

## Benefits of Register-Based Polling over Data Polling

Author: Abu Sanallah

AN201783 explores the benefits of moving to the new status register polling method instead of the data (DQ) polling method and discusses Cypress's low-level driver software, which can greatly reduce the developer's flash driver development time.

### 1 Abstract

Cypress is adopting a register-based status determination model for its memory devices instead of the data (DQ) polling method. This paper will explore the benefits of moving to the new status register polling. Furthermore, it will discuss Cypress's low-level driver software, which can greatly reduce the developer's Flash driver development time. A software layer that interfaces between current DQ polling software and status register will also be discussed.

### 2 Introduction

Cypress Flash memories have traditionally used data (DQ) polling to determine the status of a program or erase operation. However, software developers have sometimes found this method difficult to implement and they would often use a timeout instead. To make status determination simpler, Cypress is migrating to a register-based method, in which developers can access a memory-mapped location to query status. This provides an easier, improved method that most developers are familiar with.

### 3 Status Register and DQ Polling Comparison

In this section, we will show the weaknesses of the DQ polling method and detail how the status register method design addresses these problems.

#### 3.1 DQ Polling

In the DQ polling method certain bits maintain a fixed state and certain bits toggle (change value every read access). For example, during a program algorithm, DQ7 is set to the opposite value of the programmed DQ7 bit. DQ6 toggles. That is, DQ6 changes value every read access. [Table 1](#) shows the operation of each bit during erase and program algorithms.

Table 1. DQ Polling

| Op.          | DQ7    | DQ6    | DQ5   | DQ4 | DQ3                            | DQ2    | DQ1               | DQ0 |
|--------------|--------|--------|---|-----|--------------------------------|--------|-------------------|-----|
| Chip Erase   | '1'    | Toggle | '1' if the device times out before the erase algorithm is ended | NA  | NA                             | NA     | NA                | NA  |
| Sector Erase | '1'    | Toggle | '1' if the device times out before the erase finishes           | NA  | Sector Erase Timeout Indicator | Toggle | NA                | NA  |
| Program      | -DQ[7] | Toggle | NA  | NA  | NA                             | NA     | NA Buffer Aborted | NA  |

### 3.2 Status Register

The status is an 8-bit register that indicates the status of the most recent program or erase operation at the sector of interest. The Status Register is accessed by writing a “Status Register Read” command and performing a read access. The status register is shown in Table 2. Bit 7 indicates whether the algorithm is active or finished. Note that most of the rest of the bits have no meaning when bit 7 is '1'.

Table 2. Status Register

| Bit 7                                  | Bit 6                      | Bit 5                | Bit 4                  | Bit 3  | Bit 2                        | Bit 1                                  | Bit 0                                    |
|--|----------------------------|----------------------|------------------------|--|------------------------------|--|--|
| Prog/Erase                             | Erase Suspend              | Erase                | Program                | Write Buffer Abort                               | Program Suspend              | Sector Lock                            | Bank                                     |
| 0 – Device busy programming or erasing | NA                         | NA                   | NA                     | NA   | NA                           | NA                                     | 0 – Operation in Addressed Bank          |
|  |                            |                      |                        |  |                              |  | 1 – Operation in Other Bank              |
| 1 – Device Ready                       | 0 – No Erase in Suspension | 0 – Erase Successful | 0 – Program Successful | 0 – Write Buffer Successful                      | 0 – No Program in Suspension | 0 – Sector not Locked during operation | 0 – No Active Program or Erase Operation |
|  | 1 – Erase in Suspension    | 1 – Erase Error      | 1 – Program Error      | 1 – Write Buffer Aborted due to Illegal Sequence | 1 – Program in Suspension    | 1 – Sector Locked Error                | 1 – Invalid                              |

### 3.3 No Transitional States with DQ Polling

With the DQ polling scheme, devices internally multiplex the data pins between the memory array and the DQ status. When an operation begins, the multiplexer switches the data pins to output the status, and they are switched back to read array mode after the operation completes. As the pins transition from status to read array mode, it is possible to actually have data and status mixed together at the output since the internal delay paths for the individual DQ bits are different. If a CPU read occurs during this transition, what will be read is a mix of status and data. This can lead software to conclude an error condition if the precaution is not taken to re-read the DQ bits to verify that the data pins actually became stable and the device returned to read array mode.

With the new register-based scheme, status information will be latched internally at appropriate times to ensure that only valid status is presented to the software. By design, this eliminates the chance of concluding the wrong status as compared to the DQ method.

The toggle method requires the system to do multiple reads for a comparison, and as stated above, extra reads are necessary to account for transitional states. Once a valid polling read is done, very often, the system must look at multiple bits to reach the correct conclusion about device state. For example, a write buffer abort error condition is indicated by the combination of the states of four bits: DQ7, DQ6, DQ2, and DQ1.

The use of interleaved memory configurations, where two devices are used to produce a wider data bus, further complicates status determination. Situations in which one device may stop toggling before the other have to be considered carefully.

### 3.4 Simpler Code

All of the factors involved with Data Polling lead to complicated algorithms to determine status with DQ polling. The status register makes determining status much simpler. The system simply sends a status read command followed by a read access. Multiple reads to determine status are no longer necessary, neither are complicated algorithms required to detect interleaved toggling.

To illustrate the complexity of implementing the DQ polling algorithm, consider the case of the Cypress low-level driver, where the code to determine the status with the DQ method took as much as forty lines of complicated logic. With the status register method, implementing the same function takes seven lines of much simpler logic.

### 3.5 A Bit for All States

There is no way to tell the status of a device when it is program suspended using the traditional DQ polling method. In fact, when a device is program suspended, reading from any location on the device, including the suspended bank or sector, returns read array data since there is no toggling of any of the DQ bits. This inadequacy of DQ polling status can pose serious problems for systems that have to rely on suspending program operations to service interrupts. One work around that has been used in drivers is to use global variables to track the device status, but this has the potential problem of creating a mismatch between the state of the global variable and the true state of the device.

The status register design, however, has a bit dedicated to all the (external) states that device can be in, including program suspend. Therefore, the developer can simply query the register to see if the device is program suspended and the problems described above do not apply.

### 3.6 Status Available Anytime, Anywhere

Another drawback of the combination of data polling and toggling design is that the device status is not retained when an embedded operation completes since the devices automatically revert to read array mode. The software must assume that the operation was successful. In addition, it has to also know the address of the bank or sector in which the operation is ongoing to poll for status. Checking the status in a different bank would simply return read array data, giving the requestor no clue that there was actually an operation going on in some other bank.

When two executables are running, this becomes a potentially dangerous situation because one executable may try to start a second disallowed embedded operation, since it has no way of knowing that the device is actually busy somewhere else.

These time and spatial ambiguities are eliminated with the status register design since checking status is neither time dependent nor is it tied solely to a specific address region. One can query the device at any time to get the status at that given time. The system can also query it at any location within the device and expect to get the correct status of that address.

Consider the example of a dual bank device in which Bank A is erase suspended and a task wants to initiate an erase in Bank B. When the task reads bank B with the toggling scheme, it will get read array data back, and get no indication that the device is actually erase suspended in the other bank. With this misinformation, the task can mistakenly start a second erase operation in Bank B, which would be disastrous. If the status register method is used, reading status from Bank B would indicate that the device is erase suspended elsewhere. The task could then resume the operation in Bank A, wait for it to complete, and then come back to Bank B to initiate the erase.

### 3.7 Knowledge of Specific Operation Not Necessary

Another drawback of the toggling method is that in some cases, the developer has to know beforehand what embedded operation he is polling status for. This is because some DQ bits can have different or no meaning, depending on the type of operation. For example, the DQ1 bit has a different meaning when a write buffer operation is taking place than when an erase operation is going on. This makes it difficult to asynchronously check for status in software, when the embedded operation commands are not immediately followed by polling software.

This ambiguity is removed with the status register design, because querying it will simply return the current status of the device at that time. Each bit in the status register also indicates the same meaning regardless of when it is queried.

### 3.8 Easier Design and Verification

The DQ polling method, in which toggling is the fundamental method of determining if an embedded operation is ongoing, is complicated and itself prone to errors. In the past, there were cases where the toggle method has caused software to reach the wrong conclusion about status. Because of an error in design coupled to toggling, XIP reads to a different bank, sandwiched between polling reads to the busy bank, were actually causing the device state machine to toggle internally. As two successive polling reads to the busy bank did not show any toggling, the software concluded that an operation had already completed, when in fact the device was busy.

Because of the complicated design, its verification has also been arduous. Thanks to the simple register based design, verification and final test times of products will also decrease, leading to lower costs and more robust products.

### 3.9 Seamless Migration Path

Transitioning to register-based polling also assures a seamless migration path to Cypress's revolutionary MirrorBit® Eclipse™ architecture based devices that promise lower costs and superior performance. Cypress devices at 45 nm will be based on this architecture that features only status registers, and no DQ polling.

## 4 Software Tools

Cypress provides a variety of software tools to aid the customer in integrating Cypress Flash memory into their products. Among them are the Low Level Driver (LLD) and a software interface layer, as discussed below.

### 4.1 Cypress Low Level Driver (LLD)

The Cypress Low Level Driver (LLD) is a production-grade driver toolbox that manages command initiation and polling operations for the full range of Cypress memory devices featuring MirrorBit® and floating gate memory technologies. The LLD exports an API that enables access to all the capabilities of the device. Among the API functions available are those that allow a developer to do write buffer programming, suspend/resume for program and erase, and advanced sector protection.

Cypress strongly recommends that customers use the LLD for their projects, as it provides a tested, proven interface. By abstracting all the details of command sequences and status queries, it enables developers to quickly integrate Cypress devices into their system, and focus on designing their application rather than spend time on learning details of Flash operation. Furthermore, developers can be assured of continued support through LLD updates if new features are added to any device.

### 4.2 DQ Polling and Status Register Interface Layer

As described above, using the status register is the preferred method for querying the device for status. However, for developers that may not be able to change to the new register-based polling, Cypress can provide a layer that provides an interface between the status register and customer DQ polling based routines. This would enable customers to keep the DQ polling software intact by inserting this layer that would convert the DQ polling queries to status register commands and responses.

## 5 Conclusion

Cypress is migrating to the new register-based model of determining status so that customers can enjoy a more reliable and simpler way of determining device status. The status register design addresses the key disadvantages of the DQ polling method. By using the LLD, customers can shorten their development times.

## Document History Page

Document Title: AN201783 - Benefits of Register-Based Polling over Data Polling

Document Number: 002-01783

| Rev. | ECN No. | Orig. of Change | Submission Date | Description of Change       |
|------|---------|-----------------|-----------------|-----------------------------|
| **   | –       | –               | 12/10/2007      | Initial version             |
| *A   | 5009215 | MSWI            | 11/10/2015      | Updated in Cypress template |
| *B   | 5846454 | AESATMP8        | 08/08/2017      | Updated logo and Copyright. |

## Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at [Cypress Locations](#).

### Products

|                               |  |
|-------------------------------|--|
| ARM® Cortex® Microcontrollers | <a href="http://cypress.com/arm">cypress.com/arm</a>               |
| Automotive                    | <a href="http://cypress.com/automotive">cypress.com/automotive</a> |
| Clocks & Buffers              | <a href="http://cypress.com/clocks">cypress.com/clocks</a>         |
| Interface                     | <a href="http://cypress.com/interface">cypress.com/interface</a>   |
| Internet of Things            | <a href="http://cypress.com/iot">cypress.com/iot</a>               |
| Memory                        | <a href="http://cypress.com/memory">cypress.com/memory</a>         |
| Microcontrollers              | <a href="http://cypress.com/mcu">cypress.com/mcu</a>               |
| PSoC                          | <a href="http://cypress.com/psoc">cypress.com/psoc</a>             |
| Power Management ICs          | <a href="http://cypress.com/pmic">cypress.com/pmic</a>             |
| Touch Sensing                 | <a href="http://cypress.com/touch">cypress.com/touch</a>           |
| USB Controllers               | <a href="http://cypress.com/usb">cypress.com/usb</a>               |
| Wireless Connectivity         | <a href="http://cypress.com/wireless">cypress.com/wireless</a>     |

### PSoC® Solutions

[PSoC 1](#) | [PSoC 3](#) | [PSoC 4](#) | [PSoC 5LP](#) | [PSoC 6](#)

### Cypress Developer Community

[Forums](#) | [WICED IOT Forums](#) | [Projects](#) | [Video](#) | [Blogs](#) | [Training](#) | [Components](#)

### Technical Support

[cypress.com/support](http://cypress.com/support)

All other trademarks or registered trademarks referenced herein are the property of their respective owners.



Cypress Semiconductor  
198 Champion Court  
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2007-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1s) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit [cypress.com](http://cypress.com). Other names and brands may be claimed as property of their respective owners.