AN-1134 Drawing Characters on an Android App and Displaying it on LED Matrix

This Application Note describes the circuit used to go from drawing a character on an Android App, sending it to an Arduino UNO via Bluetooth, then to a GreenPAK™ via I2C which fits data to display the character on a LED matrix. Using the GreenPAK5 frees up the necessary Arduino GPIO's so that Bluetooth can be used, along with leaving GPIO's available for other purposes. Additional benefits of GreenPAK include the I2C protocol needing just 2 wires to control, for up to 16 GreenPAK5 from one I2C master. So if we want to connect more than one LED matrix to the Arduino UNO, using multiple GreenPAK's is a much more economical design. And once the GreenPAK is loaded with a character, it can work independently even if the I2C connection is disconnected.

Project Design Approach

This project uses an android app., and the app uses a virtual LED matrix. Therefore, you can update the character that is seen on the LED matrix from the mobile device directly.

This project consists of three stages:

1. Build the Android application
2. GreenPAK design
3. Arduino code

Figure 1. Components used in this App Note

Working principle of the LED matrix

LED dot matrices are very popular as they are very visible in a variety of ambient conditions.

In a dot matrix display, multiple LEDs are wired together in rows and columns in order to minimize the number of pins required. For example, an 8×8 matrix of LEDs (Figure 2) has the cathodes together in rows (R1 through R8) and anodes in columns (C1 through C8). Each LED is addressed by its row and column number. In this example we are using what is referred to as a Common Row Cathodes LED Matrix.

To immuminate a LED pixel in the matrix, a high signal is applied on the anode (column) and low signal is sent to the cathode (row). When we want to show characters or symbols, we typically need to illuminate numerous pixels. In this case, we divide the picture into sections, and we illuminate every row in a fast loop separately.

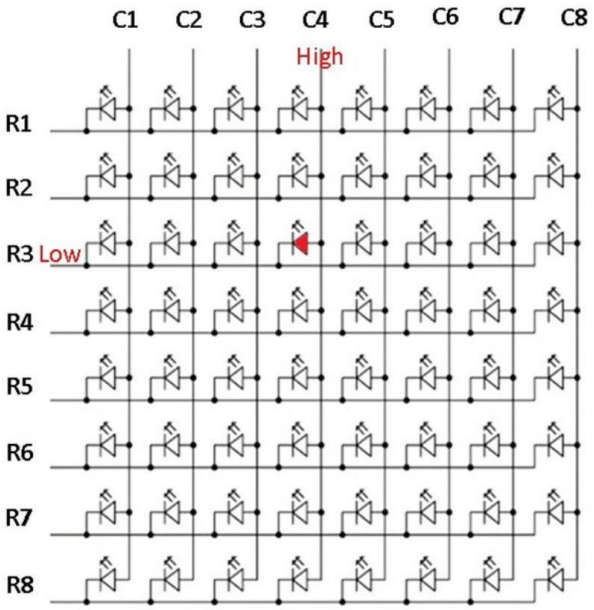

Figure 2. LED matrix diagram

The human eye can detect LEDs blinking at low frequency, but at more than 20Hz, the full character will appear and with less flicker. In this design, we will control an 8x7 LED matrix, as the last column C8 will be omitted because of a limitation of IO pins.

Android Application

For this first stage, an application for android devices is built to send data to the Bluetooth module. The app will have the graphical interface to the user.

The app interface has 56 buttons. Each button represents a pixel in the LED matrix, hence, each one will be represented by a binary variable with two states (on=1, off=0).

Every LEDs' row is represented by one byte (bit for pixel). To find the byte's value, we multiply every button state by the significant bit and then we take the sum of products.

p1..p64 are binary variables hold buttons states (**on =1, off=0**).

B1= (P1 X 1)+(P2 X 2)+(P3 X 4)+(P4 X 8)+(P5 X16)+(P6 X 32)+(P7 X 64)+(0)

B8= (P57 X 1)+(P58 X 2)+(P59 X 4)+(P60 X 8)+(P61 X16)+(P62 X 32)+(P63 X 64)+(0)

After this calculation, the character will be represented by 8 bytes (**B1..B8**) which will be sent to the Bluetooth module.

Bit Significant	x 1	x 2	x 4	x 8	x 16	x 32	x 64	x0	
Column / Row	C1	C2	C3	C4	C5	C6	C7	C8	Row BYTE
R1	P 1	P2	P3	P4	P5	P6	P7	0	B1
R2	P9	P10	P11	P12	P13	P14	P15	0	B2
R3	P17	P18	P19	P20	P21	P22	P23	0	B3
R4	P25	P26	P27	P28	P29	P30	P31	0	B4
R5	P33	P34	P35	P36	P37	P38	P39	0	B5
R6	P41	P42	P43	P44	P45	P46	P47	0	B6
R7	P49	P50	P51	P52	P53	P54	P55	0	B7
R8	P57	P58	P59	P60	P61	P62	P63	0	B8

Table 1. Control bits for LED matrix

MIT APP Inventor

The app can be made using the MIT App Inventor with ease and no prior programming experience is required. The app Inventor lets you develop applications for Android phones using a web browser with programming blocks.

To create the Android Application, a new project has to be started and the visible components need to be removed from the designer screen. Then 58 buttons need to be created; 56 buttons will be for the LED matrix, 1 will be for the list pickers for the Bluetooth devices (connect), and the last button will be used to send data via Bluetooth (print). We also need a Bluetooth client. Figure 3 below is a screen capture of our Android Application's user interface.

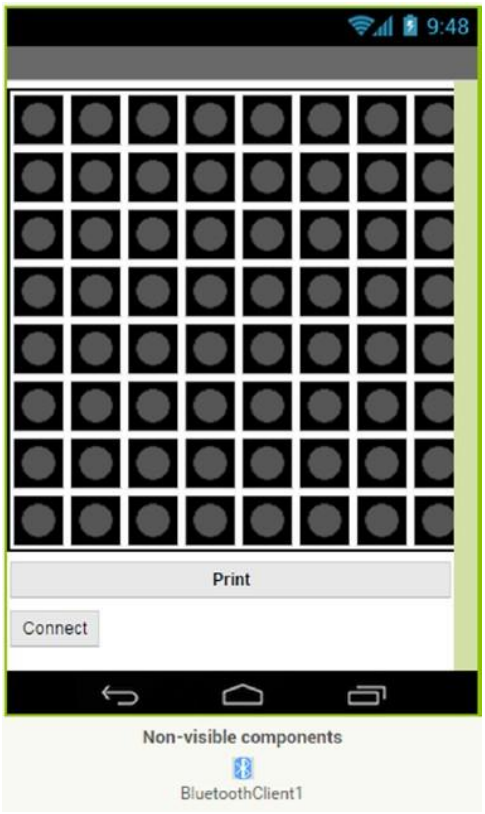

Figure 3. Android App GUI

To start programming, the "Block" button needs to be clicked. By dragging and dropping, we can add components from the bar on the left side. As in Fig 4, the global variables will be added and on-click functions for each button have to be built which will be used to change colors and/or to save the button state (on , off).

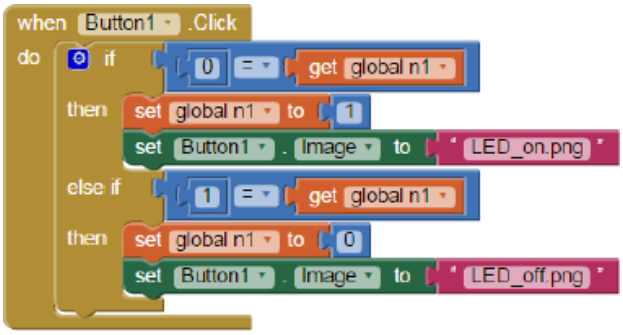

Figure 4. Block diagram for first matrix pixel

For different buttons, the process can be repeated by changing the button number and the global variable each time. Once done, the mathematical equation needs to be built which will gather every row's bits in one byte (B1..B8 equations). The 56 button states will be formed in 8 bytes variables. (Fig. 5 illustration).

Figure 5. Gathering bits in rows bytes

Finally, the app will send the 8 bytes using Bluetooth as a list of bytes when the print button is clicked (Fig. 6).

Figure 6. Send row bytes via Bluetooth

GreenPAK design

The Android application sends data to the Arduino, and then from Arduino to GreenPAK via I2C connection. Those details are explained in the "Arduino code" section later. The I2C interface in SLG46537V is very powerful because it can read and write all its configuration bits (including output states and ASM RAM). The I2C write command begins with a start bit, followed by a control byte, word address byte, data byte and stop bit (illustrated in Fig 7).

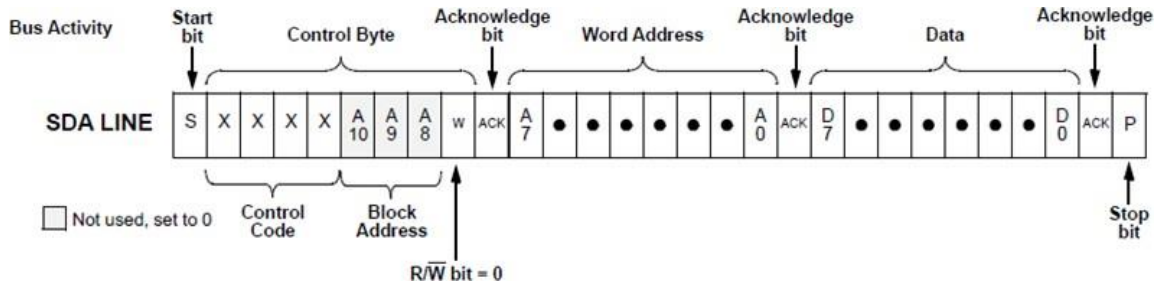

Figure 7. I2C Writing Frame

In GreenPAK we start designing the I2C block. This block is enabled from the properties bar and then the control code is chosen. Then two wires will be connected with pins 8, 9(SCL, SDA) and from the control code list, the device address is chosen (a number from 0 to 15). In our project 0 (0b0000) is selected. This is shown in Fig 8.

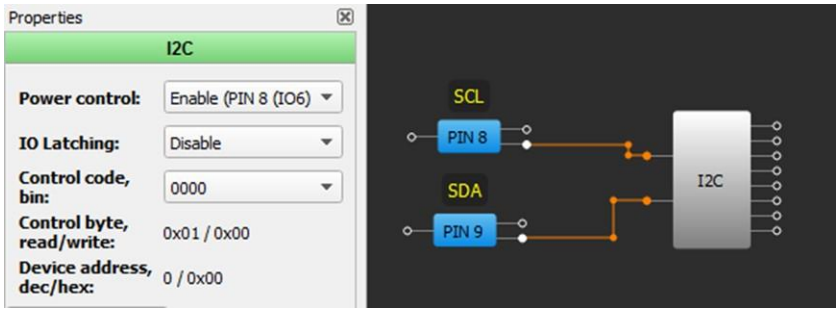

Figure 8. I2C block connection

A controller that will show bytes in rows needs to be made. For that, a State Machine block (ASM) has to be used. The ASM block is an 8-state asynchronous state machine. There are 24 state transition inputs, 1 nRESET input and 8 output lines. The ASM block is defined using state transitions and state outputs. Every state will represent one row and it will show the byte that is received from Arduino.

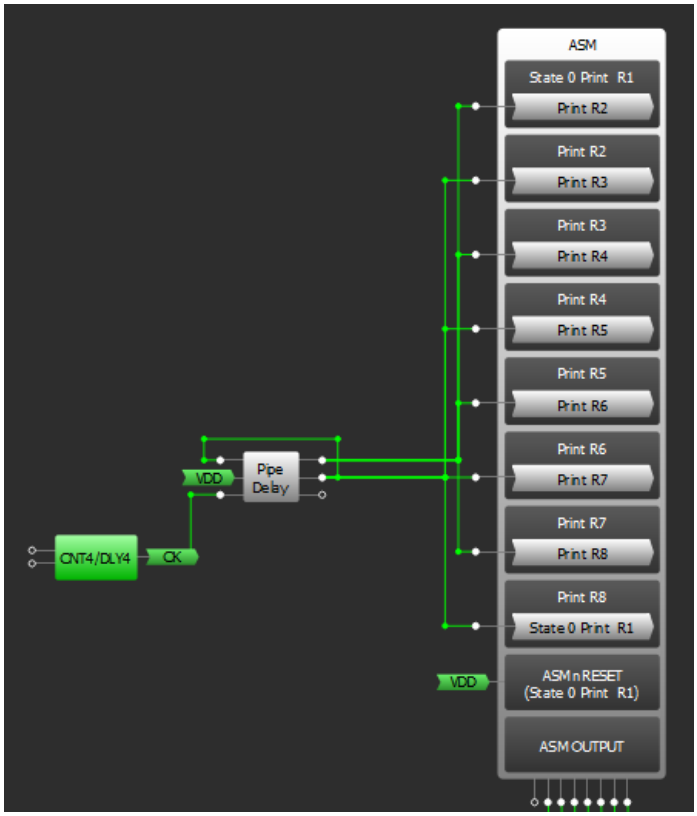

Figure 9. Connect ASM block with pulse generator

The movement from the current state to the next one happens by applying a high signal on the next state enable's pin and a low signal on the current state enable's pin.

Take State 0 for example: its box contains a state 1 arrow as the next state. This means that in order to transition from State 0 to 1, the input of the State 1 arrow needs to be high when the ASM is in state 0.

A counter with pipe delay will be responsible to generate the pulse. CNT4 is used to generate a 0.32ms timer that is used as the one-shot pulse width for all the states. Since the ASM's inputs are level triggered and not edge triggered, the CNT4's output can't simply be used as the trigger for all state transitions (because it would cause almost instantaneous transitions from a state to another instead of waiting for 0.32ms between transitions).

To address this, the Pipe Delay macro-cell is used to generate 2 complementary outputs with 50% duty cycle. While one signal is used to transition from even to odd numbered states, the other is

used to transition from odd to even numbered states.

This design can be approached from the ASM properties bar or from the ASM editor as shown in Figure 9. The ASM Editor is provided with a graphical buttons panel to set the ASM output for each state (connection matrix output RAM). So the initial character (which is to be displayed on the LED matrix) can be drawn before updating from mobile app.

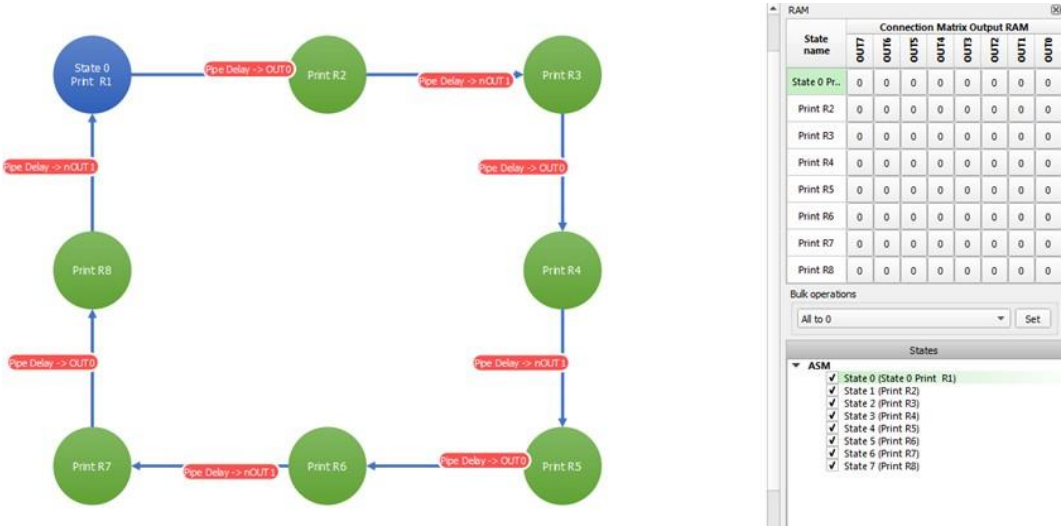

Figure 10. ASM Editor Diagram

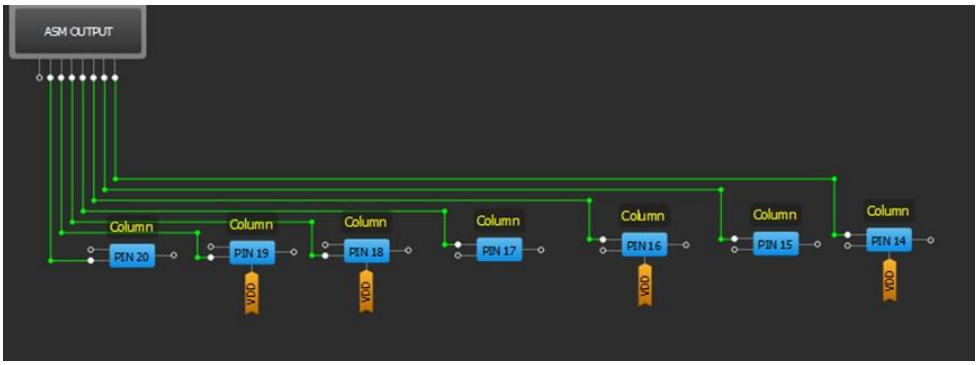

Figure 11. Connect ASM outputs with columns pins

In every state, the ASM output will be the row byte which we want to show, so we connect the ASM outputs directly to the GreenPAK output pins which are in turn connected to the LED matrix columns, (pins 14 to 20).

With every state, one row that is related to the state has to be activated and because the cathodes are wired together in rows, the active signal for rows is low; hence, a low signal is applied on the related row. A series of 8 DFF will give a low signal on one output (with every CLK) while giving a high signal for the other outputs.

This low signal will move from one row to the next with every rising edge's CLK. To do that, the output of every DFF is connected with the next DFF's Dpin. (The last one feeds back to the first DFF to complete the loop). The initial polarity of the first DFF is low while it is high for the other DFFs.

B.O.M and Tools

The components used in this project are:

- SLG46537V GreenPAK5 device https://www.dialog-semiconductor.com/products/slg46537
- HC-06 Bluetooth module https://tronixlabs.com.au/breakout-boards/bluetooth/hc06-bluetooth-to-uart-serial-wireless-adaptor-australia/
- Arduino UNO board https://www.sparkfun.com/products/11021
- 8x8 Common Row Cathodes LED Matrix https://www.adafruit.com/products/455 - 200 ohm

resistors

- GreenPAK designer https://www.dialog-semiconductor.com/greenpak-designer-software
- Arduino IDE https://www.arduino.cc/en/Main/Software

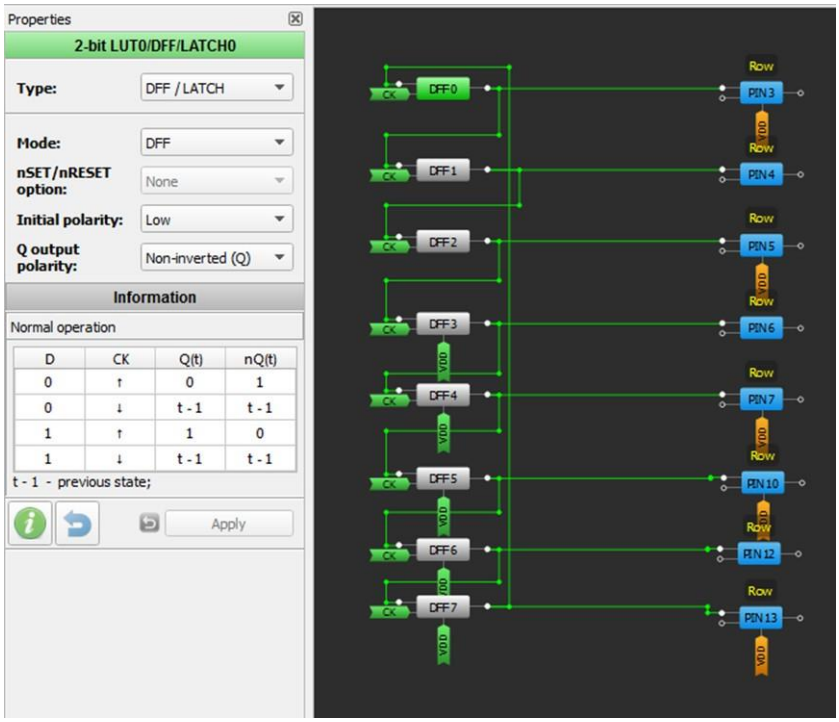

Figure 12. Activation of one row with every state

Arduino Code

In this stage, the Arduino, which is connected with the Bluetooth module, will receive the 8 bytes using the UART interface. It will perform data fitting and will send them to the GreenPAK5 using I2C. The HC06 module that has been used for Bluetooth communication uses the UART for communication protocol with a 9600 baud rate. That number will be used in the code.

Arduino UNO has a serial interface connected with pins 0,1 (RX,TX) to make the UART connection with other components. So the Bluetooth module TX will be connected with the Arduino RX (pin 0). Bluetooth module TX > Arduino RX (pin 0).

After receiving data bytes, Arduino has to be used to send these bytes (using i2C protocol) to GreenPAK. To make this easy, Silego's Arduino Library has been used.

Silego's Arduino Library is a custom library. By using it, data can easily be read from or written to a GreenPAK device from the Arduino using I2C. There are two functions available in the Silego library: writeI2C and readI2C. The syntax to call one of these functions in an Arduino sketch is silego.function(parameters); In our project, the writeI2C function will be used, which has this form: void writeI2C(byte byte_address, byte data);

Variable	Function
byte_address	The GreenPAK register address which we want to write
byte data	Data which has to be written on the address.

Table 2. Arduino writing function variables

The byte address is found in the SLG datasheet. In our project, the byte addresses that we will use are the addresses of the RAM 8 outputs for ASM-states, as shown in table 3. However, because the Arduino is used as I2C master, a Silego library for Arduino will have to be used in order to get static variable names instead of HEX numbers in Arduino functions as shown in the 3rd column of table 3.

Address byte	Signal Function	Arduino static variables
D0	RAM 8 outputs for ASM-state0	ASM_STATE_0
D1	RAM 8 outputs for ASM-state1	ASM_STATE_1
D2	RAM 8 outputs for ASM-state2	ASM_STATE_2
D3	RAM 8 outputs for ASM-state3	ASM_STATE_3
D4	RAM 8 outputs for ASM-state4	ASM_STATE_4
D5	RAM 8 outputs for ASM-state5	ASM_STATE_5
D6	RAM 8 outputs for ASM-state6	ASM_STATE_6
D7	RAM 8 outputs for ASM-state7	ASM_STATE_7

Table 3. ASM outputs state addresses

The "SilegoLibrary.zip" folder needs to be downloaded. The folder needs to be unzipped in the Arduino libraries directory before programming. The folder name has to have no spaces. For detailed information you can see AN-1107 How to Use Silego's Arduino Library with GreenPAK.

This library is compatible with SLG46531V, SLG46532V, SLG46533V. However, we can use it with SLG46537V because SLG46531V and SLG46537V have the same I2C addresses.

I2C Arduino UNO pins: SCL--->A5 , SDA--->A4.

The final code:

```
#include <Wire.h>
#include "Silego.h"     // Include Silego header file
#include "macros/SLG46531.h"    // Include macros for SLG46531
byte rows_byte[8];    //matrix to hold rows bytes
int i=0 ;                                          //counter
boolean s=0;                                    //state flag

// Create an instance of Silego class called
// "silego" with device address 0x00
Silego silego(0x00);
void setup() {
Serial.begin(9600);                //Bluetooth module's baud rate
}
void loop() {
while(Serial.available()>0) //receiving bytes coming from
{                                                  //Bluetooth
rows_byte[i]=Serial.read();
delay(5);
i++;
s=1;                                       //set flag
}
if(s==1)              // send bytes to GreenPAK using I2C
{
   silego.writeI2C(ASM_STATE_0, rows_byte[0]);
   silego.writeI2C(ASM_STATE_1, rows_byte[1]);
   silego.writeI2C(ASM_STATE_2, rows_byte[2]);
   silego.writeI2C(ASM_STATE_3, rows_byte[3]);
   silego.writeI2C(ASM_STATE_4, rows_byte[4]);
   silego.writeI2C(ASM_STATE_5, rows_byte[5]);
   silego.writeI2C(ASM_STATE_6, rows_byte[6]);
   silego.writeI2C(ASM_STATE_7, rows_byte[7]);
   s=0;     // Reset flag
    i=0;    //  Rest counter
 } }
```

Circuit Schematic

GreenPAK pin	11	12	13	14	15	16	17	18	19	20
Function	GND	R7	R8	C1	C2	C3	C4	C5	C6	C7

GreenPAK pin	1	2	3	4	5	6	7	8	9	10
Function	VDD	-	R1	R2	R3	R4	R5	R6	R7	R8

Table 4. GreenPAK Output Pins Connection

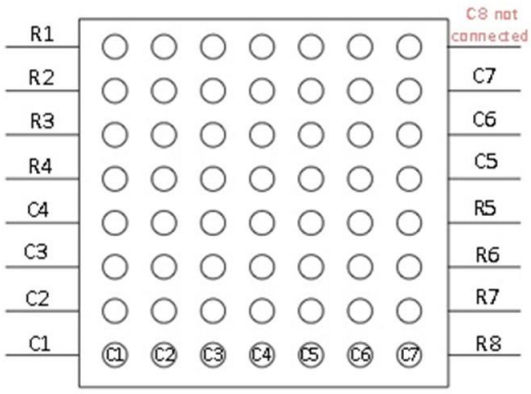

Figure 13.LED Matrix Pins Diagram

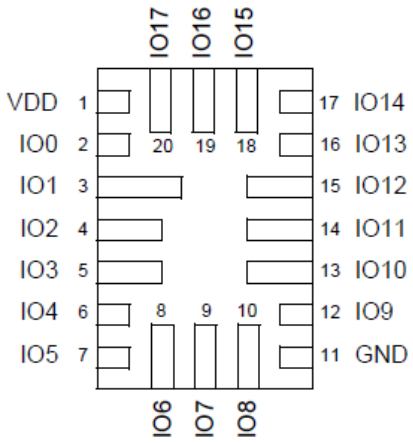

Figure 14. SLG46537V Pinout

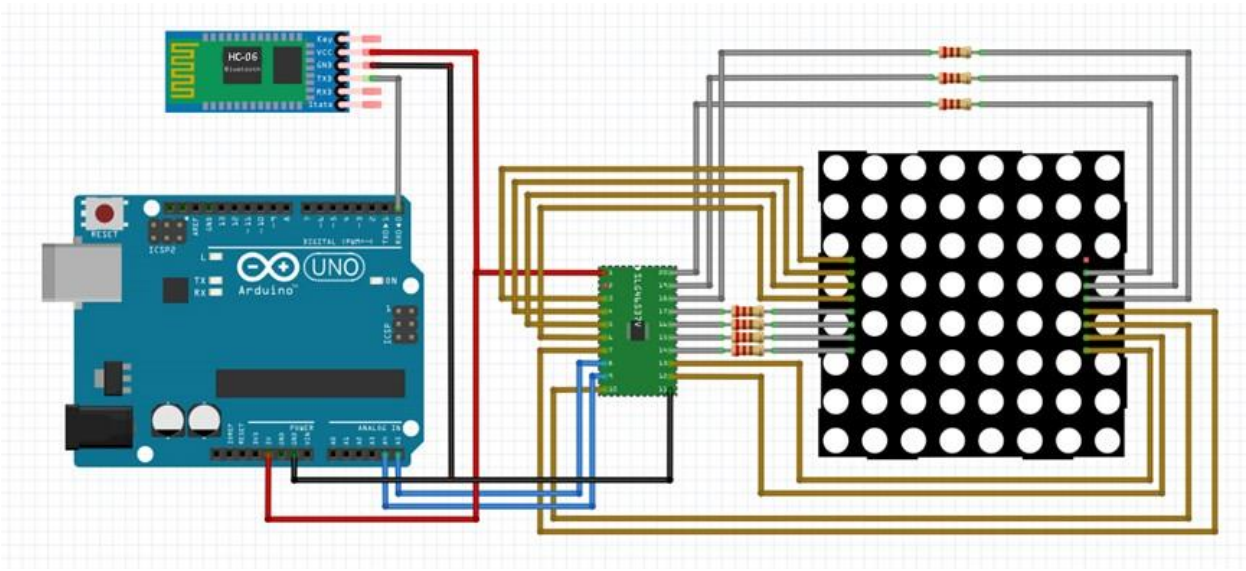

Figure 15. Top level Circuit Schematic

Conclusion

In this Application Note, a LED matrix driver has been created that can be controlled from ucontroller via the I2C protocol. An android app has also been built with a virtual LED matrix to draw characters and to display them on the LED matrix. The GreenPAK Programmable Mixed- signal ASIC has a variety of digital and analog components that facilitate the creation of moderately complex designs. It also has the I2C interface which allows control of GreenPAK registers from any I2C master.

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 skilled in the art 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 for development of 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 out of 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.

(Rev.1.0 Mar 2020)

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/

Trademarks

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