Application Note

SLG46824/6 MTP Arduino Programming Example

AN-CM-255

Abstract

In this application note, we use the Arduino MTP Programmer sketch to program an SLG46824/6. Through analyzing the code, a firmware designer can create a modified version that is compatible with their unique microcontroller.

This application note comes complete with design files which can be found in the References section.

Contents

Figures

Tables

1 Terms and Definitions

EEPROM	Electrically erasable programmable read-only memory
I^{2}C	Inter-integrated circuit
MTP	Multiple-time programmable
NVM	Non-volatile memory
OTP	One-time programmable

2 References

For related documents and software, please visit:

https://www.dialog-semiconductor.com/configurable-mixed-signal.

Download our free GreenPAK Designer software [1] to open the .gp files [2] and view the proposed circuit design. Use the GreenPAK development tools [3] to freeze the design into your own customized IC in a matter of minutes. Dialog Semiconductor provides a complete library of application notes [4] featuring design examples as well as explanations of features and blocks within the Dialog IC.

[1] GreenPAK Designer Software, Software Download and User Guide, Dialog Semiconductor

[2] AN-CM-255 SLG46824/6 MTP Arduino Programming Eample.gp, GreenPAK Design File, Dialog Semiconductor

[3] GreenPAK Development Tools, GreenPAK Development Tools Webpage, Dialog Semiconductor

[4] GreenPAK Application Notes, GreenPAK Application Notes Webpage, Dialog Semiconductor

[5] In-System Programming Guide, GreenPAK User Guides and Manuals, Dialog Semiconductor

3 Introduction

In this application note, we show how to use the SLG46824/6 Arduino programming sketch to program a Dialog SLG46824/6 GreenPAK™ Multiple-Time Programmable (MTP) device.

Most GreenPAK devices are One-Time Programmable (OTP), meaning that once their Non-Volatile Memory bank (NVM) is written, it cannot be overwritten. GreenPAKs with the MTP feature, like the SLG46824 and SLG46826, have a different type of NVM memory bank that can be programmed more than once.

We've written an Arduino sketch that allows the user to program an MTP GreenPAK with a few simple serial monitor commands. In this application note we use an SLG46826 as our GreenPAK with MTP.

We provide sample code for the Arduino Uno using an open-source platform based on C/C++. Designers should extrapolate the techniques used in the Arduino code for their specific platform.

For specific information regarding I^{2}C signal specifications, I^{2}C addressing, and memory spaces, please reference the GreenPAK In-System Programming Guide provided on the SLG46826 product page. This application note provides a simple implementation of this programming guide.

4 Arduino-GreenPAK Connections

To program the NVM of our SLG46826 GreenPAK with our Arduino sketch, we'll first need to connect four Arduino Uno pins to our GreenPAK. You can connect these pins directly to the GreenPAK Socket Adapter or to a breakout board with the GreenPAK soldered down.

Table 1: Arduino Uno / GreenPAK Connections

GreenPAK	Arduino
VDD (Pin 1)	Digital Pin 2
GND (Pin 11)	GND
SCL (Pin 8)	A5
SDA (Pin 9)	A4

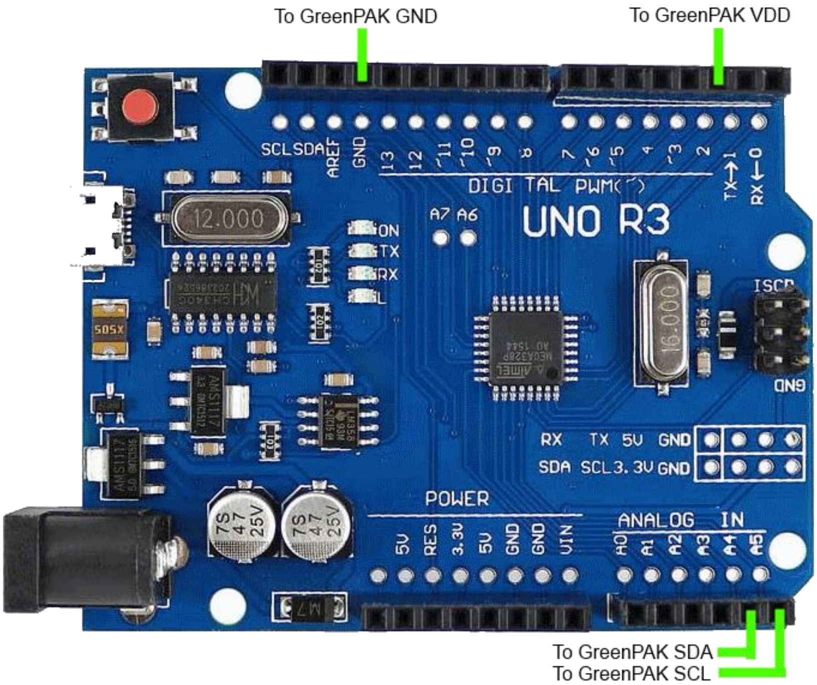


Figure 1. Arduino Connections

Please note that external **I^{2}C** pull up resistors are not shown in Figure 1. Please connect a 4.7 kΩ pull up resistor from both SCL and SDA to the Arduino's 3.3 V output.

5 Exporting GreenPAK NVM Data from a GreenPAK Design File

We'll put together a very simple GreenPAK design to illustrate how to export the NVM data. The design below is a simple level shifter where the blue pins on the left are tied to VDD (3.3v), while the yellow pins on the right are tied to VDD2 (1.8v).

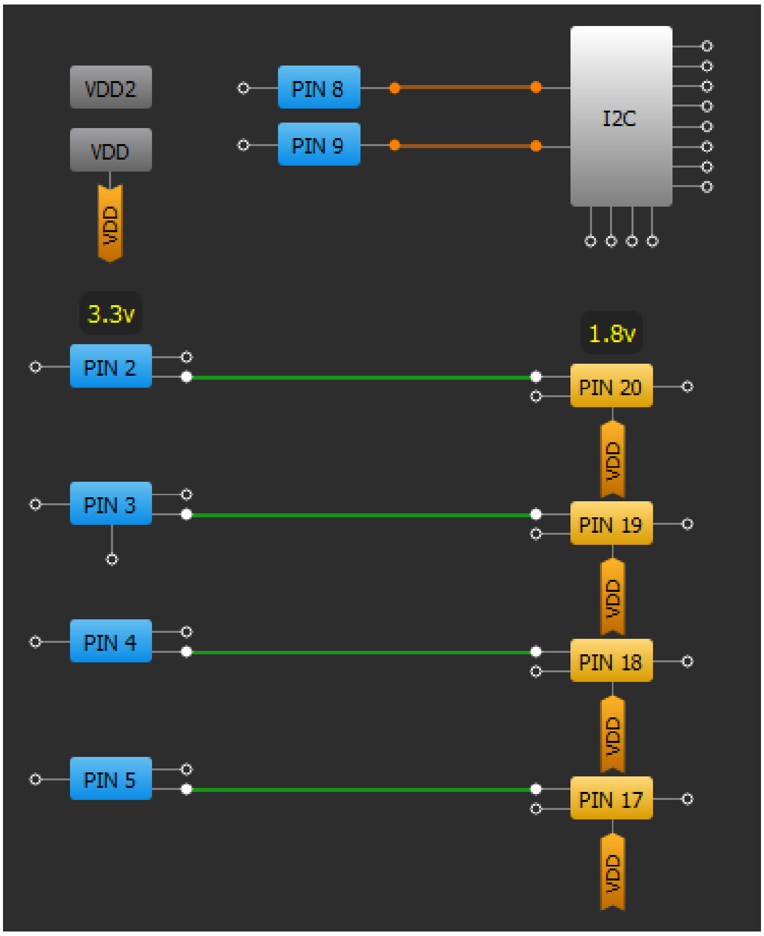


Figure 2. Simple GreenPAK Design in a SLG46826

To export the information from this design, you need to select File → Export → Export NVM, as shown in Figure 3.

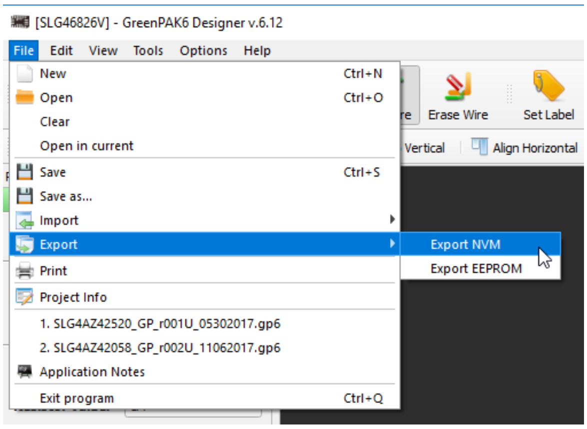
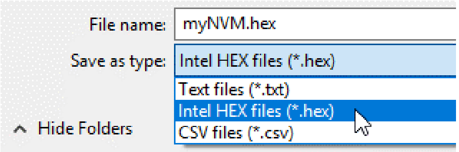


Figure 3. Export NVM

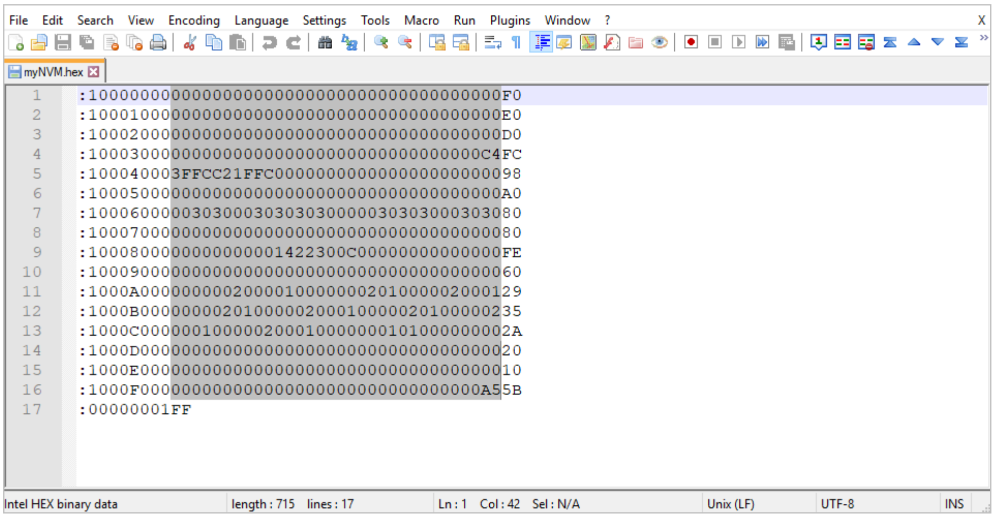
You will then need to select Intel HEX Files (*.hex) as the file type and save the file.


File name: myNVM.hex

Save as type: Intel HEX files (*.hex)

Figure 4. Save as .hex File

Now, you'll need to open the .hex file with a text editor (like Notepad++). To learn more about the Intel's HEX file format and syntax, check out its Wikipedia page. For this application we're only interested in the data portion of the file as shown in Figure 5.

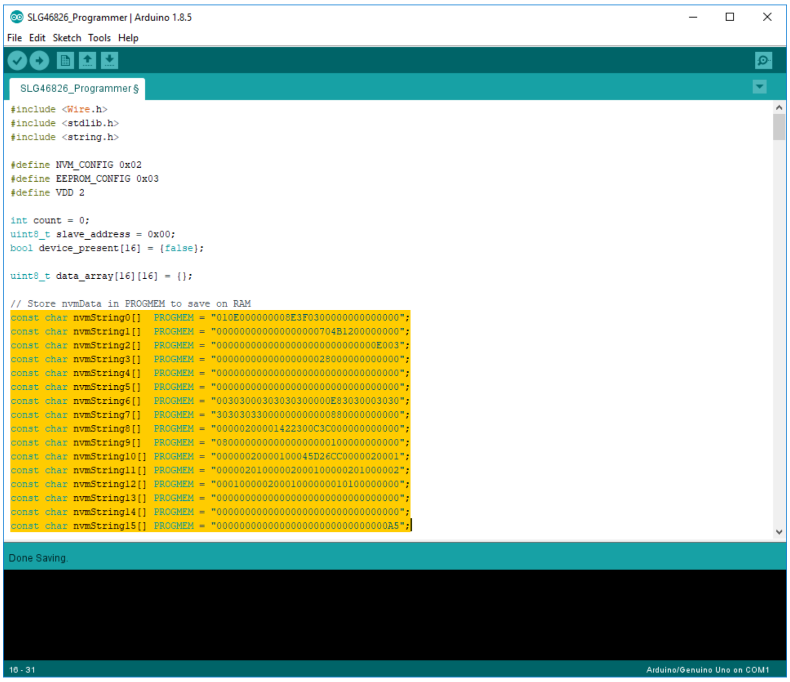


```
:1000000000000000000000000000000000000000F0
:1000100000000000000000000000000000000000E0
:1000200000000000000000000000000000000000D0
:10003000000000000000000000000000000000C4FC
:100040003FFCC21FFC000000000000000000000098
:1000500000000000000000000000000000000000A0
:100060000030300030303030000030303000303080
:100070000000000000000000000000000000000080
:1000800000000000001422300C00000000000000FE
:100090000000000000000000000000000000000060
:1000A0000000002000010000000201000002000129
:1000B0000000020100000200010000020100000235
:1000C000000100000200010000000101000000002A
:1000D0000000000000000000000000000000000020
:1000E0000000000000000000000000000000000010
:1000F000000000000000000000000000000000A55B
:00000001FF
```

Figure 5. Viewing the NVM Data in Notepad++

Highlight and copy the 256 bytes of NVM configuration data located within the HEX file. Each line that we are copying is 32 characters long, which corresponds to 16 bytes.

Paste the information into the highlighted nvmString[] section of the Arduino sketch as shown in Figure 6. If you're using a non-Arduino Microcontroller, you could write a function to parse the nvmData saved in the GreenPAK .GP6 file. (If you open a GreenPAK file with a text editor, you'll see that we store project information in an easily-accessible XML format.)



SLG46826_Programmer | Arduino 1.8.5
File Edit Sketch Tools Help
SLG46826_Programmer §

```
#include <Wire.h>
#include <stdlib.h>
#include <string.h>

#define NVM_CONFIG 0x02
#define EEPROM_CONFIG 0x03
#define VDD 2

int count = 0;
uint8_t slave_address = 0x00;
bool device_present[16] = {false};

uint8_t data_array[16][16] = {};

// Store nvmData in PROGMEM to save on RAM
const char nvmString0[]  PROGMEM = "010E000000008E3F0300000000000000";
const char nvmString1[]  PROGMEM = "000000000000000000704B1200000000";
const char nvmString2[]  PROGMEM = "0000000000000000000000000000E003";
const char nvmString3[]  PROGMEM = "00000000000000000028000000000000";
const char nvmString4[]  PROGMEM = "00000000000000000000000000000000";
const char nvmString5[]  PROGMEM = "00000000000000000000000000000000";
const char nvmString6[]  PROGMEM = "00303000303030300000E83030003030";
const char nvmString7[]  PROGMEM = "30303033000000000000880000000000";
const char nvmString8[]  PROGMEM = "00000200001422300C3C000000000000";
const char nvmString9[]  PROGMEM = "08000000000000000000100000000000";
const char nvmString10[] PROGMEM = "00000020000100045D26CC0000020001";
const char nvmString11[] PROGMEM = "00000201000002000100000201000002";
const char nvmString12[] PROGMEM = "00010000020001000000010100000000";
const char nvmString13[] PROGMEM = "00000000000000000000000000000000";
const char nvmString14[] PROGMEM = "00000000000000000000000000000000";
const char nvmString15[] PROGMEM = "000000000000000000000000000000A5";
```

Figure 6. Arduino Sketch

To set the EEPROM data for your GreenPAK design, select the EEPROM block from the components panel, open its properties panel, and click "Set Data."

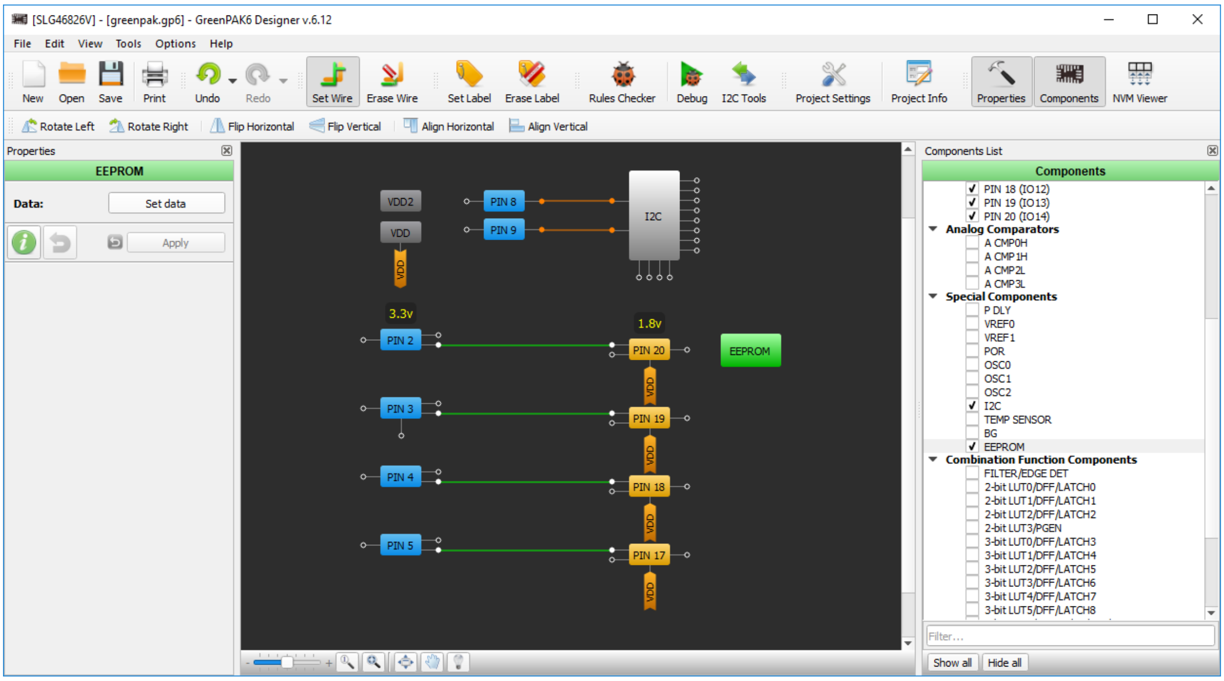
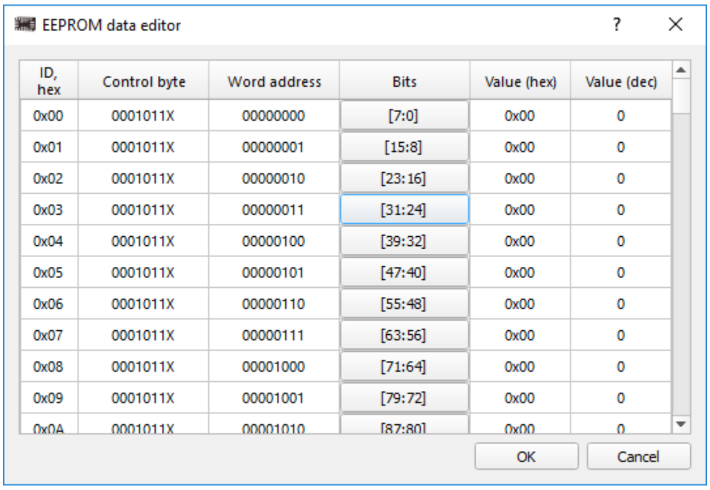


Figure 7. Set EEPROM Data

Now you can edit each byte in the EEPROM individually with our GUI interface.

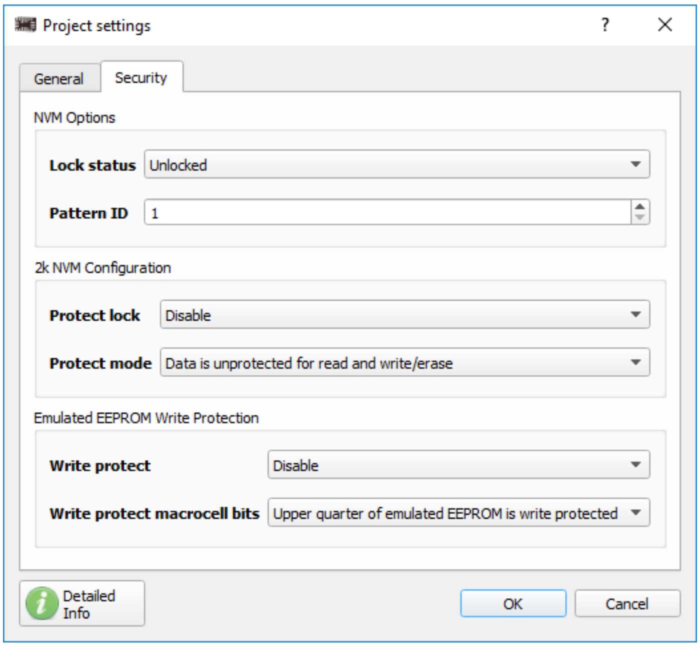


ID, hex	Control byte	Word address	Bits	Value (hex)	Value (dec)
0x00	0001011X	00000000	[7:0]	0x00	0
0x01	0001011X	00000001	[15:8]	0x00	0
0x02	0001011X	00000010	[23:16]	0x00	0
0x03	0001011X	00000011	[31:24]	0x00	0
0x04	0001011X	00000100	[39:32]	0x00	0
0x05	0001011X	00000101	[47:40]	0x00	0
0x06	0001011X	00000110	[55:48]	0x00	0
0x07	0001011X	00000111	[63:56]	0x00	0
0x08	0001011X	00001000	[71:64]	0x00	0
0x09	0001011X	00001001	[79:72]	0x00	0
0x0A	0001011X	00001010	[87:80]	0x00	0

Figure 8. EEPROM Data Editor

Once your EEPROM data is set, you can export it to a HEX file using the same method described previously for exporting the NVM data. Insert these 256 bytes of EEPROM data into the eepromString[] section of the Arduino sketch.

For each custom design, it is important to check the protection settings within the "Security" tab of the project settings. This tab configures the protection bits for the matrix configuration registers, the NVM, and the EEPROM. Under certain configurations, uploading the NVM sequence can lock the SLG46824/6 to its current configuration and remove the MTP functionality of the chip.



NVM Options

Lock status: Unlocked

Pattern ID: 1

2k NVM Configuration

Protect lock: Disable

Protect mode: Data is unprotected for read and write/erase

Emulated EEPROM Write Protection

Write protect: Disable

Write protect macrocell bits: Upper quarter of emulated EEPROM is write protected

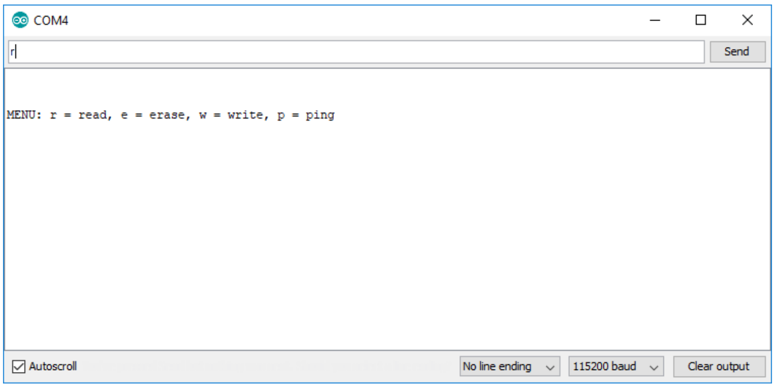
Figure 9. Matrix Registers, NVM, and EEPROM Protection Settings

6 Use the Arduino Sketch

Upload the sketch to your Arduino and open the serial monitor with a 115200 baud rate. Now you can use the sketch's MENU prompts to perform several commands:

- Read - reads either the device’s NVM data or EEPROM data using the specified slave address
- Erase - erases either the device’s NVM data or EEPROM data using the specified slave address
- Write - Erases and then writes either the device’s NVM data or EEPROM data using the specified slave address. This command writes the data that is saved in the nvmString[] or eepromString[] arrays.
- Ping - returns a list of device slave addresses that are connected to the I²C bus

The results of these commands will be printed to the serial monitor console.



```
r
```

```
MENU: r = read, e = erase, w = write, p = ping
```

Figure 10. Arduino Serial Monitor

7 Programming Tips and Best Practices

Over the course of supporting the SLG46824/6, we’ve documented a few programming tips to help avoid common pitfalls associated with erasing and writing to the NVM address space. The following subsections outline this topic in more detail.

7.1 Executing Precise 16-Byte NVM Page Writes:

When writing data to the SLG46824/6’s NVM, there are three techniques to avoid:

- Page writes with less than 16 bytes
- Page writes with more than 16 bytes
- Page writes that don’t begin at the first register within a page (IE: 0x10, 0x20, etc.)

If any of the above techniques are used, the MTP interface will disregard the I²C write to avoid loading the NVM with incorrect information. We recommend performing an I²C read of the NVM address space after writing to verify correct data transfer.

7.2 Transferring NVM Data into the Matrix Configuration Registers

When the NVM is written, the matrix configuration registers are not automatically reloaded with the newly written NVM data. The transfer must be initiated manually by cycling the PAK VDD or by generating a soft reset using I²C. By setting register <1601> in address 0xC8, the device re-enables the Power-On Reset (POR) sequence and reloads the register data from the NVM into the registers.

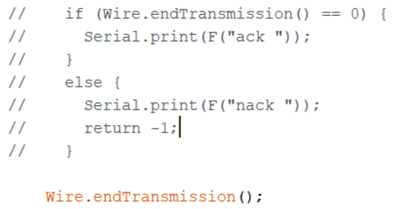
7.3 Resetting the I²C Address after an NVM Erase:

When the NVM is erased, the NVM address containing the I²C slave address will be set to 0000. After the erase, the chip will maintain its current slave address within the configuration registers until the device is reset as described above. Once the chip has been reset, the I²C slave address must be set in address 0xCA within the configuration registers each time the GreenPAK is power-cycled or reset. This must be done until the new I²C slave address page has been written in the NVM.

8 Errata Discussion

When writing to the "Page Erase Byte" (Address: 0xE3), the SLG46824/6 produces a non-I2C compliant ACK after the "Data" portion of the I2C command. This behavior might be interpreted as a NACK depending on the implementation of the I2C master.

To accommodate for this behavior, we modified the Arduino programmer by commenting out the code shown in Figure 11. This section of code checks for an I2C ACK at the end of every I2C command in the eraseChip() function. This function is used to erase the NVM and EEPROM pages. Since this section of code is located in a For loop, the "return -1;" line causes the MCU to pre-maturely exit the function.



```
//    if (Wire.endTransmission() == 0) {
//      Serial.print(F("ack "));
//    }
//    else {
//      Serial.print(F("nack "));
//      return -1;
//    }

    Wire.endTransmission();
```

Figure 11: ACK Behavior Modification to the Arduino Programmer

Despite the presence of a NACK, the NVM and EEPROM erase functions will execute properly. For a detailed explanation of this behavior, please reference "Issue 2: Non-I2C Compliant ACK Behavior for the NVM and EEPROM Page Erase Byte" in the SLG46824/6 errata document (Revision XC) on Dialog's website.

9 Conclusion

In this application note we describe the process of using the provided Arduino programmer to upload custom NVM and EEPROM strings to a GreenPAK IC. The code in the Arduino Sketch is thoroughly commented, but if you have any questions regarding the sketch, please contact one of our Field Application Engineers or post your question on our forum. For more in-depth information regarding MTP programming registers and procedures, please reference Dialog's In-System Programming Guide (provided in the Resources section at the link).

Revision History

Revision	Date	Description
1.1	25-Feb-2019	Modified Arduino script to accommodate for SLG46824/6 (XC Revision) errata. Discussion added in Section 8 of this AN.
1.0	05-Sep-2018	Initial Version

Status Definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

Disclaimer

Information in this document is believed to be accurate and reliable. However, Dialog Semiconductor does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information. Dialog Semiconductor furthermore takes no responsibility whatsoever for the content in this document if provided by any information source outside of Dialog Semiconductor.

Dialog Semiconductor reserves the right to change without notice the information published in this document, including without limitation the specification and the design of the related semiconductor products, software and applications.

Applications, software, and semiconductor products described in this document are for illustrative purposes only. Dialog Semiconductor makes no representation or warranty that such applications, software and semiconductor products will be suitable for the specified use without further testing or modification. Unless otherwise agreed in writing, such testing or modification is the sole responsibility of the customer and Dialog Semiconductor excludes all liability in this respect.

Customer notes that nothing in this document may be construed as a license for customer to use the Dialog Semiconductor products, software and applications referred to in this document. Such license must be separately sought by customer with Dialog Semiconductor.

All use of Dialog Semiconductor products, software and applications referred to in this document are subject to Dialog Semiconductor's Standard Terms and Conditions of Sale, available on the company website (www.dialog-semiconductor.com) unless otherwise stated.

Dialog and the Dialog logo are trademarks of Dialog Semiconductor plc or its subsidiaries. All other product or service names are the property of their respective owners.

Contacting Dialog Semiconductor

United Kingdom (Headquarters)
Dialog Semiconductor (UK) LTD
Phone: +44 1793 757700

Germany
Dialog Semiconductor GmbH
Phone: +49 7021 805-0

The Netherlands
Dialog Semiconductor B.V.
Phone: +31 73 640 8822

Email:
enquiry@diasemi.com

North America
Dialog Semiconductor Inc.
Phone: +1 408 845 8500

Japan
Dialog Semiconductor K. K.
Phone: +81 3 5769 5100

Taiwan
Dialog Semiconductor Taiwan
Phone: +886 281 786 222

Web site:
www.dialog-semiconductor.com

Hong Kong
Dialog Semiconductor Hong Kong
Phone: +852 2607 4271

Korea
Dialog Semiconductor Korea
Phone: +82 2 3469 8200

China (Shenzhen)
Dialog Semiconductor China
Phone: +86 755 2981 3669

China (Shanghai)
Dialog Semiconductor China
Phone: +86 21 5424 9058