KEITHLEY

Application Note Series

Number 3177

Programming and Erasing Flash Memory Devices Using the Keithley S530 Pulse Generator Option

Introduction and Background

Normally, in parametric test, the instrument used most is the Source Measurement Unit (SMU). The SMU allows supplying a DC voltage or current to the device under test (DUT) and simultaneously measuring the resultant voltage or current. However, there are some cases where it's necessary to apply a voltage to the device in a time-controlled manner. Often, the duration of these applied voltages must be on the order of a few microseconds in order to prevent the DUT from over-heating or being over-stressed. SMUs are not designed to output voltages this quickly. Therefore, a different instrument is required: a pulse generator.

A pulse generator allows outputting a voltage in a time-controlled, time-accurate manner, including control over the amount of voltage (pulse height), the duration of the pulse (pulse width), as well as the voltage ramp rate (rise and fall time). This type of instrument also provides the ability to control the number of pulses that are output and even to synchronize multiple pulses.

The Keithley S530 Parametric Test System offers a pulse generator option that offers two to six channels of pulse outputs, each of which is capable of outputting a maximum of ±40 VDC with pulse durations from 100ns to 1s.

Typical applications for a pulse generator are preventing device heating, time-controlled device stressing or charging, generating clock signals, fuse testing, and setting and resetting memory devices. This note describes how the pulse generator option of the S530 Parametric Test System can be used to characterize flash memory cells.

Flash Memory Basics

Flash memory is currently the dominant form of solid-state, non-volatile memory technology. It is used in a wide range of devices and applications—everything from the common USB "thumb drive" to smartphones, MP3 players, and digital cameras.

Flash memory is part of a class of MOS devices that use floating gates. There are two types of flash cells: NOR and NAND. In NOR technology, the storage cells can be programmed and erased individually. Unfortunately, the storage densities for this type of flash memory are comparatively low. In the second type, NAND, it's possible to write to the cells individually, but they must be erased in blocks. NAND-type memory has a much higher storage density and is by far the most dominant of the two types, so this note will focus on NAND flash memory.

In addition to the floating gate, NAND flash memory cells (***Figure 1***) usually have a control gate, drain, source, and bulk. The memory cell is set (programmed) and reset (erased) by applying or removing charge from the floating gate. Charge can be applied or removed from the floating gate of any type of flash memory cell via Fowler-Nordheim (FN) current tunneling or via Hot Carrier Injection (HCI). In a normal CMOS transistor, both of these mechanisms cause device degradation and are usually to be avoided, but they are beneficial for flash memory. Moreover, although FN tunneling and HCI are useful for programming and erasing flash memory, they are also why flash memory cells have a limited lifetime.

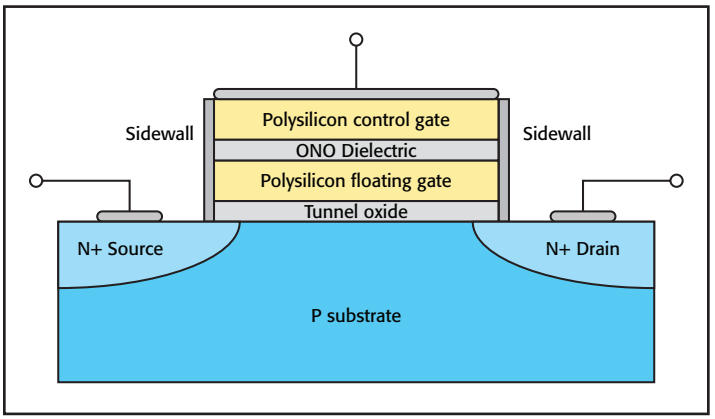

Figure 1. NAND flash memory cell cross-section

When charge is applied to or removed from the floating gate, the threshold voltage (V_T) of the underlying transistor changes (***Figure 2***). This threshold voltage change is what allows the flash memory cell to be used as a memory storage device. Further, once the charge is injected into or removed from the floating gate, the floating gate remains in that state even after power is removed, which means flash memory is *non-volatile*.

To program or erase a flash memory cell, a set of pulses are applied. Pulses are used because applying a steady DC voltage would cause the cell to be over-programmed or over-erased. Once a cell is placed into one of these states, it cannot be set to the opposite state, usually because the gate oxide has been damaged in some way. The stimulus voltage must be applied in a time-controlled manner, which is why a pulse generator is required.

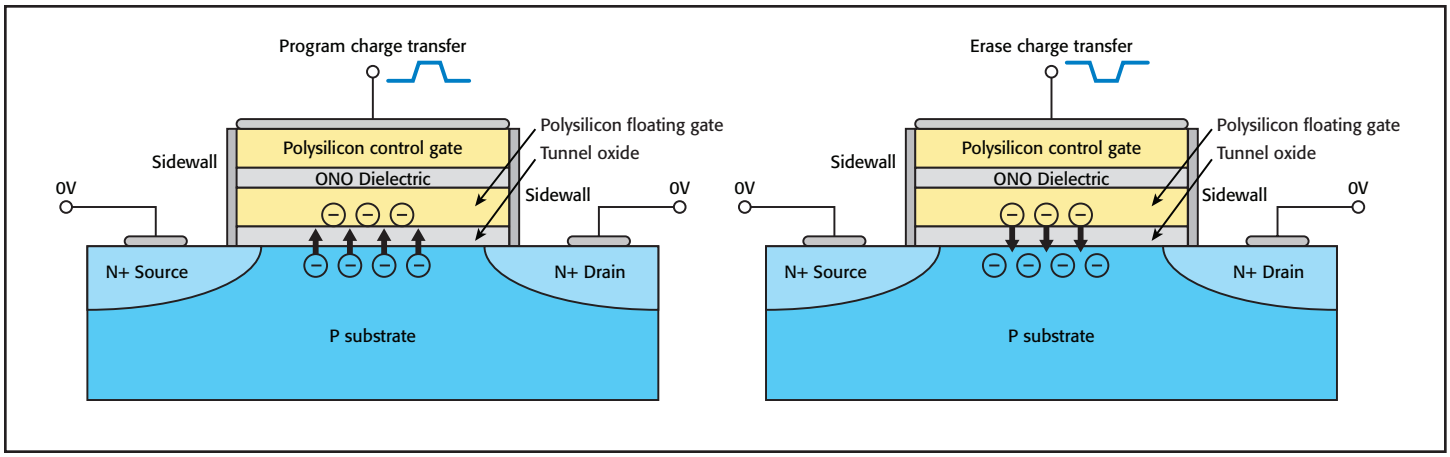

Figure 2. Charge transfer in a NAND flash cell

NAND flash cells fall into two categories: single-bit (logical 0/1) and multi-bit. As the names imply, in single-bit cells, each storage location can hold only one bit; in multi-bit cells, each storage location can hold multiple bits. In a single-bit cell, a two-level pulse is required to set or reset the device, which results in two distinct V_T values (***Figure 3***). In multi-bit cells, multi-level pulses are required to place the cell in each of its possible states, which results in from four to eight possible V_T values (***Figure 4***).

To program the cell using FN tunneling, a positive pulse is applied to the gate, while the drain, source, and bulk voltages are set to 0V (or grounded). This causes charge to be pushed into the floating gate. To erase the cell via FN tunneling, a negative pulse is applied to the gate (with the drain, source, and bulk terminals set to 0V or connected to ground).

To use HCI, simultaneous pulses are applied to the gate and drain (with the source and bulk grounded or set to zero). This causes a field to appear in the transistor channel, thereby creating the necessary hot carriers. The pulse height and polarity of the gate pulse determines whether charge is applied to or removed from the floating gate.

Usually, the threshold voltage is measured afterward to ensure that the cell has indeed been programmed or erased. If one programs and erases the cells thousands of time, one can monitor its lifetime. (For the sake of simplicity, this note focuses only on single-bit flash memory cells.)

Measurement Considerations

A parametric test system is oriented primarily toward performing accurate DC measurements. Therefore, the switch matrix and relays typically used are designed and optimized to ensure good DC performance, such as low leakage current, minimal offset voltages and currents, and low resistances. The optimization of DC performance usually comes at some cost to the system's AC performance.

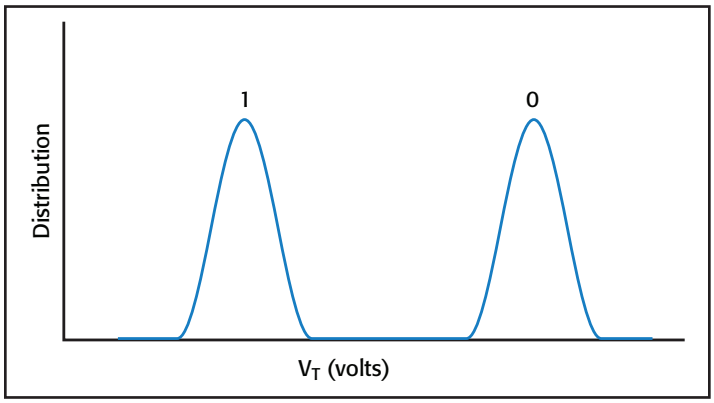

Figure 3. Single-bit V_T distribution

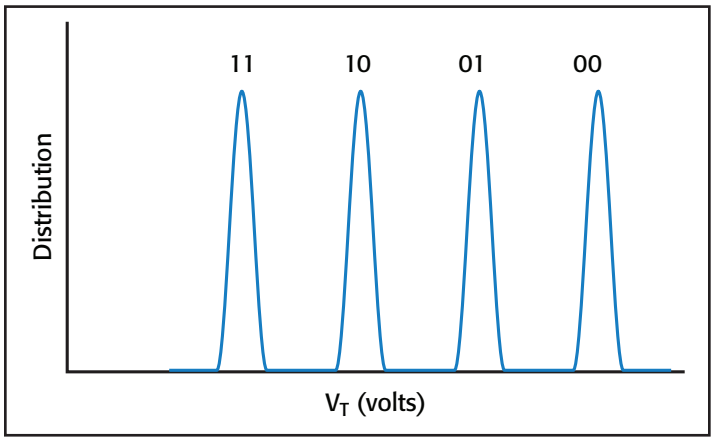

Figure 4. Multi-bit V_T distribution

In contrast, pulses are essentially AC signals. A square pulse train can be represented by the Fourier expansion as an infinite series of sinusoids:

$$f_{pulse}(t) = \frac{\tau}{T} + \sum_{n=1}^{\infty} \frac{2}{n\pi} \sin\left(\frac{\pi n \tau}{T}\right) \cos\left(\frac{2\pi n}{T} t\right)$$

where: T is the period, τ the pulse width, and t is the total time.

Because the switching subsystem of the parametric tester is optimized for DC, it has a limited bandwidth (less than 30MHz).

This limited bandwidth can cause distortions in the pulse signal, such ringing, overshoot, and other harmonic distortions. Given that setting and resetting a flash cell is essentially charge transfer, significant waveform distortions will change the amount of charge that is transferred, potentially resulting in the flash cell being placed in an indeterminate (or undefined) state.

To reduce these distortions, the higher frequency content of the pulse signal must be reduced. This can be done by slowing down the pulse transitions (the rise and fall times) or reducing the pulse width. A good rule of thumb for the pulse generators in the S530 is to keep the rise and fall times greater than 50ns and the pulse widths greater than 150ns.

Another cause of pulse distortions is impedance mismatches. If the impedances in the signal path are not matched, then signal reflections will result, which when combined with the incident signal will result in some of the pulse's frequency content being accentuated and some reduced, the net effect being a non-ideal pulse.

The two sources of impedance mismatches are the DUT and the tester's switching sub-system. In the case of the DUT impedance, the pulse generator can compensate to some extent for the mismatch. This compensation can be achieved by selecting the correct output impedance for the pulse generator (either 50Ω or 1kΩ). This can also help a bit when dealing with the impedance mismatch caused by the switching subsystem's impedance (which is around 90Ω).

Pulse generators, being time-based instruments, do not have Kelvin sensing capabilities. That is, they cannot sense whether the voltages that they are outputting are being accurately applied to the DUT. Therefore, they cannot sense any voltage losses caused by the interconnecting cables, switch matrix, and the impedance of the DUT itself. However, most pulse generators (including the S530's), can correct for most of these losses by calculating how much voltage should be applied given the amount of impedance at the DUT. Usually, this is done through a user-supplied impedance value.

S530 Pulse Functions

The S530 provides several commands for controlling the pulse generator, which are summarized in ***Table 1***.

For detailed information on each of these commands, refer to the documentation included with the S530 Parametric Test System or the online help in the Keithley Interactive Test Tool (KITT).

Table 1: S530 Pulse Option Command Set

Command	Description
pulse_current_limit	Sets the maximum amount of current that pulse channel can supply as a result of the pulse amplitude and DUT impedance
pulse_delay	Sets the amount of time to wait after triggering pulses before outputting a pulse
pulse_fall	Defines the fall time or trailing edge of a pulse
pulse_rise	Defines the rise time, or leading edge of a pulse
pulse_halt	If the PGU is in continuous mode, stops all pulsing
pulse_height	Sets the height of the pulse relative to a 0V base
pulse_init	Initializes all PGUs to the following conditions: • 1kΩ impedance • normal polarity • software trigger • rise/fall time = 100ns • pulse width = 500ns • pulse delay = 0s • pulse height = 0.2V
pulse_load	Sets the expected DUT load impedance (This parameter is used by the PGU to compensate for losses due to the impedance of the DUT and interconnect.)
pulse_mode	Defines the operating mode of the PGU. There are three modes: • Single Pulse Output • Continuous Stream of Pulses • A "burst" of a specified number of pulses
pulse_offset	Defines the voltage characteristics in terms of peak-to-peak amplitude and voltage offset
pulse_period	Sets the period of a continuous stream or burst of pulses
pulse_range	Sets the voltage range of a PGU channel
pulse_trig	Triggers the output of pulses from both channels of all PGU cards
pulse_trig_burst	Triggers a burst of a specified number of pulses from both channels of a single PGU card
pulse_trig_unit	Triggers the output of pulses from both channels of a single PGU card
pulse_width	Defines the width of a pulse for a single PGU channel
select_channel	Selects which channel of a PGU that subsequent setup commands (pulse_rise, pulse_width, etc.) will affect

Command Set Considerations

Before the pulse generators in the S530 can be used in a test sequence, they must first be initialized using the `pulse _ init()` function. This command will establish a communications channel with the pulse generators and place each channel in its safe default state (i.e., no signal being output).

The pulse generator setup commands (`pulse _ rise`, `pulse _ fall`, `pulse _ height`, `pulse _ period`, `pulse _ width`, `pulse _ delay`) operate on a single pulse channel (each pulse generator card in the S530 has two output channels). The `select _ channel` command selects the channel on which subsequent pulse setup commands will operate. For example:

Example 1: Setting up two output channels

```
s1 = pulse_initialize();
s2 = select_channel(1); //Select channel 1
s3 = pulse_width(3e-6); // ch1 pulse width
s4 = pulse_rise(100e-9);// ch1 rise, fall (transition time)
s5 = pulse_fall(100e-9);//
s6 = pulse_height(10.0);// ch1 pulse amplitude
c1 = select_channel(2); //Select channel 2
c2 = pulse_delay(1e-6); //Start ch2 pulse 1us
after channel 1
c3 = pulse_width(1e-6); // ch2 pulse width
c4 = pulse_rise(100e-9);// ch2 rise, fall
c5= pulse_fall(100e-9); //
c6 = pulse_height(5.0); // ch2 pulse amplitude
```

Trigger commands operate on both channels of a S530 pulse card or on all available pulse channels. Several trigger commands are available for the S530 pulse generators. These commands allow triggering a single pulse for both channels of a single pulse card, a burst of pulses for both channels of a pulse card, or a single pulse on all available pulse channels. In order to output a pulse from a single channel, do not use `select_channel` to set up the other channels. The trigger commands are `pulse_trig`, `pulse_trig_unit`, and `pulse_trig_burst`. For example, to cause the example shown in ***Example 1*** to output a pulse, add the following command to the end of the command sequence:

```
trig_status = pulse_trig()
```

Channels can be synchronized through the use of the `pulse_delay` command. For example, in ***Example 1***, the `pulse_delay` command causes Channel 2's pulse to start 1μs after the start of Channel 1's, resulting in the signal shown in ***Figure 5***. To make the pulses between the channels to coincide, either specify 0 for the `pulse_delay`, or leave the command out completely.

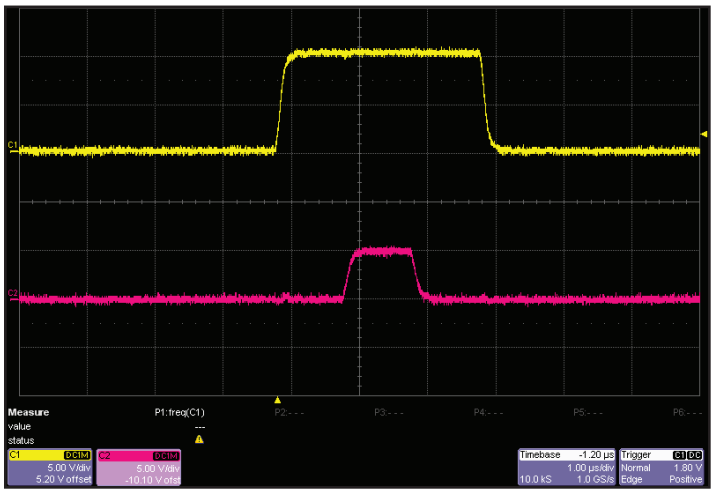

Figure 5. Synchronized pulses

Example 2: Program a NAND cell using FN tunneling

```
#include <stdio.h>
#include <keithley.h>
main()
{
int gatePin = 8;                // Define the connections
int drainPin = 9;
int sourcePin = 10;
int bulkPin = 1;
int commandStatus = 0;          // The pulse commands return
                                // a status value. Negative
                                // values indicate an error
double programVoltage = 12.0;   // The PROGRAM pulse voltage
double transitionTime = 100e-9; // The rise and fall times
double programWidth = 1e-6;     // The PROGRAM pulse width

//
// Make the device connections. Even though
// for this test, we want the drain grounded
// we will use channel 2 of the PGU as
// the ground for the drain by setting its
// pulse height to 0V
//
conpin(PGU1A, gatePin, 0);
conpin(PGU1B, drainPin, 0);
conpin(bulkPin, sourcePin, GND, 0);

//
// Initialize the PGU and check for errors
//
commandStatus = pulse_init();
if (commandStatus < 0) {
      //An error occurred
      printf("Something unexpected happened!\n");
      return;
}
//
// Set up the PROGRAM pulse for the gate
//
commandStatus = select_channel(1);
commandStatus = pulse_mode(1);
commandStatus = pulse_load(1e6);
commandStatus = pulse_height(programVoltage);
commandStatus = pulse_rise(transitionTime);
commandStatus = pulse_fall(transitionTime);
commandStatus = pulse_width(programWidth);

//
// Set up the drain. Here we're using
// channel 2 of the PGU as ground. We
// could have also connected the drain
// terminal to the system ground.
//
commandStatus = select_channel(2);
commandStatus = pulse_load(1e3);
commandStatus = pulse_height(0.0);
commandStatus = pulse_rise(transitionTime);
commandStatus = pulse_fall(transitionTime);
commandStatus = pulse_width(programWidth);

//
// Trigger the Program pulse
//
commandStatus = pulse_trig();

return;
}
```

Example 3: Erase a NAND cell using FN tunneling.

```
#include <stdio.h>
#include <keithley.h>
main()
{
int gatePin = 8;                // Define the connections
int drainPin = 9;
int sourcePin = 10;
int bulkPin = 1;
int commandStatus = 0;          // The pulse commands return
                                // a status value. Negative
                                // values indicate an error
double eraseVoltage = -18.0;    // The ERASE pulse voltage
double transitionTime = 100e-9; // The rise and fall times
double eraseWidth = 1e-3;       // The ERASE pulse width

//
// Make the device connections. Even though
// for this test, we want the drain grounded
// we will use channel 2 of the PGU as
// the ground for the drain by setting its
// pulse height to 0V
//
conpin(PGU1A, gatePin, 0);
conpin(PGU1B, drainPin, 0);
conpin(bulkPin, sourcePin, GND, 0);

//
// Initialize the PGU and check for errors
//
commandStatus = pulse_init();
if (commandStatus < 0) {
      //An error occurred
      printf("Something unexpected happened!\n");
      return;
}
//
// Set up the ERASE pulse for the gate
//
commandStatus = select_channel(1);
commandStatus = pulse_mode(1);
commandStatus = pulse_load(1e6);
commandStatus = pulse_height(eraseVoltage);
commandStatus = pulse_rise(transitionTime);
commandStatus = pulse_fall(transitionTime);
commandStatus = pulse_width(eraseWidth);

//
// Set up the drain. Here we're using
// channel 2 of the PGU as ground. We
// could have also connected the drain
// terminal to the system ground.
//
commandStatus = select_channel(2);
commandStatus = pulse_load(1e3);
commandStatus = pulse_height(0.0);
commandStatus = pulse_rise(transitionTime);
commandStatus = pulse_fall(transitionTime);
commandStatus = pulse_width(eraseWidth);

//
// Trigger the ERASE pulse
//
commandStatus = pulse_trig();

return;
}
```

Example 4: Erase a NAND Cell using the HCI method

```
#include <stdio.h>
#include <keithley.h>
main()
{
int gatePin = 8;                // Define the connections
int drainPin = 9;
int sourcePin = 10;
int bulkPin = 1;
int commandStatus = 0;          // The pulse commands return
                                // a status value. Negative
                                // values indicate an error
double gateEraseVoltage = -18.0; // The ERASE pulse voltages
double drainEraseVoltage = -10.0; //
double transitionTime = 100e-9; // The rise and fall times
double eraseWidth = 1e-3;       // The ERASE pulse width(s)

//
// Make the device connections. Even though
// for this test, we want the drain grounded
// we will use channel 2 of the PGU as
// the ground for the drain by setting its
// pulse height to 0V
//
conpin(PGU1A, gatePin, 0);
conpin(PGU1B, drainPin, 0);
conpin(bulkPin, sourcePin, GND, 0);

//
// Initialize the PGU and check for errors
//
commandStatus = pulse_init();
if (commandStatus < 0) {
//An error occurred
printf("Something unexpected happened!\n");
return;
}
//
// Set up the ERASE pulse for the gate
//
commandStatus = select_channel(1);
commandStatus = pulse_mode(1);
commandStatus = pulse_load(1e6);
commandStatus = pulse_height(eraseVoltage);
commandStatus = pulse_rise(transitionTime);
commandStatus = pulse_fall(transitionTime);
commandStatus = pulse_width(eraseWidth);

//
// Set up the drain. Since we are
// using the HCI mechanism to ERASE
// the cell, the drain must be pulsed
// simultaneously with the gate.
//
commandStatus = select_channel(2);
commandStatus = pulse_load(1e3);
commandStatus = pulse_height(drainEraseVoltage);
commandStatus = pulse_rise(transitionTime);
commandStatus = pulse_fall(transitionTime);
commandStatus = pulse_width(eraseWidth);

//
// Trigger the ERASE pulses
//
commandStatus = pulse_trig();

return;
}
```

Testing a NAND Flash Cell

As mentioned previously, a NAND flash memory cell can be programmed (set) using FN tunneling by applying a positive pulse to the gate while holding the drain, source, and bulk terminals at 0V (or ground). ***Figure 6*** shows the S530 device connections required to program the NAND flash memory cell.

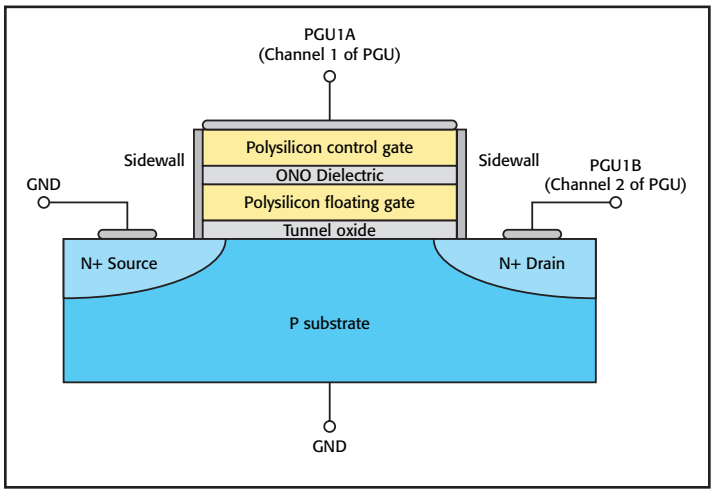

Figure 6. Program/erase connections

For example, let's assume a certain NAND flash cell requires a 12V pulse that's 2μs in width in order to place the cell in the "set" state. Let's further assume that the gate of the flash cell is connected to pin 8 of the S530. The sequence of commands listed in ***Example 2*** can be used to put this cell in "set".

To erase the NAND cell shown in ***Figure 6***, let's assume that it requires a –18V gate pulse that's 1ms in width in order to place the cell in the "reset" state using FN tunneling. The sequence of commands listed in ***Example 3*** can be used to achieve this.

Note that the sequence of commands is mostly the same in ***Example 2*** and ***Example 3***. The only differences are in the pulse heights and pulse widths. In fact, the same sequence of commands could be used with the HCI method to program or erase the cell. For example, if erasing the cell using HCI requires the gate to be pulsed at –18V while simultaneously pulsing the drain to –10V, the code from ***Example 3*** could be modified as shown in ***Example 4***.

As mentioned previously, the threshold voltage (V_T) is usually measured after programming or erasing the flash memory cell in order to verify that the cell's set/reset state has changed. Many different V_T measurement algorithms are available to perform this measurement; contact a local Keithley applications engineer for examples.

Conclusion

This note has provided an overview of how to use the S530 Parametric Test System's pulse source option to program and erase NAND flash memory cells. For further information on these measurements and on the S530 Parametric Test System pulse option, consult the user documentation provided with the test system or contact a local Keithley applications engineer.

Specifications are subject to change without notice.
All Keithley trademarks and trade names are the property of Keithley Instruments, Inc.
All other trademarks and trade names are the property of their respective companies.

KEITHLEY

A GREATER MEASURE OF CONFIDENCE

KEITHLEY INSTRUMENTS, INC. ■ 28775 AURORA RD. ■ CLEVELAND, OH 44139-1891 ■ 440-248-0400 ■ Fax: 440-248-6168 ■ 1-888-KEITHLEY ■ www.keithley.com

BELGIUM
Sint-Pieters-Leeuw
Ph: 02-3630040
Fax: 02-3630064
info@keithley.nl
www.keithley.nl

CHINA
Beijing
Ph: 86-10-8447-5556
Fax: 86-10-8225-5018
china@keithley.com
www.keithley.com.cn

FRANCE
Les Ulis
Ph: 01-69868360
Fax: 01-69868361
info@keithley.fr
www.keithley.fr

GERMANY
Germering
Ph: 089-84930740
Fax: 089-84930734
info@keithley.de
www.keithley.de

INDIA
Bangalore
Ph: 080-30792600
Fax: 080-30792688
support_india@keithley.com
www.keithley.in

ITALY
Peschiera Borromeo (Mi)
Ph: 02-5538421
Fax: 02-55384228
info@keithley.it
www.keithley.it

JAPAN
Tokyo
Ph: 81-3-6714-3070
Fax: 81-3-6714-3080
info.jp@keithley.com
www.keithley.jp

KOREA
Seoul
Ph: 82-2-574-7778
Fax: 82-2-574-7838
keithley@keithley.co.kr
www.keithley.co.kr

MALAYSIA
Penang
Ph: 60-4-643-9679
Fax: 60-4-643-3794
sea@keithley.com
www.keithley.com

NETHERLANDS
Son
Ph: 040-2675502
Fax: 040-2675509
info@keithley.nl
www.keithley.nl

SINGAPORE
Singapore
Ph: 65-6747-9077
Fax: 65-6747-2991
sea@keithley.com
www.keithley.com.sg

TAIWAN
Hsinchu
Ph: 886-3-572-9077
Fax: 886-3-572-9031
info_tw@keithley.com
www.keithley.com.tw

UNITED KINGDOM
Bracknell
Ph: 044-1344-392450
Fax: 044-1344-392457
info@keithley.co.uk
www.keithley.co.uk

© Copyright 2012 Keithley Instruments, Inc.
Printed in the U.S.A No. **3177** 8.1.12