3 is used to generate the input to DU's D input.

The DU in Smart I/O must be connected before it can be configured

A. In Smart I/O General view, connect (click on the connect bubble) LUT3 output to DU TR2 input.
This enables changes to the DU settings.

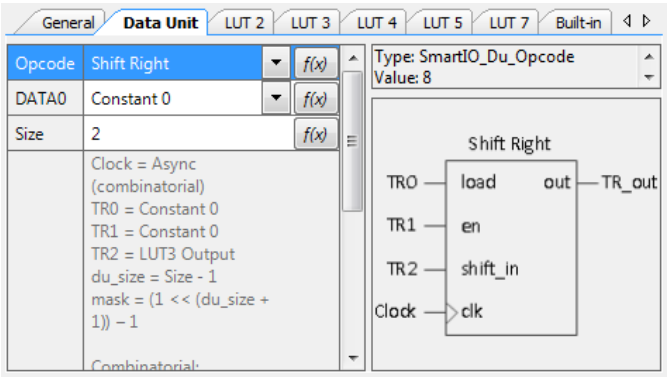
B. Click on DU's **TR0** drop-down, select **Constant 0**
This disables loading of the DU registers.

C. Click on the DU's **TR1** drop-down and select **Constant 1**
This enables DU clocked operation.

D. Click on the **Opcode** drop-down, select **Shift Right**.

E. Enter value for length = 2 to divide by 4.
(length = 3 yields divide by 6, length = 4 yields divide by 8)

Figure 13. Data Unit Configured as 8-bit Shift Register



5. Configure LUT3 as an inverter.

A. In **LUT3 input0** drop-down, set input0 to **DU**.

B. In **LUT3 input1** drop-down, set input1 to **LUT7** (this is a dummy connection).

C. In **LUT3 input2** drop-down, set input2 to **LUT7** (this is a dummy connection).

D. In **LUT3** drop-down, configure the output to be the complement of input0.

Figure 14. LUT3 Configured as Inverter on Data Unit Output

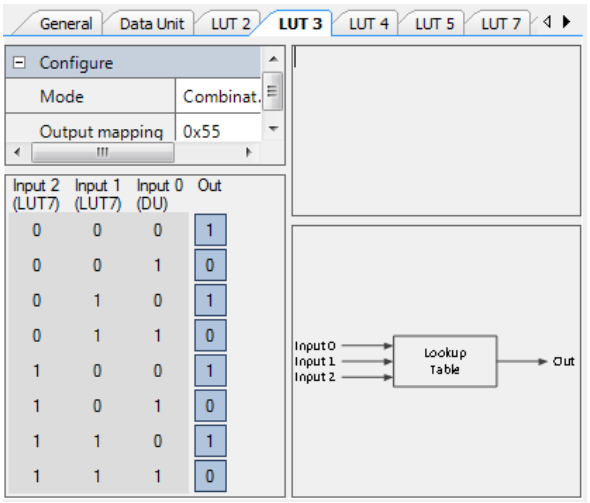
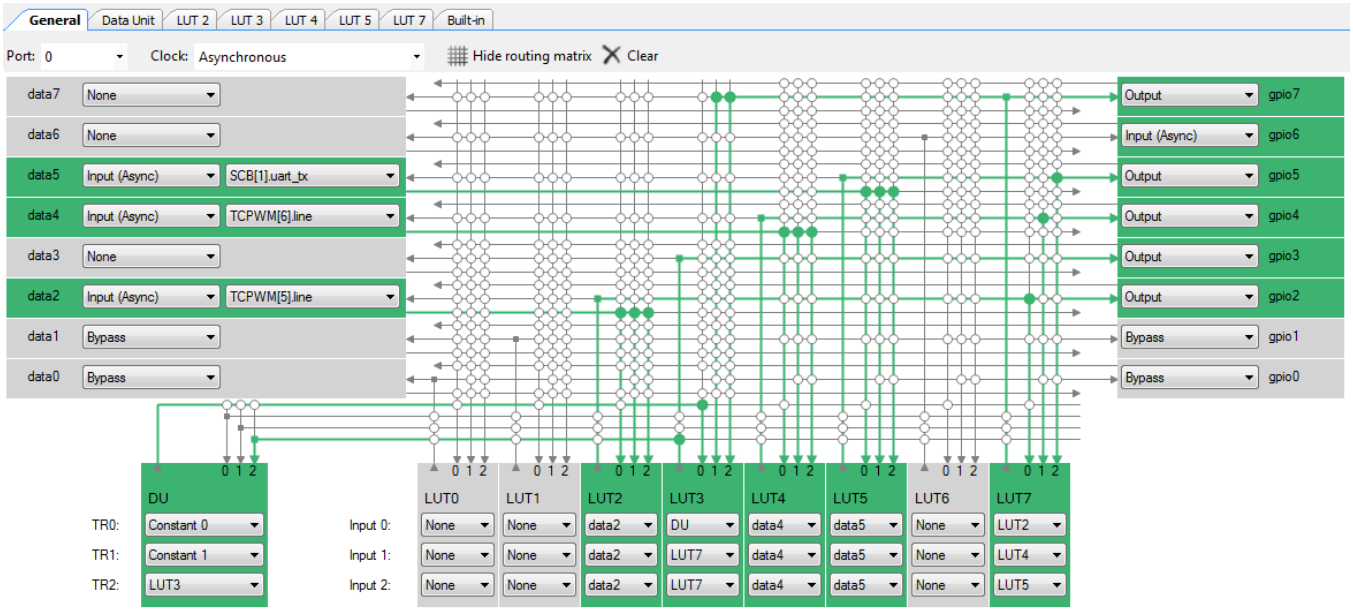


Figure 15. Smart I/O Routing for Clock Feedback



6. **Route Smart I/O clock.**

   The DU can be clocked from a long list of sources; the default is asynchronous. This design requires the clock for the divide by 4 in the DU to be clocked from an external signal. It would seem obvious to set the clock drop-down to gpio6, but that doesn't quite work.

   Here's where it gets interesting: The Smart I/O block goes through a several clock cycle start sequence after the issuance of a SmartIO_Start() command. The DU is to be clocked from GPIO6, but until the DU and the whole Smart I/O are running, there is no clock signal on GPIO6. So, the DU clock is set to be asynchronous as part of configuration. After a delay to allow for clock start, the Smart I/O clock is changed to GPIO6 at run-time. The SmartIO_ClockSelect() API stops the clock, changes the source, the restarts the clock. When the clock stops, the output on gpio6 stops and won't restart. Instead, the clock selection must be changed on the fly.

Add clock change code to project. The complete firmware is:

Code 2

```
#include "project.h"
uint8 clockSrc;
int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */
    /*  Initialization     */
    PWM_1_LOW_Start();
    PWM_2_HIGH_Start();
    UART_1_Start();
    SmartIO_1_Start();
    CyDelay(50);
    /* Re-route prgio clock from none to P0.7
       Modified from standard clock assignment to not
       disable Smart I/O during switch */
    clockSrc = 0x06;
    uint32 regVal;
    regVal = SmartIO_1_CTL & SmartIO_1_CLOCK_CLEAR;
    SmartIO_1_CTL = regVal | (((uint32)(clockSrc <<
    CYFLD_PRGIO_PRT_CLOCK_SRC__OFFSET)
                                        &
SmartIO_1_CLOCK_MASK));
    /* End Smart I/O clock re-assignment */
    /*Send continuous AA to UART */
    while(1)
        UART_1_UartPutChar(0xAA);
}
```

7. **Build project, program, and test.**
   The divide-by-4 clock function is clearly seen in the oscilloscope image. Single-stepping through oscilloscope triggers shows multiple images where the worst-case phase error at the bit transition is substantially reduced, compared to the earlier logic without the divide-by-4.

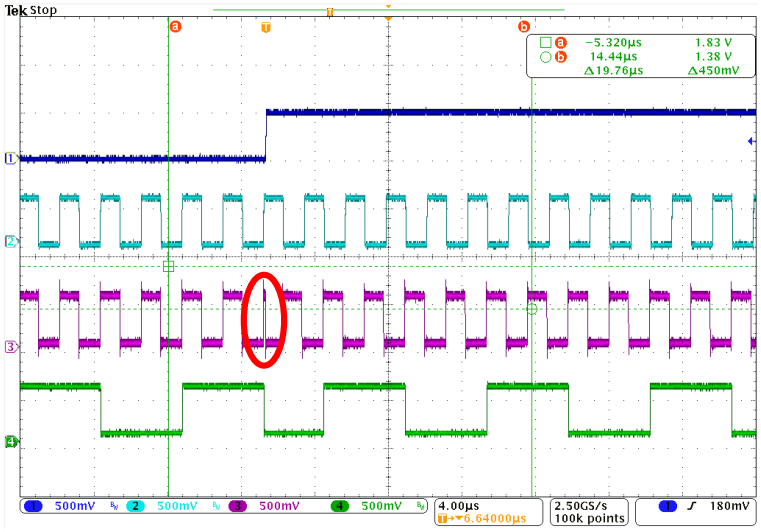
Figure 16. FSK Output with Reduced Phase Error

Trace 1: UART data

Trace 2: 4 * f_{Low}

Trace 3: 4 * FSK

Trace 4: FSK Out



The short pulse in the 4*FSK waveform at the bit transition creates a nearly 180-degree phase shift. By dividing this waveform's frequency by four, the phase shift in the FSK output is substantially reduced.

5 Summary

We have demonstrated a method for building a digital FSK transmitter with reduced phase distortion at the bit transition. This method has been successfully used in developing Power Line Communications (PLC) systems using Cypress PSoC Analog Coprocessor.

6 Related Documents

Application Notes	
None	
PSoC Creator Component Datasheets	
PWM	Supports fixed block and UDB-based PWMs
Smart I/O	Supports creation of logic functions in PSoC 4A-S and Analog Coprocessor
Pins	Supports connection of hardware resources to physical pins
Device Documentation	
PSoC 4000S Datasheet	PSoC 4100S Architecture TRM
PSoC 4100S Datasheet	PSoC 4100S Registers TRM
PSoC Analog Coprocessor Datasheet	PSoC Analog Coprocessor Architecture TRM
PSOC 4000S Architecture TRM	PSoC Analog Coprocessor Registers TRM
PSOC 4000S Registers TRM	
Development Kit (DVK) Documentation	
PSoC4100S Kit 145-40xx	PSoC 4100S Kit 041-41xx

About the Author

Name: Dennis Seguine
Title: Senior Member of Technical Staff
Background: BSEE, MSEE, University of California, Berkeley

Document History

Document Title: AN221886 - Using Smart I/O to Generate a Digital FSK Transmit Signal

Document Number: 002-21886

Revision	ECN	Submission Date	Description of Change
**	6044373	01/24/2018	New Application Note.
*A	6743983	12/05/2019	Minor text edits. Updated to template

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 MCU

Cypress Developer Community

Community | Code Examples | 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, 2018-2019. This document is the property of Cypress Semiconductor Corporation and its subsidiaries ("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. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress shall have no liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. CYPRESS DOES NOT REPRESENT, WARRANT, OR GUARANTEE THAT CYPRESS PRODUCTS, OR SYSTEMS CREATED USING CYPRESS PRODUCTS, WILL BE FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (collectively, "Security Breach"). Cypress disclaims any liability relating to any Security Breach, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from any Security Breach. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. 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. "High-Risk Device" means any device or system whose failure could cause personal injury, death, or property damage. Examples of High-Risk Devices are weapons, nuclear installations, surgical implants, and other medical devices. "Critical Component" means any component of a High-Risk Device whose failure to perform can be reasonably expected to cause, directly or indirectly, the failure of the High-Risk Device, 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 any use of a Cypress product as a Critical Component in a High-Risk Device. You shall indemnify and hold Cypress, its directors, officers, employees, agents, affiliates, distributors, and assigns harmless from and against all claims, costs, damages, and expenses, arising out of any claim, including claims for product liability, personal injury or death, or property damage arising from any use of a Cypress product as a Critical Component in a High-Risk Device. Cypress products are not intended or authorized for use as a Critical Component in any High-Risk Device except to the limited extent that (i) Cypress's published data sheet for the product explicitly states Cypress has qualified the product for use in a specific High-Risk Device, or (ii) Cypress has given you advance written authorization to use the product as a Critical Component in the specific High-Risk Device and you have signed a separate indemnification agreement.

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.