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

Ab	strac		1
Со	ntent	s	2
Fig	jures.		2
1	Tern	ns and Definitions	3
2	Refe	rences	3
3	Intro	duction	4
4	Ardu	ino-GreenPAK Connections	5
5	Ехро	orting GreenPAK NVM Data from a GreenPAK Design File	6
6	Use	the Arduino Sketch	10
7	Prog	ramming Tips and Best Practices	10
	7.1	Executing Precise 16-Byte NVM Page Writes or change to Deviations from the valid command structure?:	10
	7.2	Transferring NVM Data into the Matrix Configuration Registers	10
	7.3	Resetting the I ² C Address after an NVM Erase:	11
8	Erra	a Discussion	11
9	Con	clusion	11
Re	visior	n History	12

Figures

Figure 1. Arduino Connections	. 5
Figure 2. Simple GreenPAK Design in a SLG46826	. 6
Figure 3. Export NVM	. 6
Figure 4. Save as .hex File	. 7
Figure 5. Viewing the NVM Data in Notepad++	. 7
Figure 6. Arduino Sketch	. 8
Figure 7. Set EEPROM Data	. 8
Figure 8. EEPROM Data Editor	. 9
Figure 9. Matrix Registers, NVM, and EEPROM Protection Settings	. 9
Figure 10. Arduino Serial Monitor	10
Figure 11: ACK Behavior Modification to the Arduino Programmer	11

Tables

Table 1: Arduino Uno / GreenPAK C	Connections	5
-----------------------------------	-------------	---



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:

GreenPAK[™] Programmable Mixed-Signal Products | Renesas

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. Renesas Electronics provides a complete library of application notes [4] featuring design examples as well as explanations of features and blocks within the IC.

- [1] GreenPAK Designer Software, Software Download and User Guide, Renesas Electronics
- [2] AN-CM-255 SLG46824/6 MTP Arduino Programming Eample.gp, GreenPAK Design File, Renesas Electronics
- [3] GreenPAK Development Tools, GreenPAK Development Tools Webpage, Renesas Electronics
- [4] GreenPAK Application Notes, GreenPAK Application Notes Webpage, Renesas Electronics
- [5] In-System Programming Guide, GreenPAK User Guides and Manuals, Renesas Electronics

Author: Craig Cary

3 Introduction

In this application note, we show how to use the SLG46824/6 Arduino programming sketch to program an 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 I²C signal specifications, I²C 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 l^2C pull up resistors are not shown in Figure 1. Please connect a 4.7 k Ω 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 \rightarrow Export \rightarrow Export NVM, as shown in Figure 3.



Figure 3. Export NVM

Application Note

Revision 1.1

25-Feb-2019

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

Save as type: Intel	HEX files (*.hex)
Text 1 Intel	files (*.bxt) HEX files (*.hex) files (*.csv)

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.

File	Edit S	Search	View	Encoding	Languag	e Settin	gs Tools	Macro	Run	Plugin	s Window	?									Х
6		e 🔒	۵ 🖨	i 🖌 🗈	D D	C i m	b	۹ 🖪		E2 1	JF 🥃 🛽	2	E 💌			ء چ	-5	X	<u> </u>	~ 2	S »
🔚 my	NVM.he	x 🖂																			
1	L	:100	0000	00000	000000	00000	00000	000000	0000	00F0											_
2	2	:100	0100	00000	000000	00000	00000	000000	0000	00E0											
3	3	:100	0200	000000	000000	00000	00000	000000	0000	00D0											
4	1	:100	0300	000000	000000	00000	00000	000000	0000	C4FC											
5	5	:100	0400	03FFCC	21FFC0	00000	00000	000000	0000	0098											
6	5	:100	0500	000000	000000	00000	00000	000000	0000	00A0											
	7	:100	0600	000303	000303	03030	00003	030300	0030	3080											
8		:100	0700	000000	000000	00000	00000	000000	0000	0080											
-	9	:100	0800	000000	000001	42230	0C000	000000	0000	OOFE											
10)	:100	0900	000000	000000	00000	00000	000000	0000	0060											
11		:100	0A00	000000	020000	10000	00020	100000	0200	0129											
12	2	:100	0800	000000	201000	00200	01000	002010	0000	0235											
13	3	:100	0000	000010	000020	00100	00000	101000	0000	002A											
14	1	:100	ODOO	000000	000000	00000	00000	000000	0000	0020											
13	0	:100	0E00	000000	000000	00000	00000	000000	0000	0010											
10	5	:100	0F00	000000	000000	00000	00000	000000	0000.	A55B											
11	/	:000	0000	lFF																	
Intel H	IEX bina	ry data			length : 7	15 lines	: 17	L	n:1 (Col : 42	Sel : N/A			Unix (L	F)	UTF-8	3			INS	

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				
			ø	
SLG46826_Programmer§				I
<pre>#include <wire.h></wire.h></pre>			^	L
<pre>#include <stdlib.h> finclude <stdlib.h></stdlib.h></stdlib.h></pre>				
Finelade (String.n/				
#define NVM_CONFIG 0x02				
#define VDD 2				
<pre>int count = 0; uint0 t slave address = 0x00;</pre>				
<pre>bool device_present[16] = {false};</pre>				
uint8 t data arrav[[6][[6] =]]:				
anno- anadanalisalisal (),				
// Store nvmData in PROGMEM to save on RAM				
const char nymsString(] PROGMEM = "10E000000008E3F03000000000000";				
const char prestring[] PRCHME = 00000000000000000000000000000000000				
const char numstring[] PROGNES = 00000000000000000000000000000000000				
Const char pumpting() PROBME = "0000000000000000000000000000".				
const char nymString511 PROGREW = "00000000000000000000000000000":				
const_char_nvmString6[] PROGHEM = "00303003033030000E83030000303";				
const char nymString7[] PROGMEM = "303033300000000088000000000";				
const char nvmString8[] PROGMEM = "00000200001422300C3C0000000000";				
const char nvmString9[] PROGMEM = "080000000000000000000000000000000000				
const char nvmString10[] PROGMEM = "00000020000100045D26CC0000020001";				
const char nvmString1[] PROGMEM = "000002010000020100000201000002";				
const char nvmString12[] PROGMEM = "00010000020001000000010100000000";				
const_char_nvmString13[] PROGMEM = "00000000000000000000000000000000000				
const char nvmString14[] PRGGMEM = "00000000000000000000000000000000000				
const char nvmString15[] PRGEMEM = "00000000000000000000000000000000000			~	
Dana Cavina				Ľ
Done danny.				ľ
				Ľ
				Ľ
				Ľ
				Ľ
18-31	Arduino/Genuine	Uno on	COM1	

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

Ap	bli	cat	ion	NO	te

Revision 1.1



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	0000000	[7:0]	0x00	0
0x01	0001011X	0000001	[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.

Project settings				?	
General Security					
NVM Options					
Lock status Unlocke	d			Ŧ	
Pattern ID 1				4	
2k NVM Configuration					
Protect lock Disab	le			•	
Protect mode Data	is unprote	cted for read and write/erase		•	
Emulated EEPROM Write F	Protection				
Write protect		Disable		•	
Write protect macro	ocell bits	Upper quarter of emulated EEP	ROM is write protec	ted 🔻	
Detailed		ſ	ОК	Cance	el

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

Δ	h	ali	ca	tio	n I	to
	Ч	JII	La			le



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.

💿 COM4	-		×
4			Send
MENU: r = read, e = erase, w = write, p = ping			
Autoscroll No line ending 🗸 115200 baud	~	Clear	output

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.

Application Note

Revision 1.1

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 prematurely exit the function.

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).

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 In-System Programming Guide.





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

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

Trademarks

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

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 <u>www.renesas.com/contact-us/</u>.