AN218684

HyperBus™ Memory: Guide to Efficient Data Access

Author: Tsuyoshi Hikake

Associated Part Families: S26KL/S-S & S27KL/S-1

HyperBus family of devices is Cypress's first high-performance x8 input/output serial memory products - HyperFlash and HyperRAM. This application note describes, for software programmers and system engineers, methods to efficiently access HyperBus memory through Cypress's HyperBus Memory Controller.

Contents

1 Introduction

HyperBus Memory Controller IP provides the capability to control HyperBus Memory by integrating onto advanced extensible interface (AXI) slave interface. The basic functionality of the HyperBus Memory Controller is to receive AXI transaction requests from the AXI master and initiate HyperBus Memory operations.

AN218684 introduces the recommended methods for software engineers to access HyperBus Memory efficiently through HyperBus Memory Controller. The methods in this application note are based on HyperBus Memory Controller behavior.

2 HyperBus Memory Controller Behavior Overview

HyperBus Memory Controller provides two AXI Access Ports, which are composed of a *Memory Access Port* to access HyperBus Memory, and a *Control Register Access Port* to access the control register of the HyperBus Memory Controller.

The Memory Access Port supports AXI4 and AXI3 protocols with burst-based transactions. The Control Register Access Port supports the AXI4-Lite protocol with burst length always as 1 (i.e., register-style interface). This document focuses on the method to access HyperBus Memory through the Memory Access Port. In addition, since the HyperBus Memory Controller has the True Continuous Read Operation (TCR) for HyperFlash, this document provides details on the access method for HyperFlash. However, access to HyperRAM is also optimized by the access method written in this application note excluding TCR. All values of Control Register are expected to be their defaults.

2.1 Read Operations

The HyperBus Memory Controller initiates a read operation with the data size of the AXI read transaction requested from the bus master (CPU, DMA, and so on). The AXI read transaction and HyperBus read operation have a one-to-one relationship. The data size of the AXI read transaction is calculated as:

Data Size = (ARSIZE x ARLEN)

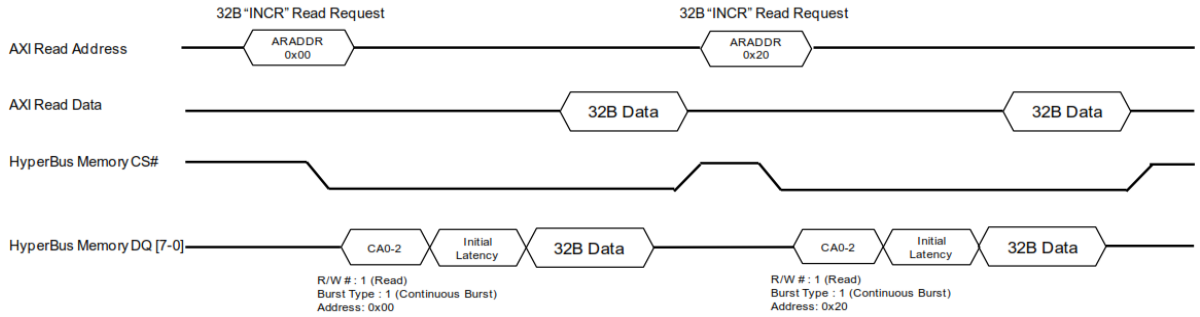
where

ARSIZE is the AXI read burst size. This signal indicates the size of each transfer in the burst.

ARLEN is the AXI read burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.

Figure 1 shows two 32-B read operations. The HyperBus Memory Controller receives two AXI read transactions and then initiates two HyperBus read operations.

Figure 1. Two 32B HyperBus Read Operations

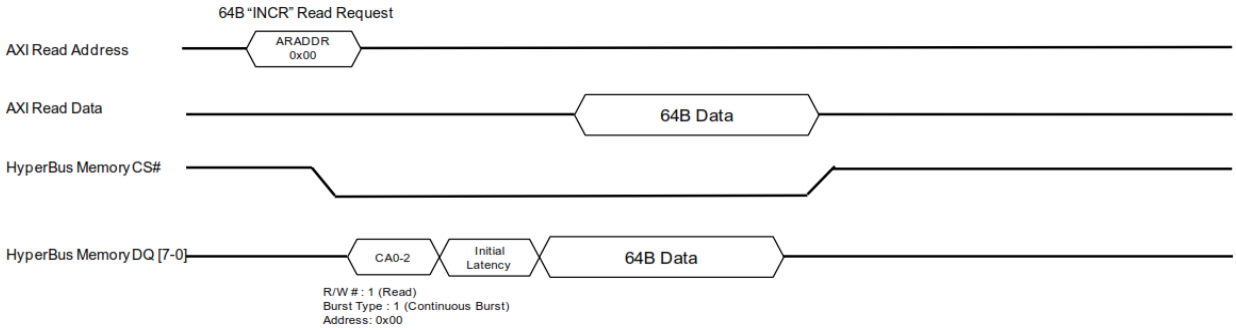


AXI Transaction Data Size: (ARSIZE x ARLEN): 2 (32-bit) x 7 (8) = 32B

HyperBus Read Transaction: 16-bit x 16 = 32B

Figure 2 shows a single 64-B read operation. The HyperBus Memory Controller receives a single AXI read transaction and then initiates a single HyperBus read operation.

Figure 2. Single 64B HyperBus Read Operation



AXI Transaction Data Size: (ARSIZE x ARLEN): 2 (32-bit) x 15 (16) = 64B

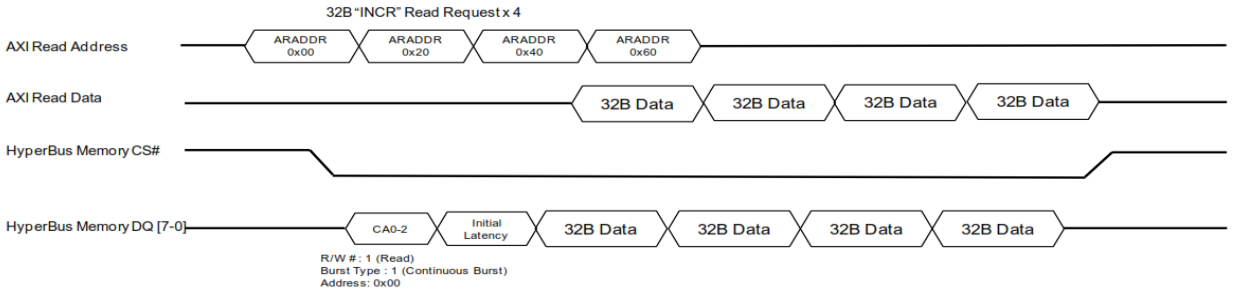
HyperBus Read Transaction: 16-bit x 32 = 64B

2.2 True Continuous Read Operation (TCR)

The HyperBus Memory Controller supports True Continuous Read operation for HyperFlash. This feature improves the HyperFlash read performance. If the HyperBus Memory Controller fetches AXI read transaction requests with the subsequent addresses before the current HyperFlash read operation is completed, the HyperBus Memory Controller continues to assert CS# and read data from HyperFlash for the next AXI read transaction. The required conditions for this feature are:

1. Multiple outstanding read transactions with subsequent addresses
2. AXI Burst Type is "INCR" (ARBURST:1)

Figure 3. TCR Operation



A TCR operation is able to merge multiple AXI read transactions to a single HyperFlash read operation. As a result, the command input and initial latency for the following AXI read transactions are saved, which leads to improve the throughput.

Note: TCR Operation is not supported in HyperRAM.

2.3 Write Operations

The HyperBus Memory Controller initiates a write operation with the data size of the AXI write transaction requested from the bus master (CPU, DMA, and so on). The AXI write transaction and HyperBus write operation have a one-to-one relationship. The data size of the AXI write transaction is calculated as:

Data Size = (AWSIZE x AWLEN)

where

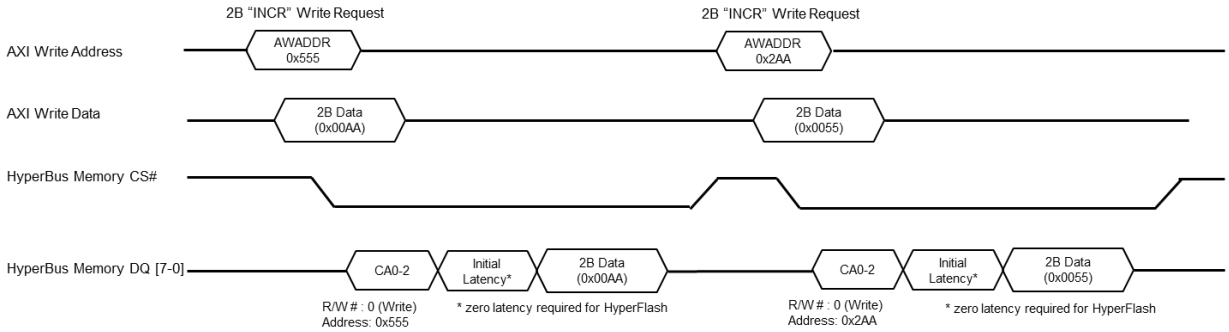
AWSIZE is the AXI write burst size. This signal indicates the size of each transfer in the burst.

AWLEN is the AXI write burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.

It is important to know that the write operation to HyperFlash is all non-burst write (asynchronous write) in the sense of sending flash "command" sequence except single word programming where data bytes can be sent to HyperFlash in burst (synchronous write) manner. See 4.3 for more details.

Figure 1 shows two 2B write operations. The HyperBus Memory Controller receives two AXI write transactions and then initiates two HyperBus write operations.

Figure 4. Two 2B HyperBus write Operations



AXI Transaction Data Size: (AWSIZE x AWLEN): 1 (16-bit) x 0 (1) = 2B

HyperBus Write Transaction: 16-bit x 1 = 2B

3 Efficient Data Access

From HyperBus memory's perspective, the best practice to read data is to initiate a single read operation, rather than multiple read operations. This maximizes the data portion of the operation and minimizes the command and latency overheads.

From HyperBus Memory Controller's perspective, the following points must be considered for an optimized HyperBus Memory read access:

1. Maximize the data size of the AXI transaction for each AXI read request
2. Generate the conditions for the TCR operation for subsequent AXI read requests

3.1 Efficient Data Access Methods

As discussed above, for efficient data access, it is important to understand the data size capability for read/write from the system as well as the device perspective. The information in this section is based on a HyperBus memory controller reference design implemented on a Xilinx Zynq ZedBoard platform (CPU: ARM® Cortex®-A9MP, DMA: ARM PL330, Protocol: AXI3) and a HyperFlash memory device. Table 1 lists the relationships between them.

Table 1. Device Operation Size on Zynq-7000 ZedBoard (AXI3)

Access	System Configuration		Data Size of HyperBus Memory Operation		
	Bus Master	I/D-Cache	Max. with TCR	Maximum	Minimum
XIP	CPU	On	1GB	32B	32B
		Off	32B	32B	32B
Data Read	CPU	On	64B/4KB[1]	32B	32B
		Off	48B	48B	2B
	DMA	N/A	32KB	64B[2]	2B
Data Write	CPU	On	N/A	N/A	N/A
		Off	N/A	2B	2B
	DMA	N/A	N/A	32B[3]	2B

Notes:

[1] 64B is the case of ACLK: 166.7 MHz/CK: 166.7 MHz.
4 KB is the case of ACLK: 83.3 MHz/CK: 6 MHz.
[2] Theoretical size under AXI4 environment will be 1 KB.
[3] Theoretical size under AXI4 environment will be 512B, coming from HyperFlash Spec.

The access type on the system is classified as code eXecute-In-Place (XIP) on HyperBus Memory, data read from HyperBus Memory, and data write to HyperBus Memory. The system configuration is defined by a combination of bus master and cache usage. The data size of the HyperBus Memory operation is categorized as:

1. Max. with TCR (Applies only to HyperFlash)
   I. When a TCR Operation is executed, the data size of the HyperFlash operation is larger than that of the AXI transaction request.
   II. The condition to enable a TCR operation depends on the throughput of the AXI interface (AXI CLK) and HyperBus memory interface (CK/CK#). In general, a higher throughput of the AXI interface than the HyperBus memory interface is required for a TCR operation.

   Note: Values in the row are based on the best-case condition for the interface throughput.
2. Minimum / Maximum:
   I. This is the case of General Read/Write Operations. The data size of the HyperBus Memory operation is equal to that of the AXI transaction request. Thus, the values in these rows are equivalent to the data size capability of the AXI transaction.

3.1.1 eXecute-In-Place (XIP)

CPU Access Recommendation: CPU Access /w I-Cache
With I-Cache enabled, the CPU pre-fetches the code as multiple outstanding read transactions with subsequent address, which allows the HyperBus Memory Controller to execute a TCR operation.

Note: See code examples in Section 4.

3.1.2 Data Read

CPU Access /w D-Cache Recommendation
CPU access with D-Cache is recommended for accessing small chunks of data (less than 32B) in a random access manner.

DMA Access Recommendation
DMA access is recommended for large chunks of data (greater than 32B) when accessed in a sequential access manner.

These accesses support TCR operation. If read access to HyperBus Memory is dominant on your system, it is recommended to use either CPU Access with the D-Cache, or DMA Access.

In general, DMA access takes more time for initialization than CPU access; however, DMA access is able to read a larger data in a single operation than CPU access. As an original design intention, DMA access is able to release CPU resources during data transfer. This trade-off needs to be considered for efficient data reads.

Note: See code examples in Section 4.

CPU Access without D-Cache Recommendation (Register Read in HyperFlash)
CPU access without D-Cache is recommended for accessing the status or configuration registers.

Note: See code examples in Section 4.

3.1.3 Data Write

To program data onto a HyperFlash memory array, an embedded programming command sequence, which is composed of write operations, is required. The write operation of command sequence is required in the same manner as an asynchronous write (single write) operation.

The write operation of programming data has a requirement depending on the method of embedded programming shown as below.

- Write to Buffer Programming: In the same manner as an asynchronous write operation. Each single word address/data must be written.
- Word Programming: In the same manner as a synchronous write (burst write) operation. 1 to 256 words burst data can be written.

Both embedded programming methods are able to program up to 256 words data with a single embedded programming command sequence.

CPU Access without D-Cache Recommendation
CPU access without D-Cache is recommended for command sequence and Write to Buffer programming.

DMA Access Recommendation
DMA access is recommended for Word Programming.

Note: See code examples in Section 4.

4 Coding Examples (Zynq Platform)

Software implementation examples and tips are shown for efficient data access on Xilinx Zynq 7000 ZedBoard. It is important to note that Zynq 7000 SOC is based on the AXI3 protocol standard and supports up to 16-beat burst length (ARLEN/AWLEN).

4.1 eXecute-In-Place (XIP)

This section describes tips on how to enable TCR operations on Zynq 7000 for XIP access to HyperFlash with I-Cache.

Figure 5 shows a waveform of XIP with the TCR operation for the ideal case. Figure 6 shows a memory map of HyperFlash and associated I-Cache. These are based on the assumption that the code is executed in order and without branching. Timelines t1-t5 are consistent between Figure 5 and Figure 6.

When an I-Cache miss occurs on the memory mapped to HyperFlash, CPU requests the AXI read transaction to Memory Access Port for filling the code to I-Cache. The data size of a single AXI read transaction for code fetch and pre-fetch is 32B (ARSIZE: 2 (32-bit) x ARLEN: 7 (8)), which is aligned with I-Cache line size.

When the CPU fetches the code from HyperFlash, four outstanding AXI read transactions with the subsequent address (128B = 32B x 4) are requested to the Memory Access Port (Timeline t1). Then, a single AXI read transaction with the subsequent address is requested after the data of the first AXI read transaction is read out to the AXI bus (Timeline t2). Then, the same sequence at Timeline t2 is executed continuously. Timelines t3-t5 show ARM Cortex-A9 code pre-fetch scheme on Zynq.

Based on the above AXI sequence, next four AXI read transactions are always requested in advance as code pre-fetch. As long as the code is executed in order without branching, this AXI sequence is continued and the TCR operation of HyperFlash is also continued as a single read operation.

In case of code branching, if the execution address (PC) is in the current cache line or next four cache lines (160B = 32B x 5), then the AXI sequence with TCR will continue. In summary, the condition to enable and continue TCR operation is to have the code execute within five cache lines (160B) subsequently mapped onto HyperFlash.

Figure 5. Waveform of XIP /w I-Cache Ideal True Continuous Read Operation

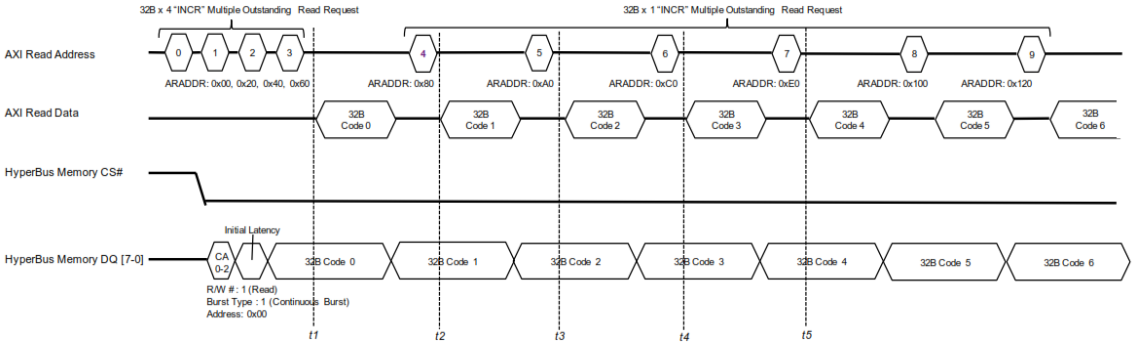
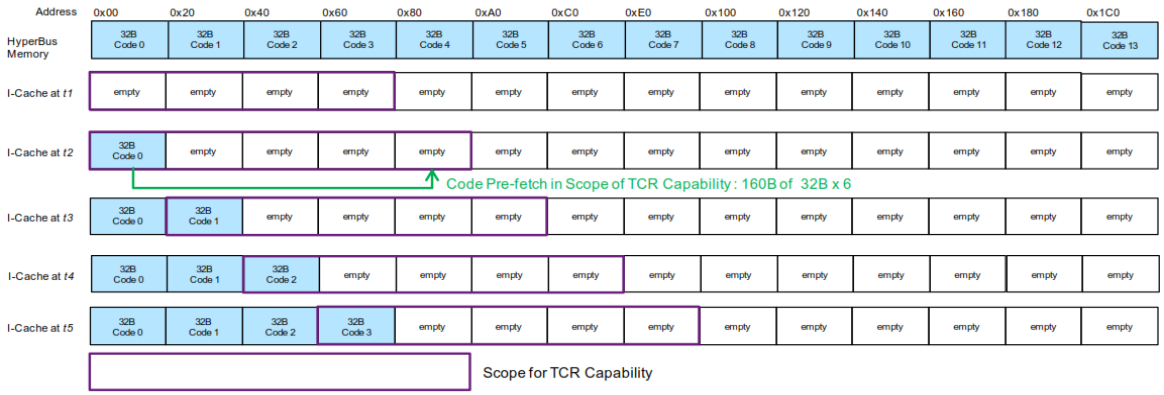


Figure 6. I-Cache State of XIP /w I-Cache Ideal True Continuous Read Operation



For the best-case scenario, AXI multiple outstanding read transactions must be operated as follows:

t1: The start of XIP from Code 0, read request 128B of Code0-3
t2: Read Code 0, read request Code 4 (Next code execution address scope is in Code 0-4)
t3: Read Code 1, read request Code 5 (Next code execution address scope is in Code 1-5)
t4: Read Code 2, read request Code 6 (Next code execution address scope is in Code 2-6)
t5: Read Code 3, read request Code 7 (Next code execution address scope is in Code 3-7)
In case of a code branch, if the execution address (PC) is out of the current cache line and the next four cache lines (160B = 32B x 5), the TCR operation is aborted. Figure 7 shows the XIP TCR operation waveforms for the abort case. Figure 8 shows a memory map of HyperFlash and associated I-Cache. Timelines t1-t5 are again consistent between Figure 7 and Figure 8.

When the code is jumped to out of 160B (Timeline t2), the outstanding three AXI read transactions are continuously executed as a TCR operation. This is a known penalty of the pre-fetch scheme regardless of the TCR operation. Once the read operation of these AXI read transactions are completed (Timeline t3), four AXI read transactions (one for fetch from jumped address and three for prefetching) are requested, followed by the TCR operation.

Figure 7. Waveform of XIP /w I-Cache Shoeing TCR Operation Abort

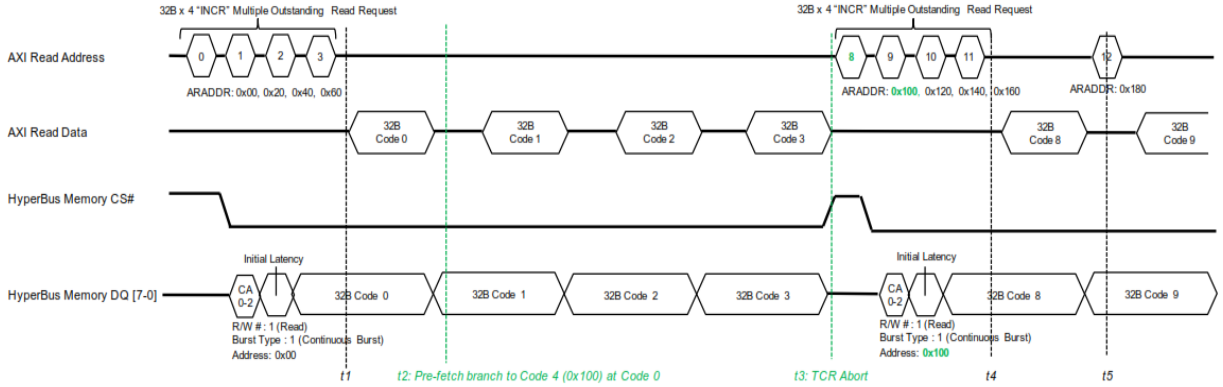
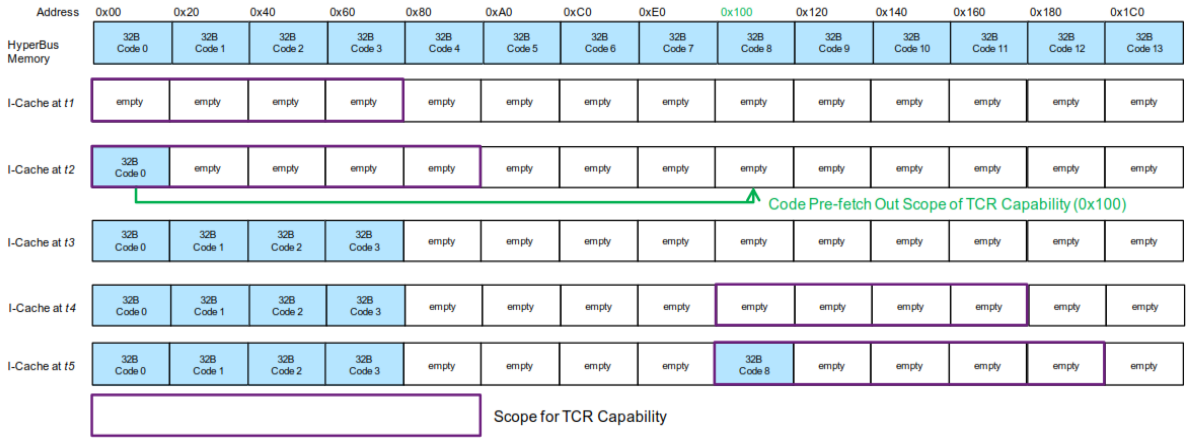


Figure 8. I-Cache State of XIP /w I-Cache Showing TCR Operation Abort



For the abort case, AXI read transactions are operated as follows:

t1: The start of XIP from Code 0, read request 128B of Code 0-3
t2: Read Code 0, (Next code execution address is Code 8 (256B offset) that is out of Code 0-4)
t3: Read Code 1-3
t4: The start of XIP from Code 8, read request 128B of Code 8-11
t5: Read Code 8 (Next code execution address scope is in Code 8-12)

4.2 Data Read

This section describes tips on how to enable TCR operations on Zynq 7000 for Data Read accesses, CPU Accesses with D-Cache, and DMA accesses to HyperFlash.

4.2.1 CPU Access with D-Cache

When a D-Cache miss occurs on the memory mapped to HyperFlash, the CPU requests an AXI read transaction to Memory Access Port for filling the data to D-Cache. The data size of a single AXI read transaction for data read is 32B (ARSIZE: 2 (32-bit) x ARLEN: 7 (8)), which is aligned with the D-Cache line size.

The D-Cache scheme is mainly classified as Write-Through or Write-Back. Only Write-Back supports multiple outstanding AXI read requests. So, it is necessary to select Write-Back for TCR operation.

The following C code example is for a TCR operation. This code is to read a 4-KB chunk of data by unsigned long data types (8B) from HyperFlash.

```
#define RPC2_BASE_ADDR  (0x40000000) // HyperFlash Access Port Base Address
#define RPC2 _SEC_1_OFFSET (0x00040000) // SA1 (Sector Address)
…
unsigned int rdData_tbl [1024]; // Read data buffer
unsigned int rbuf_adr = RPC2 _SEC_1_OFFSET;
unsigned short loop_current;  // loop count
unsigned long long *rbuf_ptr = (unsigned short *)wtData_tbl; // Current write buffer
point = wtData_tbl[0];
…
// 4KB Read by 8B x 512
for (loop_current = 0; loop_current < 512 ; loop_current++) {
*(volatile unsigned long long *) (RPC2_BASE_ADDR + rbuf_adr + (loop_current * 8)) =
rbuf_ptr[loop_current];
}
```

4.2.2 DMA Access

Zynq 7000 ARM DMA controller PL330 is able to control the DMA transaction by Microcode that is defined by PL330 as a unique instruction set. The Microcode provides the capability to control the transaction at the AXI transaction level, which means that the software engineers are able to program Microcode by explicitly setting the values of AxSIZE, AxLEN, and so forth.

Supported AXI read transaction maximum data size on PL330 is:

64B: ARSIZE: 2 (32-bit) x ARLEN: 15 (x16)

It is recommended to program Microcode as a single AXI read transaction for less than 64B chunk of data.

Microcode also provides the capability to specify the loop count of a programmed AXI read transaction as subsequent address accesses. With the loop count instruction DMALP, a series of AXI read transactions is issued as multiple outstanding read AXI transactions, which enable True Continuous Read operation.

It is recommended to program Microcode by using the loop count feature for more than 64B chunk of data.

The following Microcode is an example for True Continuous Read operation. This code is to read 512B chunk of data by eight AXI multiple outstanding read transactions with 64B. (512B = 64B x 8)

```
# Setup for 16-beat 32-bit AXI transaction from source (HyperFlash) and destination
(Read Data Buffer)
# Set Source & Destination AXI transaction: ARSIZE, AWSIZE: 1(16bit) x ARSIZE, ARLEN:
15(16beat)
DMAMOV CCR, SB16 SS32 DB16 DS32
# Set Source Address by HyperFlash SA0
DMAMOV SAR, 0x40040000
# Set Destination Address by Read Buffer
DMAMOV DAR, 0x00200000
# Loop Count (Multiple Outstanding Request Count) = 8
DMALP 8
# Load data from Source (Write Buffer) by ARSIZE:1 x ARLEN:15
DMALD
# Store data to Destination (HyperFlash) by AWSIZE:1 x AWLEN:15
DMAST
# End of Loop
DMALPEND
DMAEND
```

Figure 9 is the AXI Read Transaction of 64B (32-bit x 16) x 8 AXI multiple outstanding read transactions.

Figure 9. RM PL330 DMA AXI Read Transaction

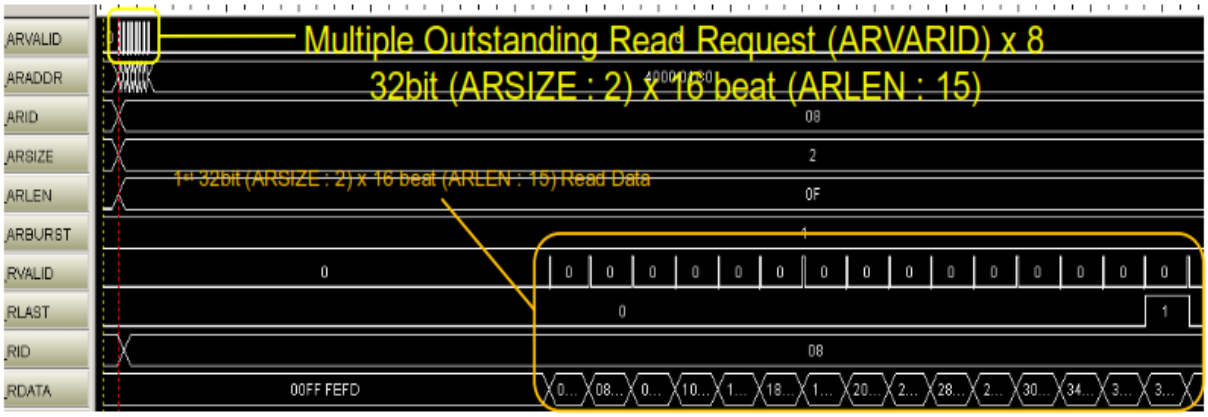
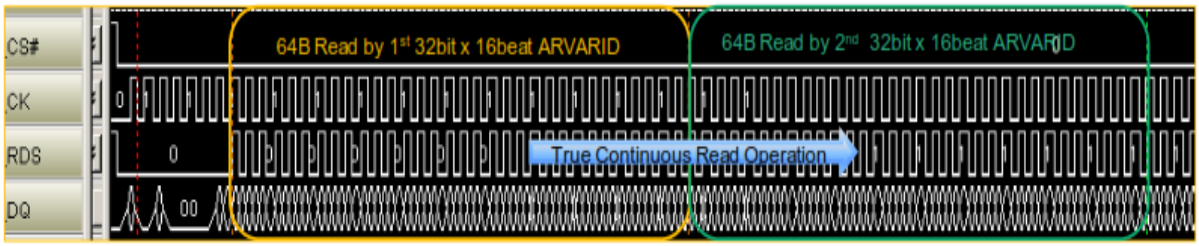


Figure 10. TCR by ARM PL330 DMA



4.3 Data Write

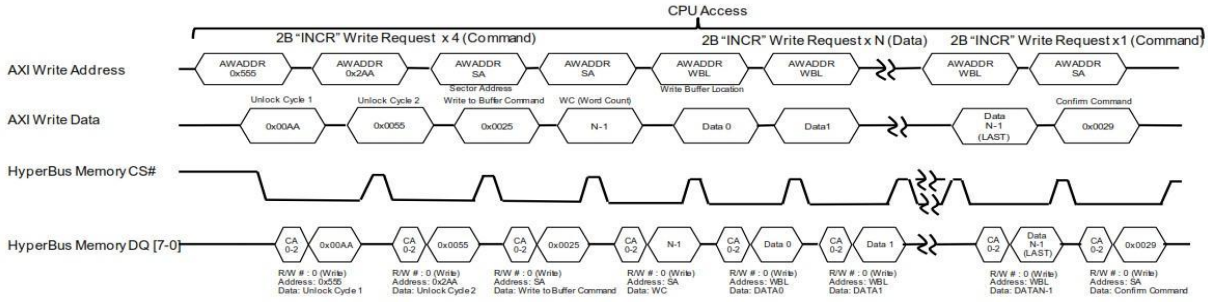
This section describes implementation example to program data onto HyperFlash memory array on Zynq 7000. To program data onto HyperFlash memory array, embedded programming command sequence, which is composed of write operations, is required. There are two methods of embedded programming, Write to Buffer Programming and Word Programming.

4.3.1 Write to Buffer Programming

For the Write to Buffer Programming operation, it is necessary to write all of the Write to Buffer command and programming data to HyperFlash as an asynchronous write operation with AWSIZE: 1 (16-bit) x AWLEN: 0 (1). In general, 'STRH' of ARMv7 single-word store instruction without the D-Cache condition meets this requirement.

After issuing the Program Buffer to Flash command, monitoring the Status Register is required to determine the status of the embedded programming operation until its completion. It is necessary to write the Status Register Read command and read the Status Register to/from HyperFlash as an asynchronous write/read operation.

Figure 11. Write to Buffer Programming Operation



The following C code example is for a Write to Buffer Programming operation. This code is to program a 512B chunk of data (256 words) with the Write to Buffer Programming operation.

```
#define RPC2_BASE_ADDR     (0x40000000) // HyperFlash Access Port Base Address
```

```
#define RPC2 _SEC_1_OFFSET       (0x00040000) // SA1 (Sector Address)
…
unsigned int wtData_tbl [128]; // Write data buffer
…
unsigned int wbuf_adr = RPC2 _SEC_1_OFFSET; // WBL Start Address = SA1
unsigned short wbuf_cnt = 256;                        // programdata count = 256 words
unsigned short loop_current;  // WBL offset
unsigned short *wbuf_ptr = (unsigned short *)wtData_tbl; // Current write buffer point
= wtData_tbl[0];
unsigned short rd_status = 0; // Read Status register value

// Send Write to Buffer program commands to HyperFlash
*(volatile unsigned short *)(RPC2_BASE_ADDR + (0x555 << 1)) = 0x00AA; // unlock cycle
1
*(volatile unsigned short *)(RPC2_BASE_ADDR + (0x2AA << 1)) = 0x0055; // unlock cycle
2
*(volatile unsigned short *)(RPC2_BASE_ADDR + wbuf_adr) = 0x0025; // SA <- Write to
Buffer command
*(volatile unsigned short *)(RPC2_BASE_ADDR + wbuf_adr) = wbuf_cnt -1 ; //  SA <-
Write Count -1
// Send 256 words programming data to HyperFlash
for (loop_current = 0; loop_current < wbuf_cnt ; loop_current++) {
*(volatile unsigned short *) (RPC2_BASE_ADDR + wbuf_adr + (loop_current * 2)) =
wbuf_ptr[loop_current];
}

// Send Write to Buffer programming commands to HyperFlash
*(volatile unsigned short *)(RPC2_BASE_ADDR + wbuf_adr) = 0x0029; // SA <- Write to
Buffer command

// Read Status Register and poll while bit7 = 0 (busy)
while (!(rd_status & 0x0080)) {
// send status register read command
*(volatile unsigned short *)(RPC2_BASE_ADDR + (0x555 << 1)) = 0x0070;
rd_status = *(volatile unsigned short *)(RPC2_BASE_ADDR); // read status register
// describes status register error check
        …
}
// data programming is completed and describes status register error check
…
```

4.3.2 Word Programming

For the Word Programming Operation, it is necessary to write all of the Word Programming commands to HyperFlash as an asynchronous write operation with AWSIZE: 1 (16-bit) x AWLEN: 0 (1). On the other hand, it is necessary to write all of the programming data to HyperFlash as an synchronous write (burst write) operation. The AXI write transaction for N words burst write is AWSIZE: 1 (16-bit) x AWLEN: N-1(N). It is recommended to use the DMA Controller ARM® PL330 to meet this requirement.

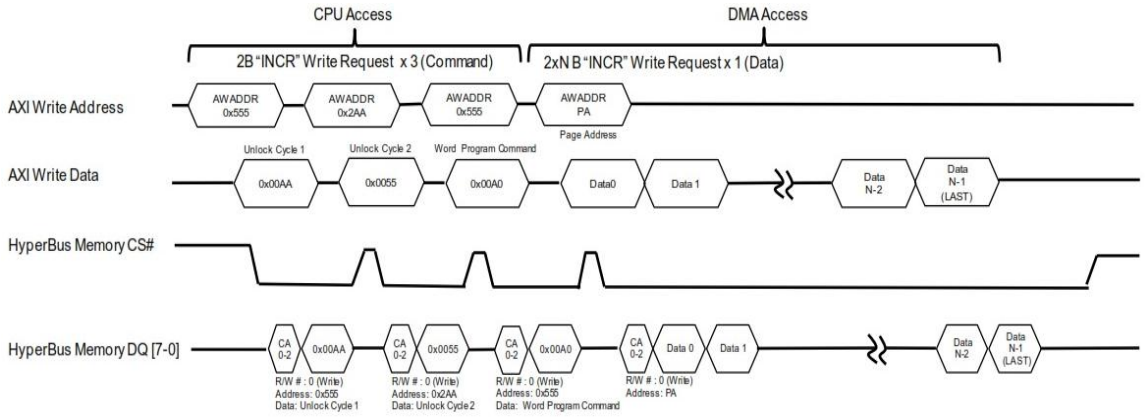
Note: Burst write during a Word Programming Operation supports up to 50 MHz of HyperBus memory interface clock per the HyperFlash device specification.

PL330 on Zynq 7000 is based on the AXI3 protocol standard, so that PL330 supports up to 16 words (AWSIZE: 1 (16-bit) x AWLEN: 15 (16)) as a single burst write operation. It means that the Word Programming operation is able to support up to 16 words of data programming on Zynq, even though up to 256 words are supported as HyperFlash device specification.

It is recommended to use the Write to Buffer Programming operation for more than 16 words chunk of data on Zynq 7000.

After loading the programming data, monitoring the Status Register is required to determine the status of the embedded programming operation until its completion. It is necessary to write Status Register Read command and read Status Register to/from HyperFlash in the same manner as a synchronous write/read operation.

Figure 12. Word Programming



The following source code example is for a Word Programming operation. This code is to program a 32B chunk of data (16 words) with the Word Programming operation.

```
unsigned short rd_status = 0; // Read Status register value
// Setup DMA Controller & Create Microcode (DMA driver)
…
// Send Word program commands to HyperFlash
*(volatile unsigned short *)(RPC2_BASE_ADDR + (0x555 << 1)) = 0x00AA; // unlock cycle
1
*(volatile unsigned short *)(RPC2_BASE_ADDR + (0x2AA << 1)) = 0x0055; // unlock cycle
2
*(volatile unsigned short *)(RPC2_BASE_ADDR + (0x555 << 1)) = 0x00A0; // word program
command
// Start DMA and Wait DMA transaction is completed (DMA driver)
…
// Read Status Register and poll while bit7 = 0 (busy)
while (!(rd_status & 0x0080)) {
// send status register read command
*(volatile unsigned short *)(RPC2_BASE_ADDR + (0x555 << 1)) = 0x0070;
rd_status = *(volatile unsigned short *)(RPC2_BASE_ADDR); // read status register
// describes status register error check
…
}
// data programming is completed and describes status register error check
…
```

The following Microcode is an example for a burst write of 16 words data programming.

```
# Setup for 16-beat 16-bit AXI transaction from source (Write Data Buffer) and
destination (HyperFlash)
# Set Source & Destination AXI transaction: ARSIZE, AWSIZE: 1(16bit) x ARSIZE, ARLEN:
15(16beat)
DMAMOV CCR, SB16 SS16 DB16 DS16
# Set Source Address by Write Buffer
DMAMOV SAR, 0x00100000
# Set Destination Address by HyperFlash SA1
DMAMOV DAR, 0x40040000
# Load data from Source (Write Buffer) by ARSIZE:1 x ARLEN:15
DMALD
# Store data to Destination (HyperFlash) by AWSIZE:1 x AWLEN:15
DMAST
DMAEND
```

5 Conclusion

This application note introduced the recommended methods for a software engineer to access HyperBus Memory efficiently through the HyperBus Memory Controller. The methods in this application note are on the basis of the HyperBus Memory Controller behavior. The examples shown are for Xilinx® Zynq® ZedBoard.

6 Related Documents

Table 2. Cypress HyperBus Product-Specific Datasheets

Product Family	Spec. Number	Title
KL/S-S Family	001-99198	512 Mbit (64 Mbyte)/256 Mbit (32 Mbyte)/ 128 Mbit (16 Mbyte), 1.8V/3.0V HyperFlash™ Family
KL/S-1 Family	001-97964	3.0 V/1.8 V, 64 Mbit (8 Mbyte)/128 Mbit (16 Mbyte), HyperRAM™ Self-Refresh DRAM

Document History

Document Title: AN218684 – HyperBus™ Memory: Guide to Efficient Data Access

Document Number: 002-18684

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	5646323	SZZX	03/08/2017	New Application Note.
*A	5849031	HARA	08/17/2017	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	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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