Author: Michael Alba
Date: March 09, 2017

Introduction

This application note will discuss a one-button Bluetooth keyboard that takes input in the form of Morse Code. Morse Code is a scheme for sending messages using only short and long intervals, known respectively as dots and dashes. In this way, users can type any alphanumeric character using only a single input. See Figure 1 for a Morse Code reference.

The GreenPAK SLG46537V will be used to translate the Morse Code input into a byte of data. This byte will then be transmitted using an external Bluetooth module (specifically, the **Rigado BMD-300**) to be displayed as the appropriate character on a simple computer application.

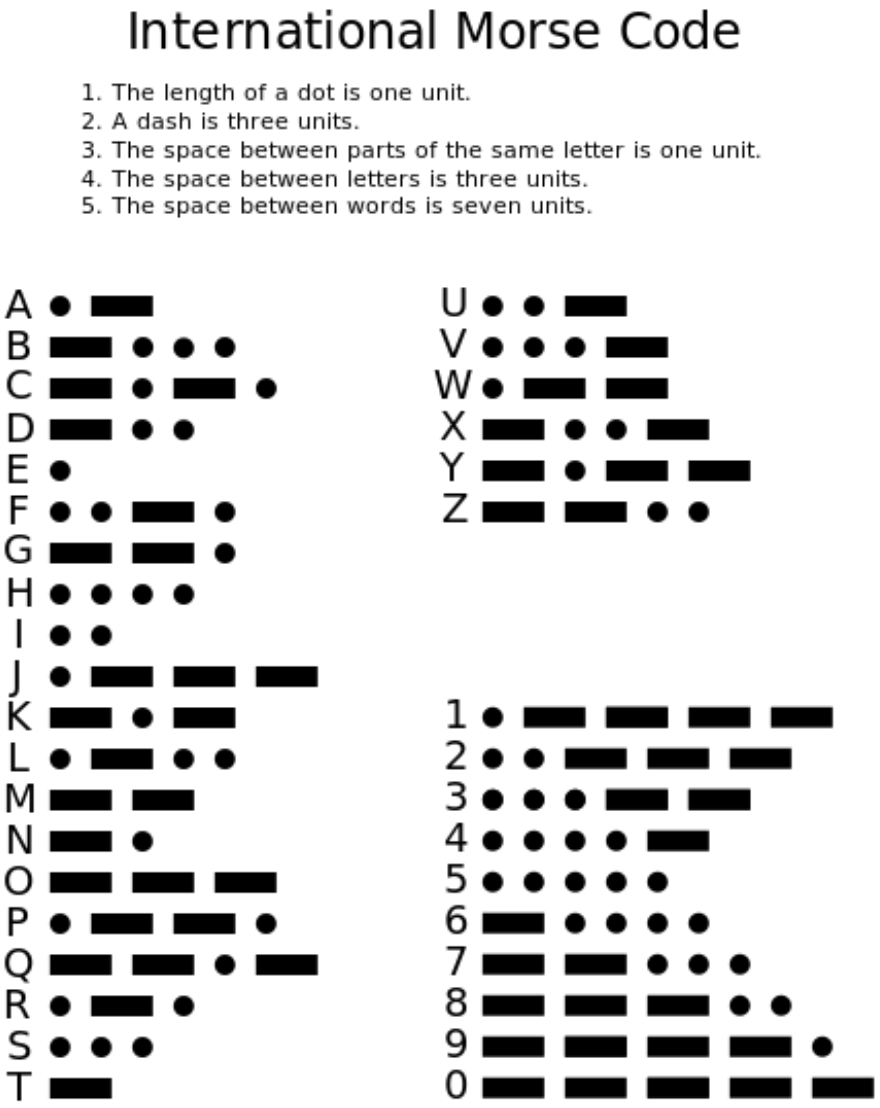

Figure 1. An overview of Morse Code

External Components

The external components required for this application include a button for Morse Code input and a Bluetooth module to send the GreenPAK's data to the computer application.

- **Button**: This application note uses an E-Switch KS-01Q-02 (**Digi-Key EG4792-ND**) through-hole pushbutton switch, though any momentary-on button would suffice.
- **Bluetooth**: The Bluetooth module chosen for this application is a Rigado BMD-300 (**Digi-Key 1604-1006-1-ND**). While an individual unit sells for USD$11.71 (at the time of writing), there are significant price breaks when ordering in bulk, making this module an attractive low-cost Bluetooth option.

For this application note, a BMD-300 Evaluation Board (**Digi-Key 1604-1007-ND**) was used in order to simplify prototyping.

Realization with GreenPAK Designer

The first step in configuring the GreenPAK to serve as a Morse Code keyboard is to devise a method for encoding Morse Code input (dots and dashes) into binary data. To generate a unique byte of data from each Morse Code input, we'll use the following scheme:

Using this scheme, we can create a complete conversion table (Table 1).

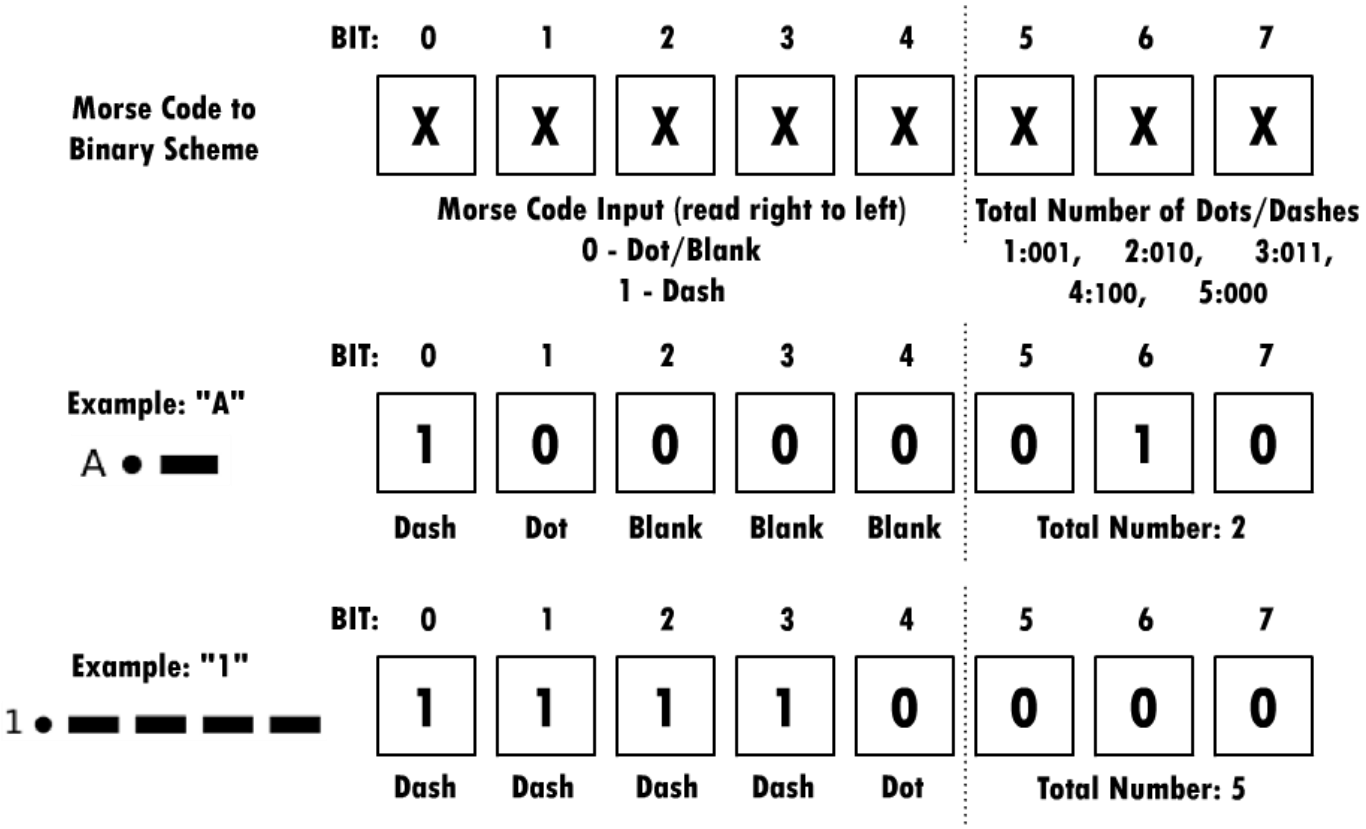

Figure 2. Conversion from Morse Code input to byte of alphanumeric data

Character	Byte	Character	Byte
A	10000010	S	00000011
B	00010100	T	10000001
C	01010100	U	10000011
D	00100011	V	10000100
E	00000001	W	11000011
F	01000100	X	10010100
G	01100011	Y	11010100
H	00000100	Z	00110100
I	00000010	0	11111000
J	11100100	1	11110000
K	10100011	2	11100000
L	00100100	3	11000000
M	11000010	4	10000000
N	01000010	5	00000000
O	11100011	6	00001000
P	01100100	7	00011000
Q	10110100	8	00111000
R	01000011	9	01111000

Table 1. Complete conversion table from Morse Code input to binary data

With this scheme in hand, we can go on to describe the function of the GreenPAK with a state diagram.

Figure 3 displays this diagram as configured in GreenPAK Designer's Asynchronous State Machine (ASM) Editor.

Figure 4 gives an overview of the connections configured in GreenPAK Designer, and highlights the various functional groups used in this design: Counters, Registers, Button Input, ASM, and Data Output. We'll cover each of these groups in turn.

Asynchronous State Machine

We'll begin by describing the ASM in further detail. There are eight states available in GreenPAK's ASM, each of which has eight outputs and can take in as many as three inputs. The inputs control the state transitions, and the outputs will control the actions to be taken in each state. Both the inputs and outputs can be configured in the ASM Editor.

Figure 5 shows the states, state transitions, and state outputs in the main GreenPAK Designer workspace. This allows us to visualize both when a state transition will occur (when an input goes high), and which state the ASM will take next (the state connected to the input that has gone high). We can also visualize to which blocks the outputs are connected, which correspond to the actions that will be taken in each state. Table 2 gives a description of what actions should occur in each state.

The state outputs can easily be configured using the ASM's RAM table in the ASM Editor (shown in Figure 6). The RAM table has been configured such that five of the eight possible outputs control the necessary actions in each state. These outputs are described in Table 3 and visualized in Figure 6.

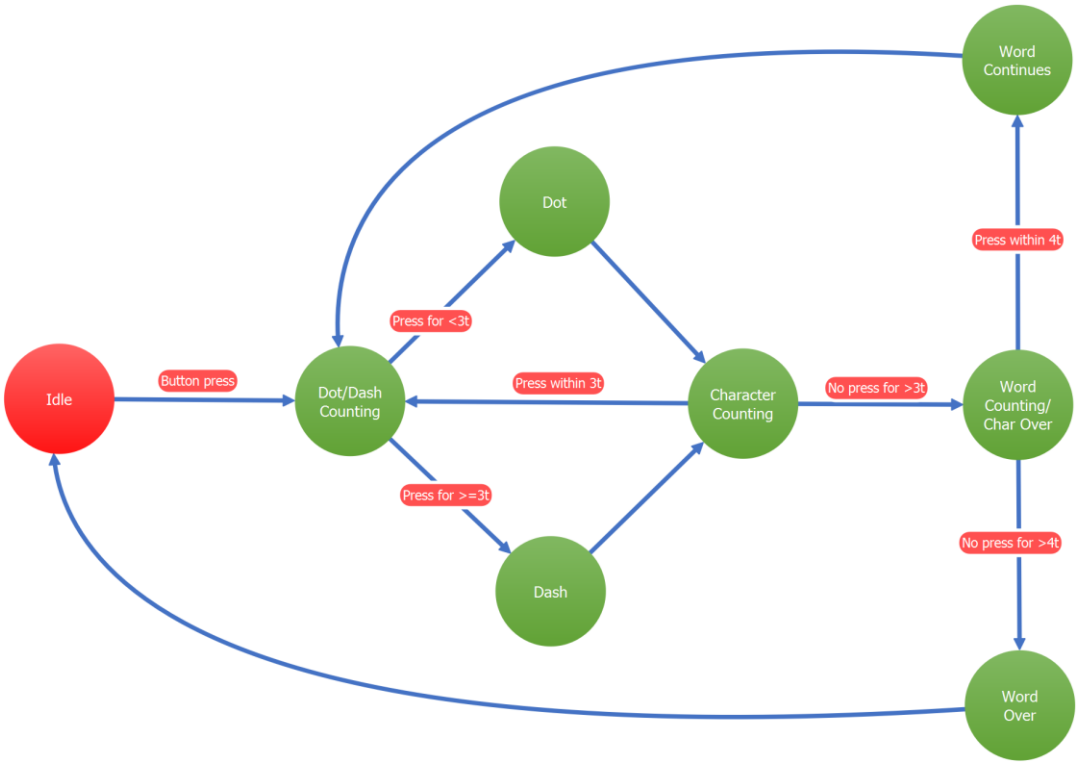

Figure 3. State diagram of Morse Code input conversion. The variable t represents one unit of Morse Code time (see Morse Code reference in Figure 1). Some alterations have been made with regards to timing to simplify the Morse Code input

State	Action(s)
Idle	N/A
Dot/Dash Counting	1. Start timer
Dot	1. Record dot to register 2. Increment dot/dash counter
Dash	1. Record dash to register 2. Increment dot/dash counter
Character Counting	1. Start timer
Word Counting/ Character Over	1. Start timer 2. Send register data to Bluetooth module
Word Continues	1. Reset registers
Word Over	1. Reset registers

Table 1. A description of what actions should take place in each state. These actions are controlled via the state outputs

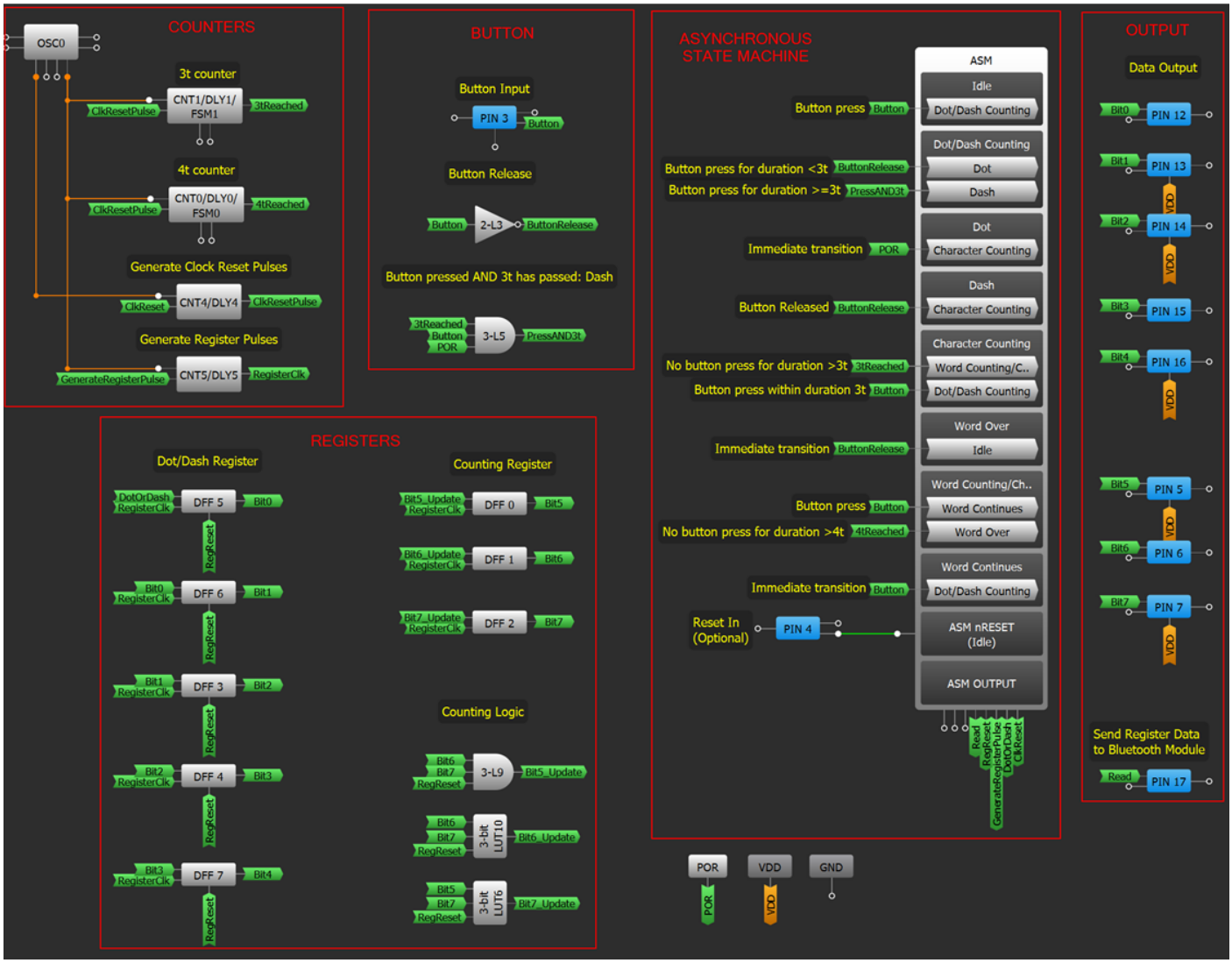

Figure 4. GreenPAK Designer overview highlighting the functional groups used in this project

Output Name	Description
ClkReset	The output necessary for starting the timer. When this value switches from HIGH to LOW, or vice versa, a pulse is generated that restarts the counters.
DotOrDash	Writes a 0 or 1 based on the Dot or Dash state, respectively.
RegisterClk	When this value switches from LOW to HIGH, a clock pulse is generated to advance the registers. The output connection is labelled "GenerateRegisterPulse" in the workspace.
RegisterReset	When LOW, resets the DFFs in the registers. The output connection is labelled "RegReset" in the workspace.
ReadRegisterData	When HIGH, prompts the Bluetooth module to read the register data. The output connection is labelled "Read" in the workspace.

Table 3. Descriptions of the ASM Connection Matrix Outputs

Figure 5. State transition signals of the ASM. The ASM nRESET input returns the ASM to the Idle state when it goes LOW. For testing purposes, this is controlled with an external button

State name	Connection Matrix Output RAM							
	StateDisplay0	StateDisplay1	StateDisplay2	ReadRegiste...	RegisterReset	RegisterClk	DotOrDash	ClkReset
Idle	0	0	0	0	1	0	0	0
Dot/Dash ..	0	0	1	0	1	0	0	1
Dot	0	1	0	0	1	1	0	1
Dash	0	1	1	0	1	1	1	1
Character ..	1	0	0	0	1	0	0	0
Word Over	1	0	1	0	0	1	0	0
Word Count..	1	1	0	1	1	0	0	1
Word Conti..	1	1	1	0	0	1	0	0

Figure 6. Connection Matrix Output RAM. StateDisplay0, StateDisplay1, and StateDisplay2 represent the state number, and have been connected to LEDs for testing purposes

Counters

Now we'll discuss the purpose and configuration of the blocks shown in Figure 7. First, to time the units of Morse Code, we'll use the GreenPAK's counter blocks in counter mode. Based on the state diagram shown in Figure 3, we'll need one counter to time three units and another to time four. The configuration of the 3t counter is shown in Figure 8.

Two additional counter blocks are used in this application, both of which are configured to generate a one-shot pulse. This pulse serves as either a reset for the previous counter blocks, or as a way to advance the registers. Both one-shot counters are controlled via the ASM outputs described in Table 3.

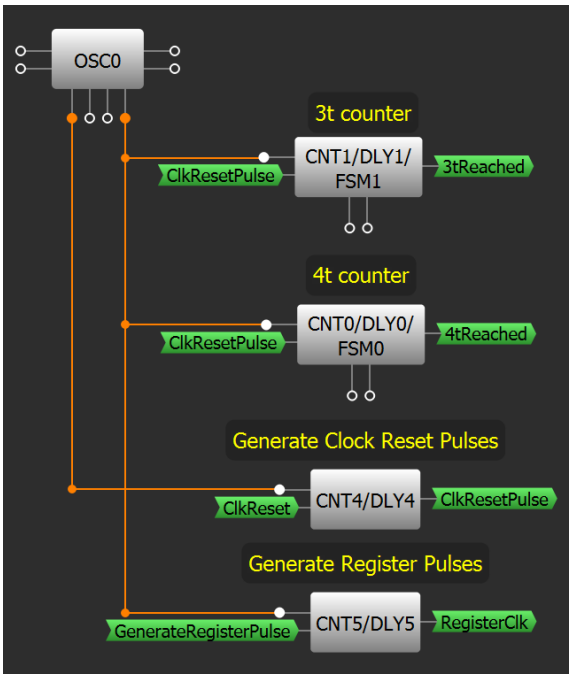

Figure 7. Four Counter blocks are used, two for timing and two for generating one-shot pulses

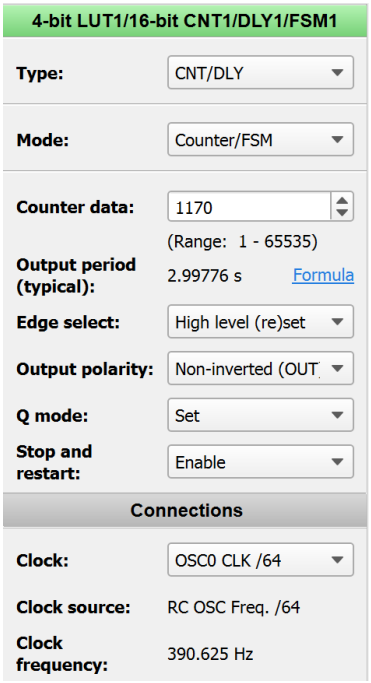

Figure 8. Configuration of the 3t counter, where t is taken to be one second. The 4t counter is configured in an identical manner, with a Counter Data value of 1561 (four seconds)

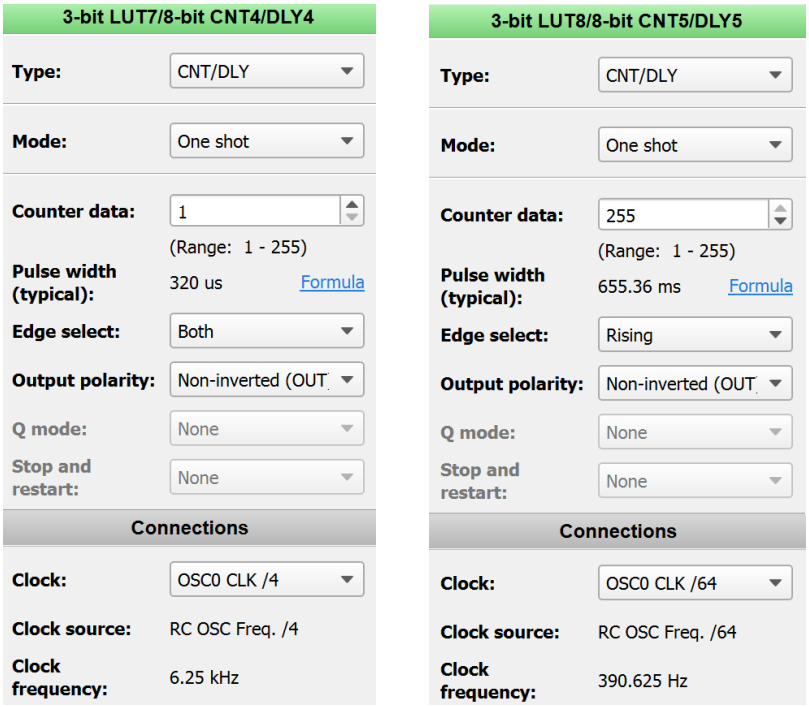

Figure 9. The two counter blocks in one-shot mode, as configured to reset the timers (left) and advance the registers (right)

Registers

To keep track of the dots and dashes input by the user, we'll create two separate registers using D Flip Flops (DFFs). The first register will consist of five DFFs and will record the sequence of user input. The second register will consist of three DFFs and will count the number of user inputs. Taken together, the registers will actively create the byte of Morse Code data as described in Figure 2.

The Dot/Dash register is comprised of five DFFs. The DotOrDash input to DFF 5 is set with the ASM output. The ASM outputs also control the RegReset (directly) and RegisterClk (indirectly, through the one-shot counter) inputs.

To create the Counting Register, three Look-Up Tables (LUTs) are used to set the next binary value to be recorded.

Figure 10. One register is used to record the user input, and the other is used to keep track of the total number of dots and dashes

3-bit LUT9/8-bit CNT6/DLY6

Type: LUT

IN3	IN2	IN1	IN0	OUT
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

3-bit LUT10/Pipe Delay

Type: LUT

IN3	IN2	IN1	IN0	OUT
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

3-bit LUT6/8-bit CNT3/DLY3

Type: LUT

IN3	IN2	IN1	IN0	OUT
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Figure 11. LUT configurations for the counting logic. IN0 corresponds to RegReset. The LUTs are shown in order from Most Significant Bit (MSB) to Least Significant Bit (LSB), and cycle through the binary values 000, 001, 010, 011, and 100 as long as RegReset is HIGH. When RegReset goes LOW, the output is 000

In this way, each time RegisterClk advances the registers, the appropriate value is stored in the DFFs.

Furthermore, since we can design each LUT to require only two inputs, the third input can be set as RegReset – this lets us work around the fact that only the five DFFs used in the Dot/Dash Register have a direct reset input. The LUT configurations are shown in Figure 11.

Buttons

We'll use PIN 3 as a digital input to record the user's button presses. In addition, we'll use another LUT (configured as an inverter) as a way to determine when the button has been released. One final LUT will be configured as an AND gate to determine when both the button is pressed, and the 3t timer has been reached.

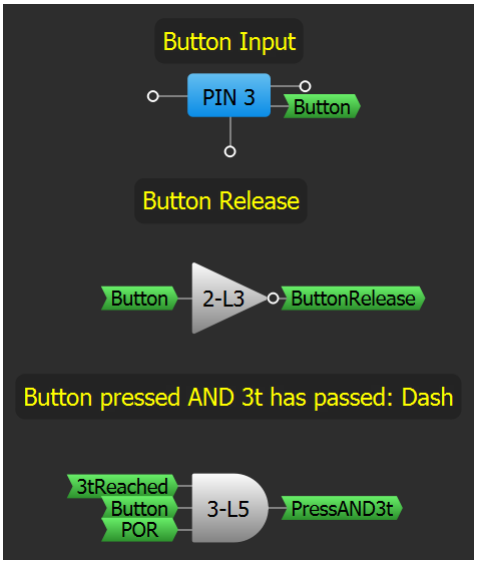

Figure 12. PIN 3 records the Morse Code input, and two LUTs pass additional information to the ASM

Output

The register data is passed directly to the Bluetooth module via output Pins 12-16 (dots/dashes) and Pins 5-7 (total number of dots/dashes). This data is only passed if the Read signal on PIN 17 is HIGH.

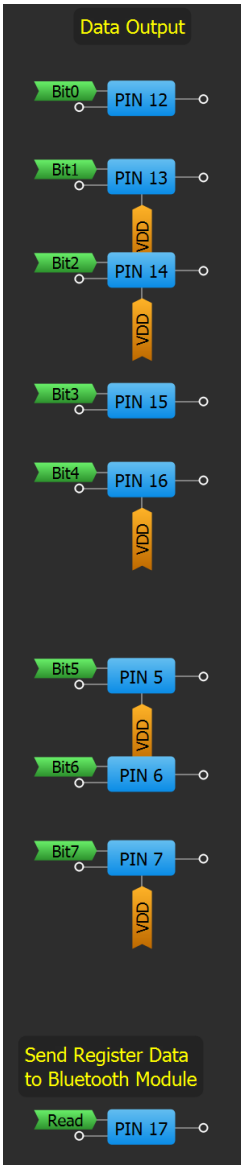

Figure 13. PINS 5-7 and 12-17 are used to pass the register data to the Bluetooth module

Example Implementation

We can easily prototype this project using a breadboard, the GreenPAK Development Board, and the Bluetooth module's evaluation board (discussed above in External Components). The prototype uses one button for Morse Code input and another as a reset to the ASM (optional). These inputs are taken from the breadboard to the GreenPAK Development Board, with the outputs taken from the GreenPAK Development Board to the Bluetooth module's GPIO pins.

With this setup, the GreenPAK Designer emulation window should be configured as shown in Figure 15.

The next step is to configure the Bluetooth module to send the Morse Code data. The full steps of this process are beyond the scope of this application note; however, full documentation (see **here** and **here**) is provided for the BMD-300 Bluetooth module should readers choose to use it if building this project.

We can test the prototype by using the input button to type a character of Morse Code. Figure 17 shows the result of this test, using a simple computer program (source code attached) to look up the relationship between the byte of data and the appropriate character.

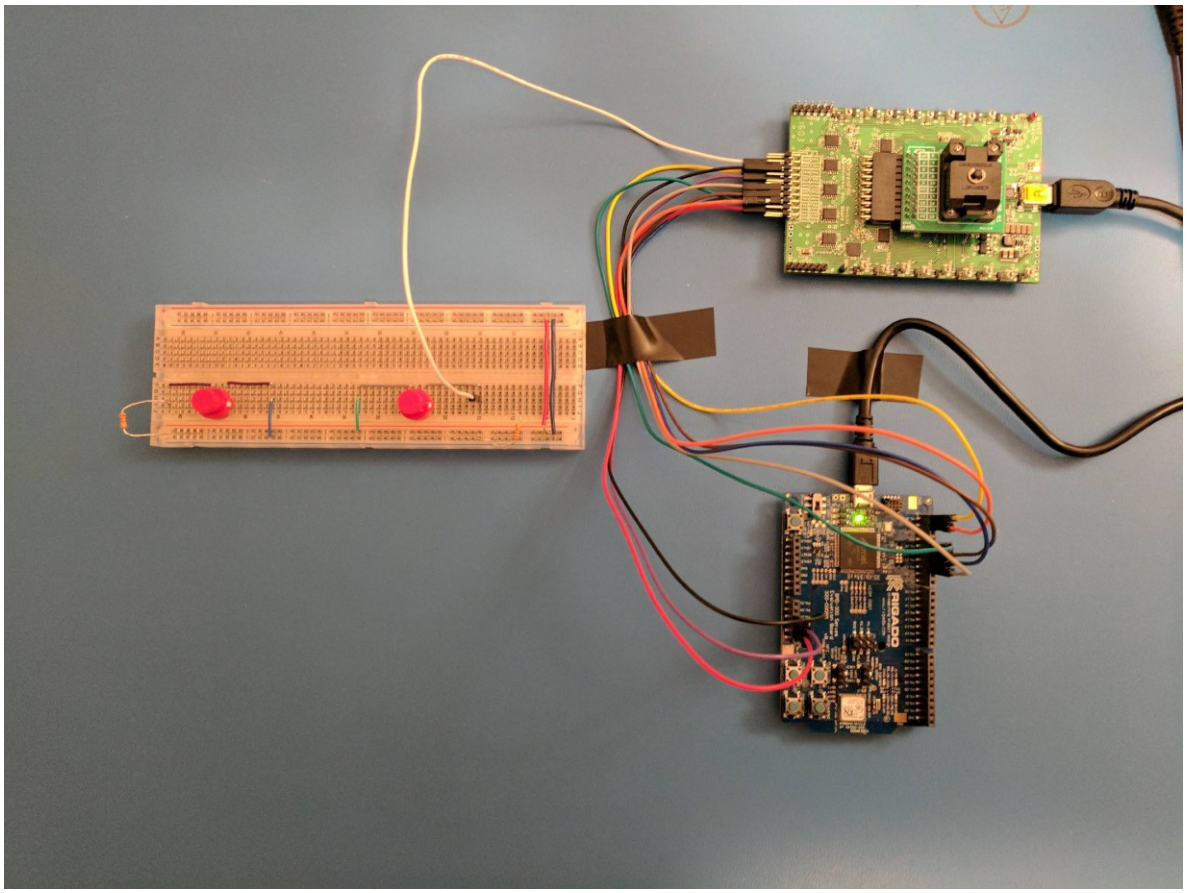

Figure 14. Picture of the wired connections between the breadboard input button, the GreenPAK Development Board (green), and the Bluetooth module's evaluation board (blue). The left button on the breadboard functions as the ASM reset and has been disconnected.

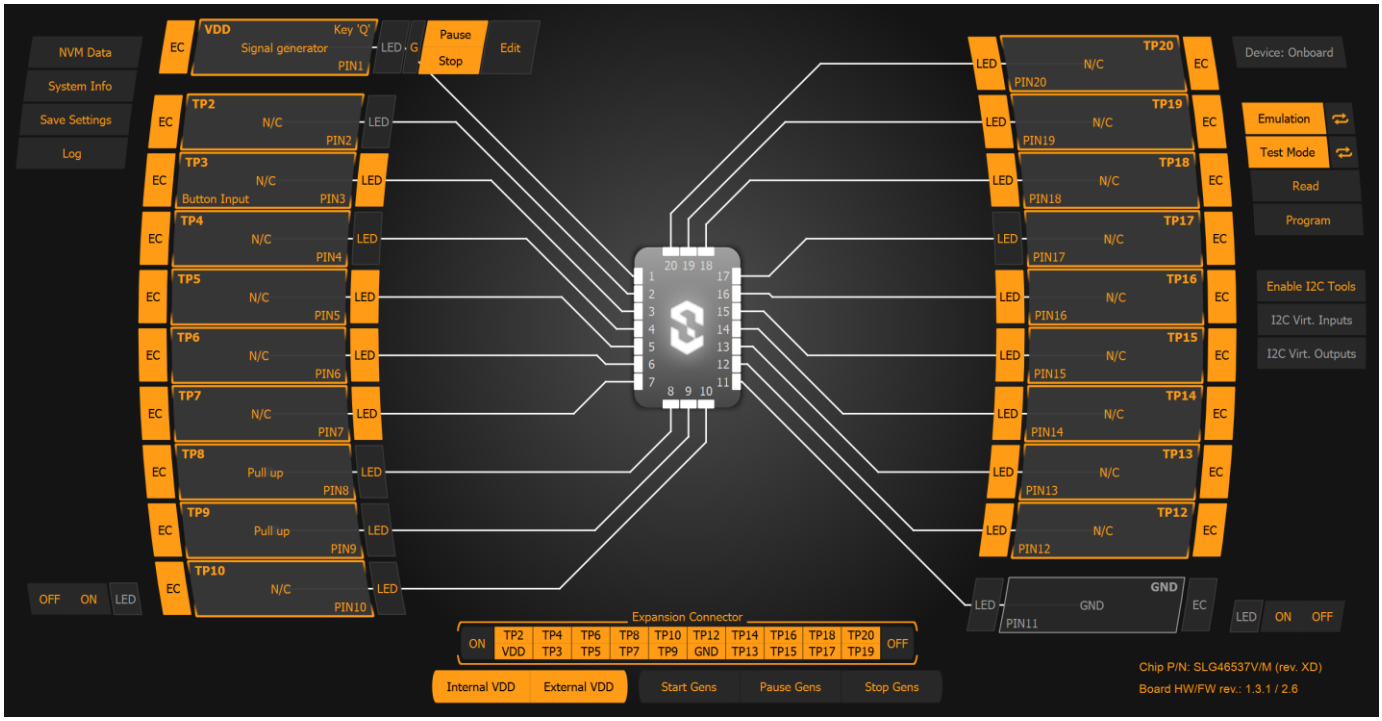

Figure 15. The emulation window in GreenPAK Designer. Note the enabling of the Expansion Connector.

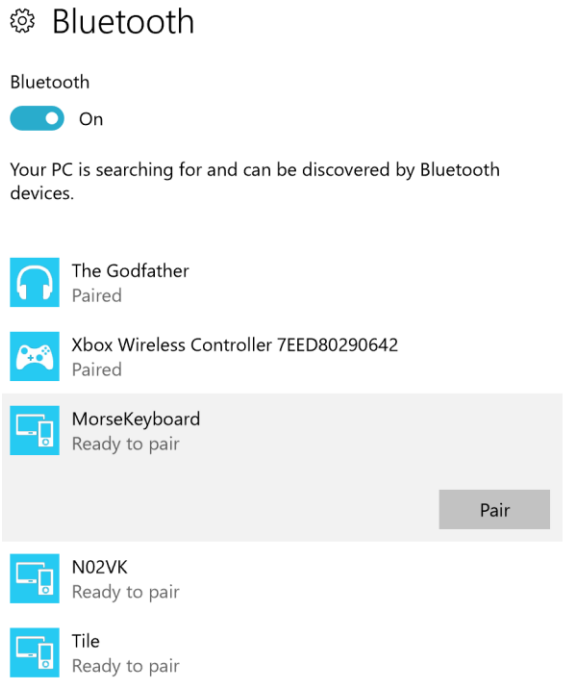

Figure 16. A screenshot of available Bluetooth devices on Windows 10, advertising the device named "MorseKeyboard"

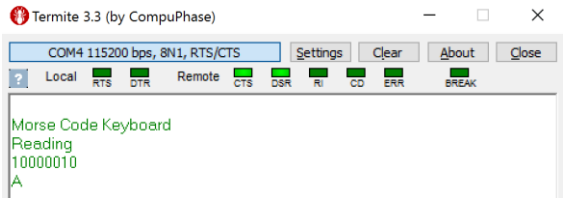

Figure 17. A dot followed by a dash creates the byte 10000010, which corresponds to A. This computer program converts the data into the correct alphanumeric character

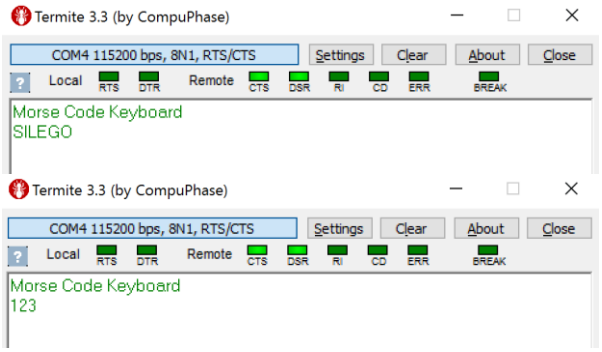

Figure 18. The prototype was used to type in the word "Silego" and the number "123"

In the same way, we can test any of the other characters given in Table 1. Now, the computer program has been altered to allow for continuous input of text (it will not display the binary data).

By using an external power source, we can test the current consumption of the prototype. The current consumption varies depending on its state. Its range is given in Table 4:

State of Prototype	Current
ASM in Idle State	36 µA
Button is pressed	3.3 mA

Table 4. Current consumption of Morse Code Keyboard prototype

Extensions

This keyboard has a number of potential applications, such as an alternative to Smart TV text input (instead of the clumsy navigation of on-screen keyboards), or as a tool for handicapped keyboard input (due to its extreme simplicity). Both possibilities can make use of the same hardware while using the data in different ways. However, both could also extend the hardware to make use of an augmented or revised Morse code.

Conclusion

In this application note, we built a low-cost, wireless keyboard that takes user input in the form of Morse Code. We used the GreenPAK SLG46537V IC to convert the user input into a customized binary format, and then transmitted this data via a Bluetooth module for translation with a simple computer program. Hopefully, this project has demonstrated the versatility and ease of GreenPAK design.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.