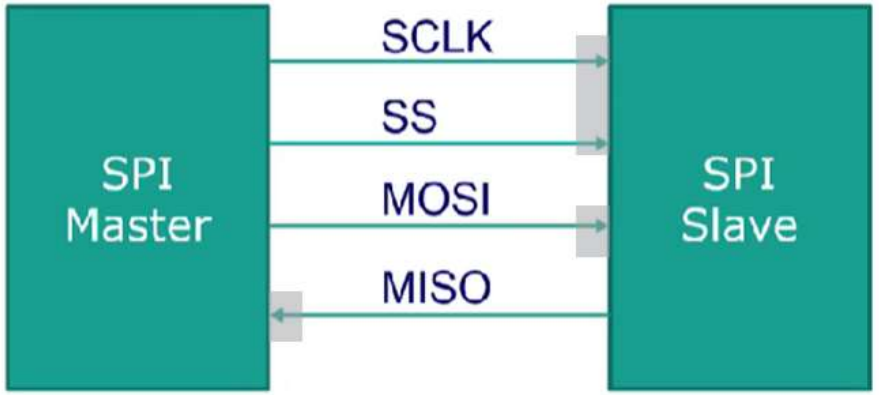
QSPI Flash Read/Write Test

AMD
together we advance

SPI Master



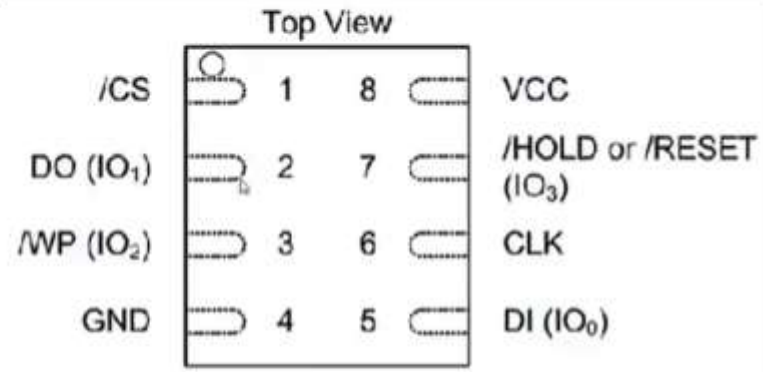
QSPI Flash

3.2 Pad Description WSON 8x6-mm

PAD NO.	PAD NAME	I/O	FUNCTION
1	/CS	I	Chip Select Input
2	DO (IO1)	I/O	Data Output (Data Input Output 1)[1]
3	/WP (IO2)	I/O	Write Protect Input (Data Input Output 2)[2]
4	GND		Ground
5	DI (IO0)	I/O	Data Input (Data Input Output 0)[1]
6	CLK	I	Serial Clock Input
7	/HOLD or /RESET (IO3)	I/O	Hold or Reset Input (Data Input Output 3)[2]
8	VCC		Power Supply

Notes:

1. IO0 and IO1 are used for Standard and Dual SPI instructions
2. IO0 – IO3 are used for Quad SPI instructions, /WP & /HOLD (or /RESET) functions are only available for Standard/Dual SPI.



" / " represent low active trigger

QSPI Flash controller

Introduction

The Quad-SPI flash controller is part of the input/output peripherals (IOP) located within the PS. It is used to access multi-bit serial flash memory devices for high throughput and low pin count applications.

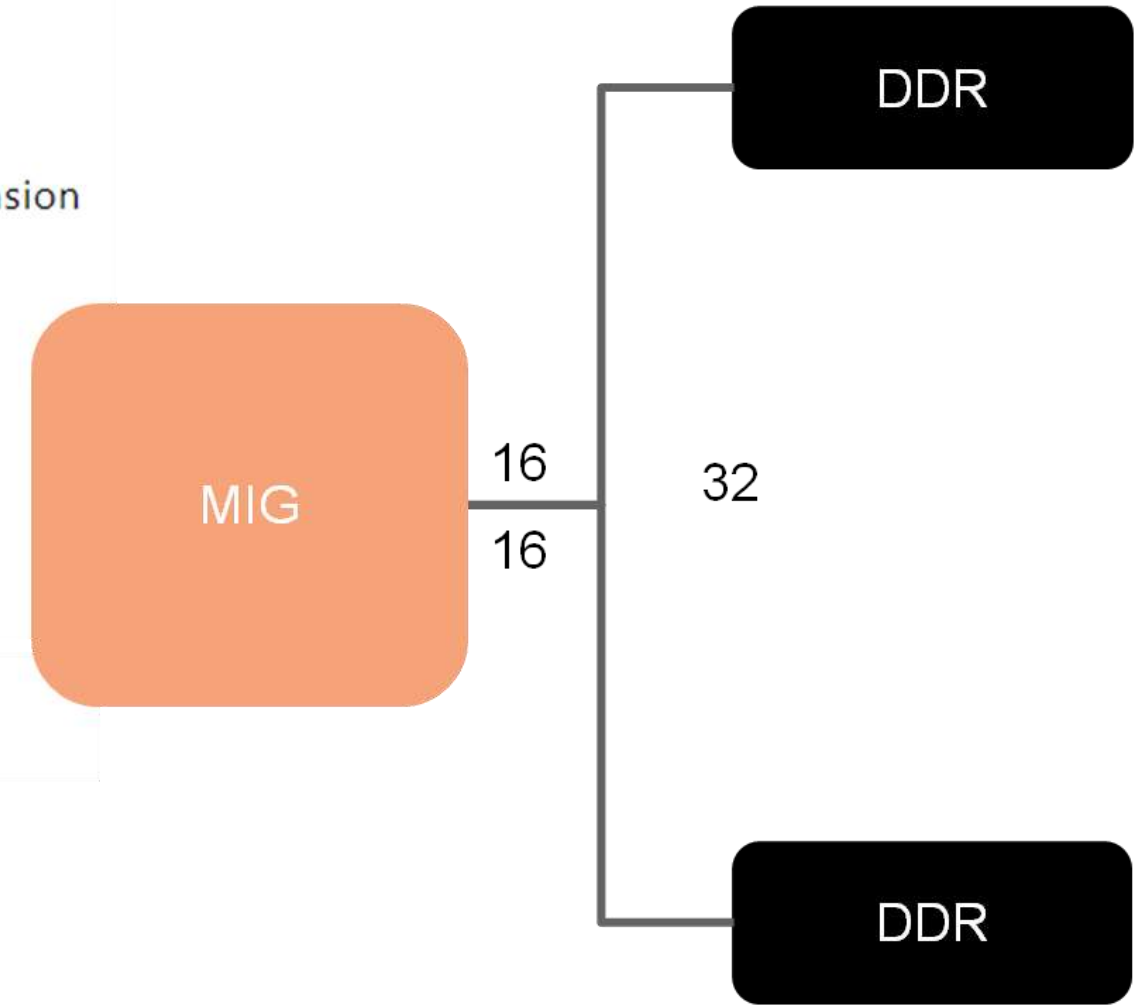
The controller operates in one of three modes: I/O mode, linear addressing mode, and legacy SPI mode. In I/O mode, software interacts closely with the flash device protocol. The software writes the flash commands and data to the controller using the four TXD registers. Software reads the RXD register that contains the data received from the flash device.

Linear addressing mode uses a subset of device operations to eliminate the software overhead that the I/O mode requires to read the flash memory. Linear Mode engages hardware to issue commands to the flash memory and control the flow of data from the flash memory bus to the AXI interface. The controller responds to memory requests on the AXI interface as if the flash memory were a ROM memory. In legacy mode, QSPI controller acts as a normal SPI controller.

The controller can interface to one or two flash devices. Two devices can be connected in parallel for 8-bit performance, or in a stacked, 4-bit arrangement to minimize pin count. The two device combinations are shown in Figure 12-1.

Features

- 32-bit AXI interface for Linear Addressing mode transfers
- 32-bit APB interface for I/O mode transfers
- Programmable bus protocol for flash memories from Micron and Spansion
- Legacy SPI and scalable performance: 1x, 2x, 4x, 8x I/O widths
- Flexible I/O
  - Single SS 4-bit I/O flash interface mode
  - Dual SS 8-bit parallel I/O flash interface mode
  - Dual SS 4-bit stacked I/O flash interface mode
  - Single SS, legacy SPI interface
- 16 MB addressing per device (32 MB for two devices)



System Viewpoint

The Quad-SPI flash controller is part of the IOP and connects to external SPI flash memory through the MIO as shown in Figure 12-1. The controller supports one or two memories.

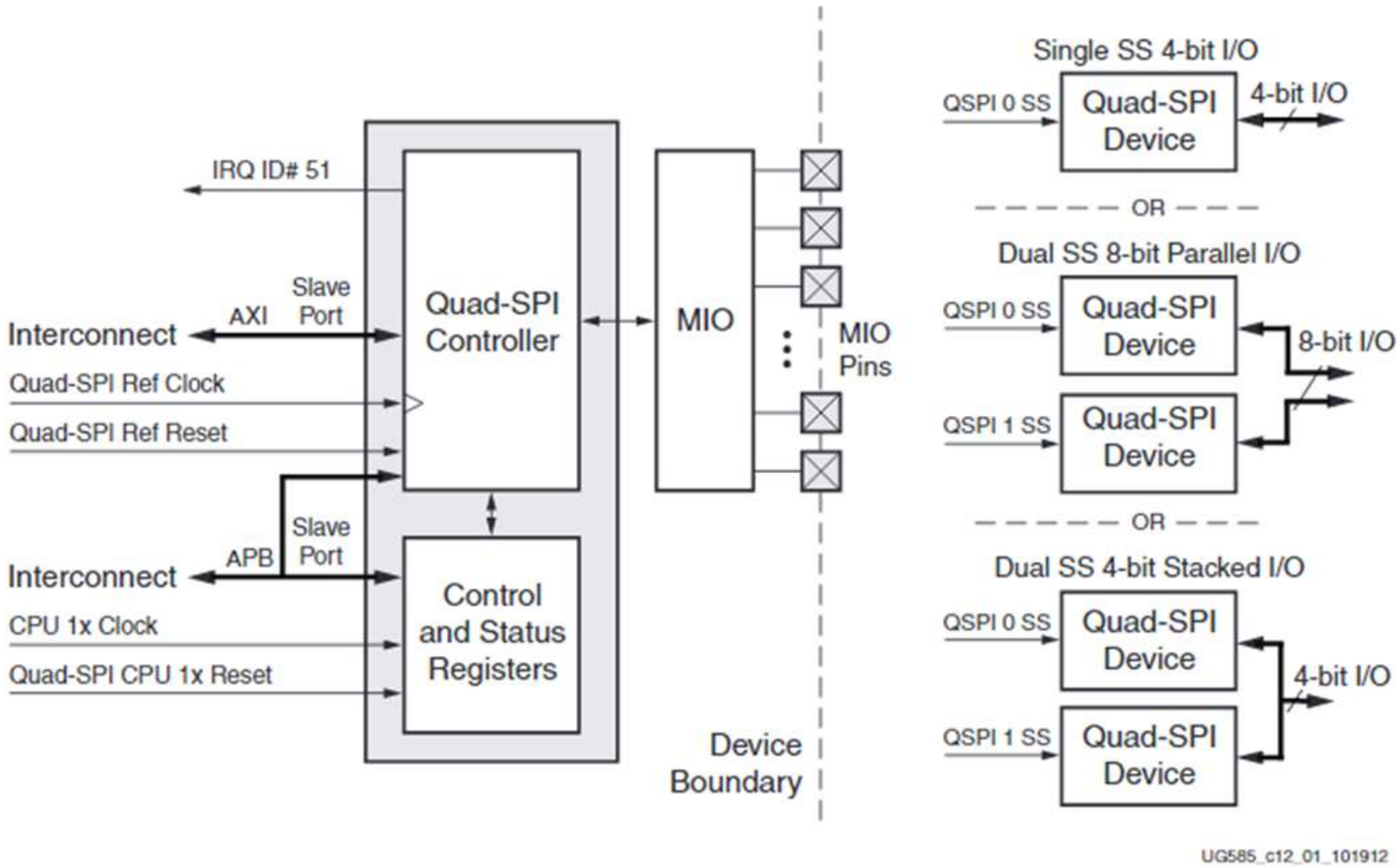


Figure 12-1: **Quad-SPI Controller System Viewpoint**

Block Diagram

The block diagram of the is shown in Figure 12-2.

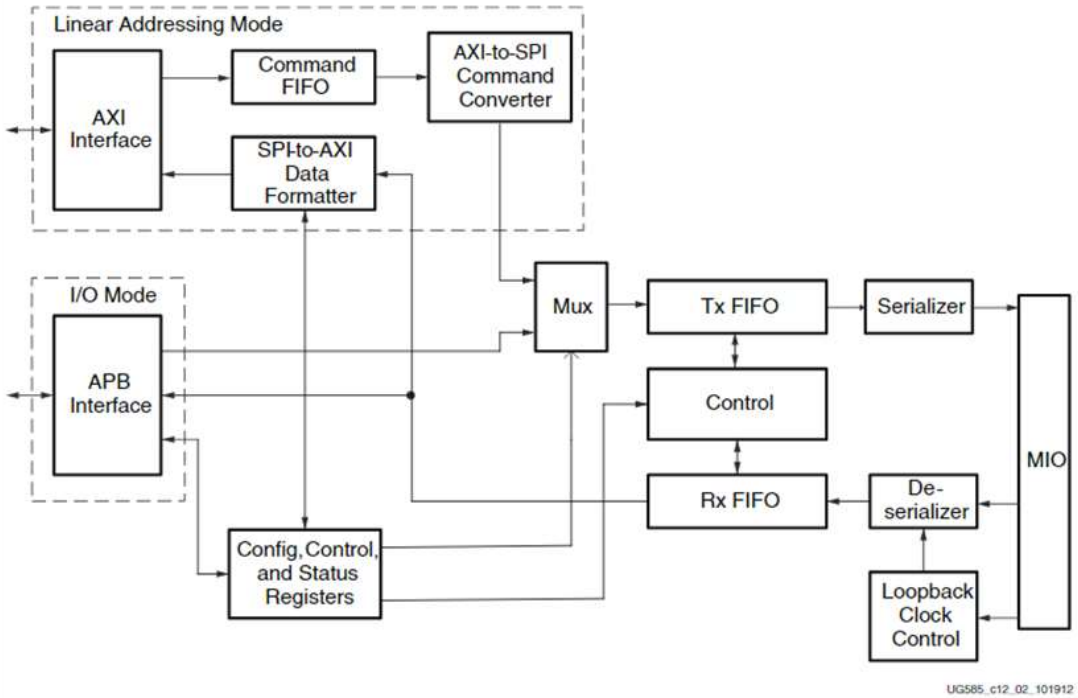


Figure 12-2: **Quad-SPI Controller Block Diagram**

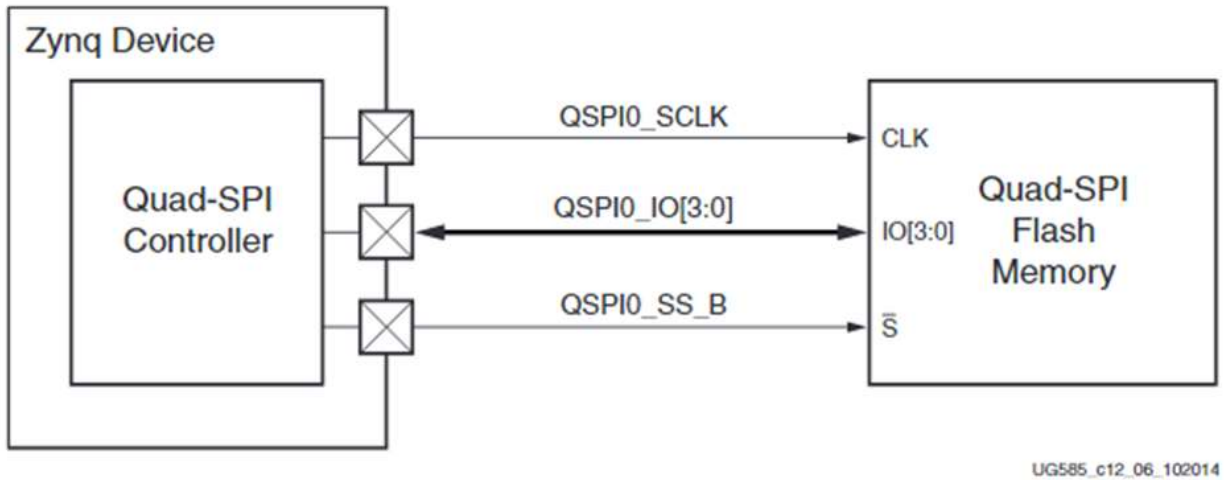


Figure 12-5: **Quad-SPI Single SS 4-bit I/O**

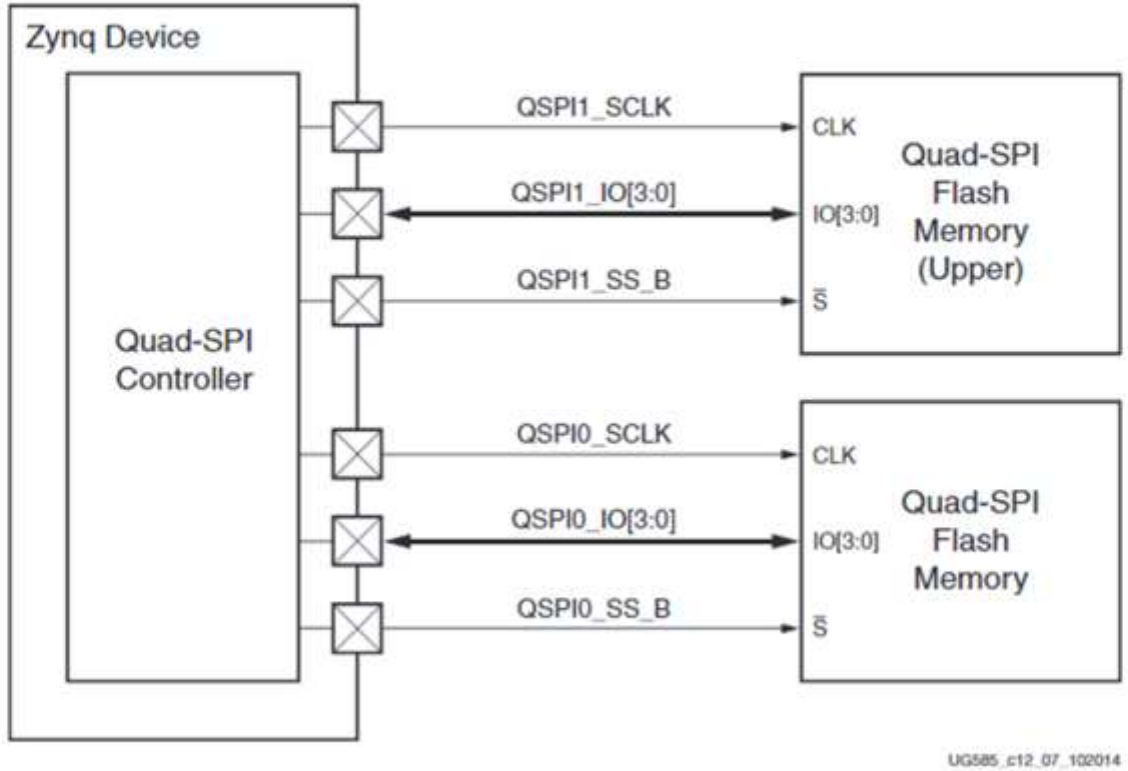


Figure 12-6: **Quad-SPI Dual SS, 8-bit Parallel I/O**

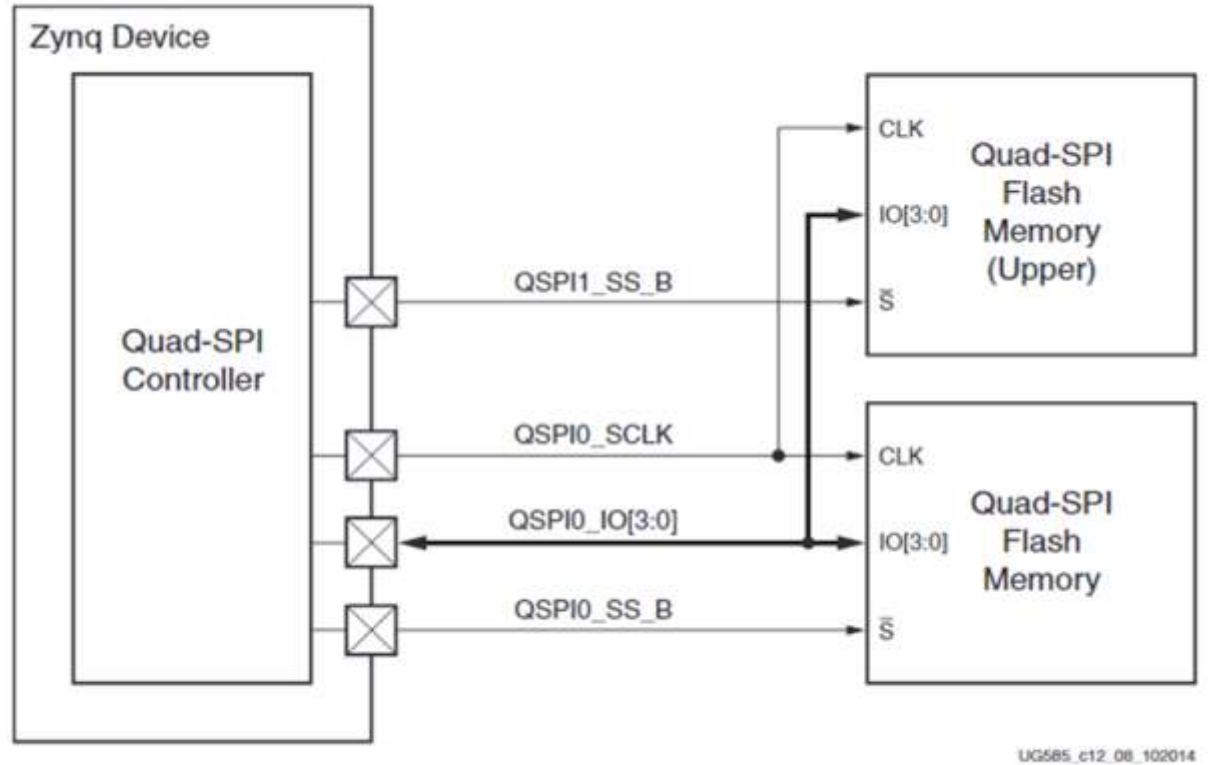
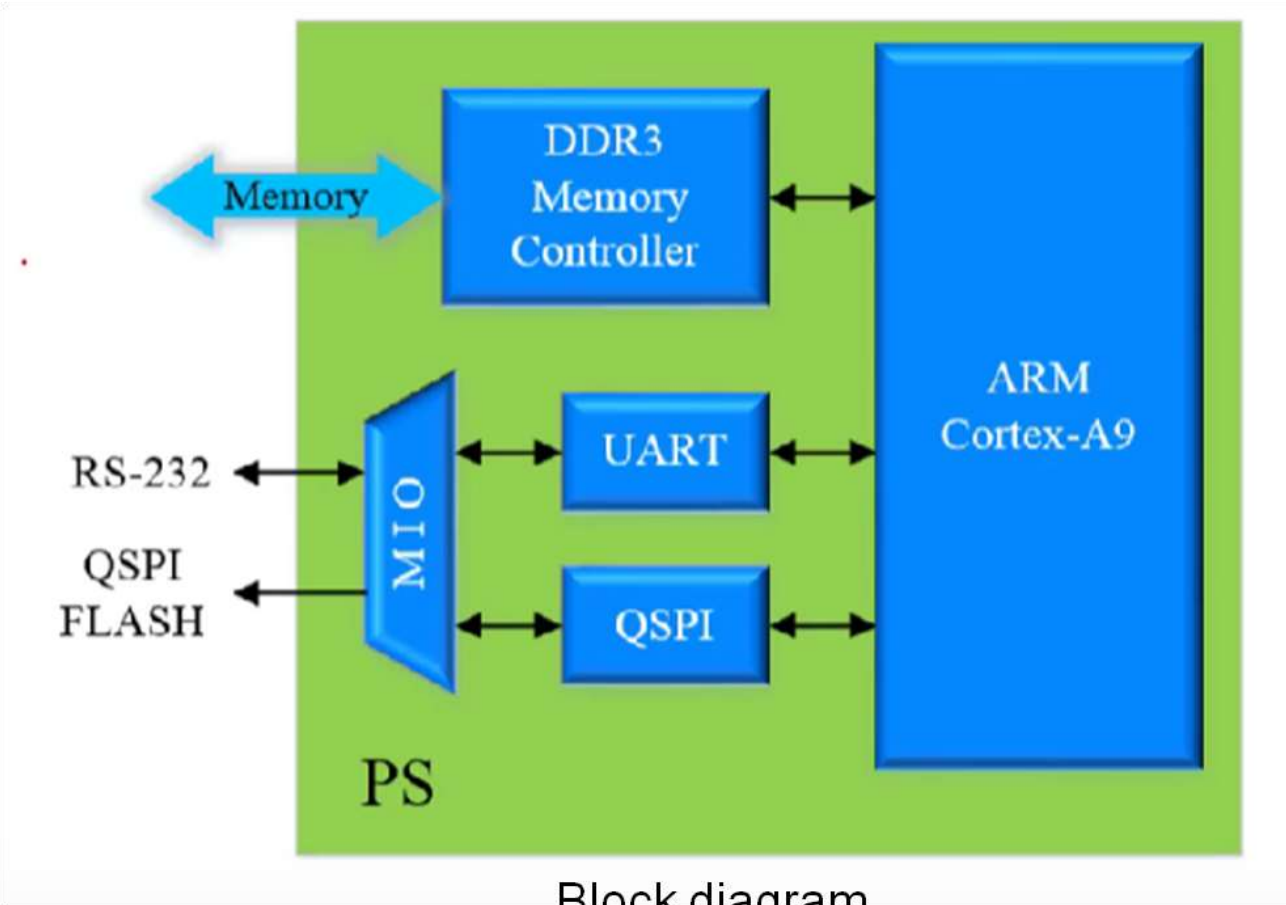


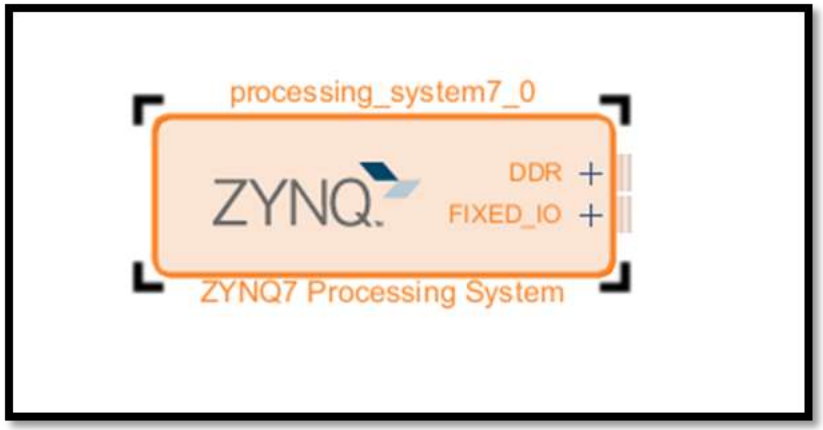
Figure 12-7: **Quad-SPI Dual SS 4-bit Stacked I/O**

Program Design

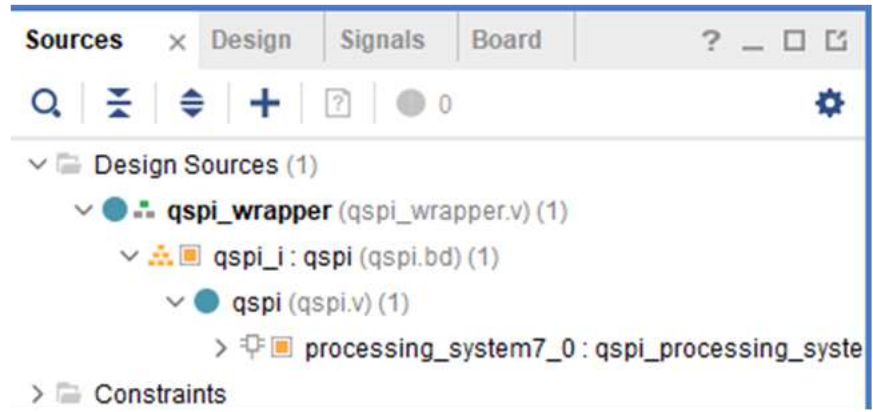
- Using QSPI Flash controller, Doing the Write/Read to the QSPI flash, than compare the read data with written data to verify whether the read and write operations are correct.

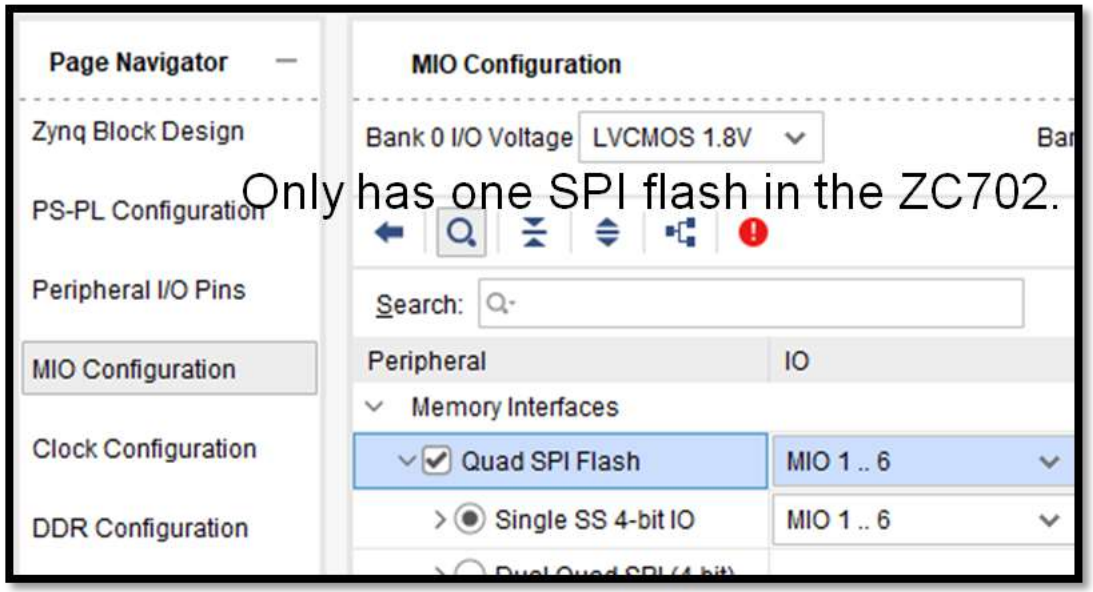


Block diagram

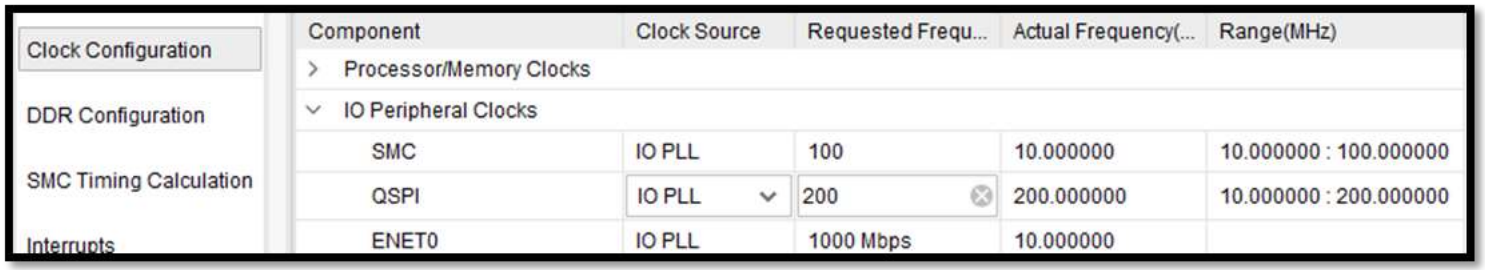


Generate output product
& Create HDL wrappers

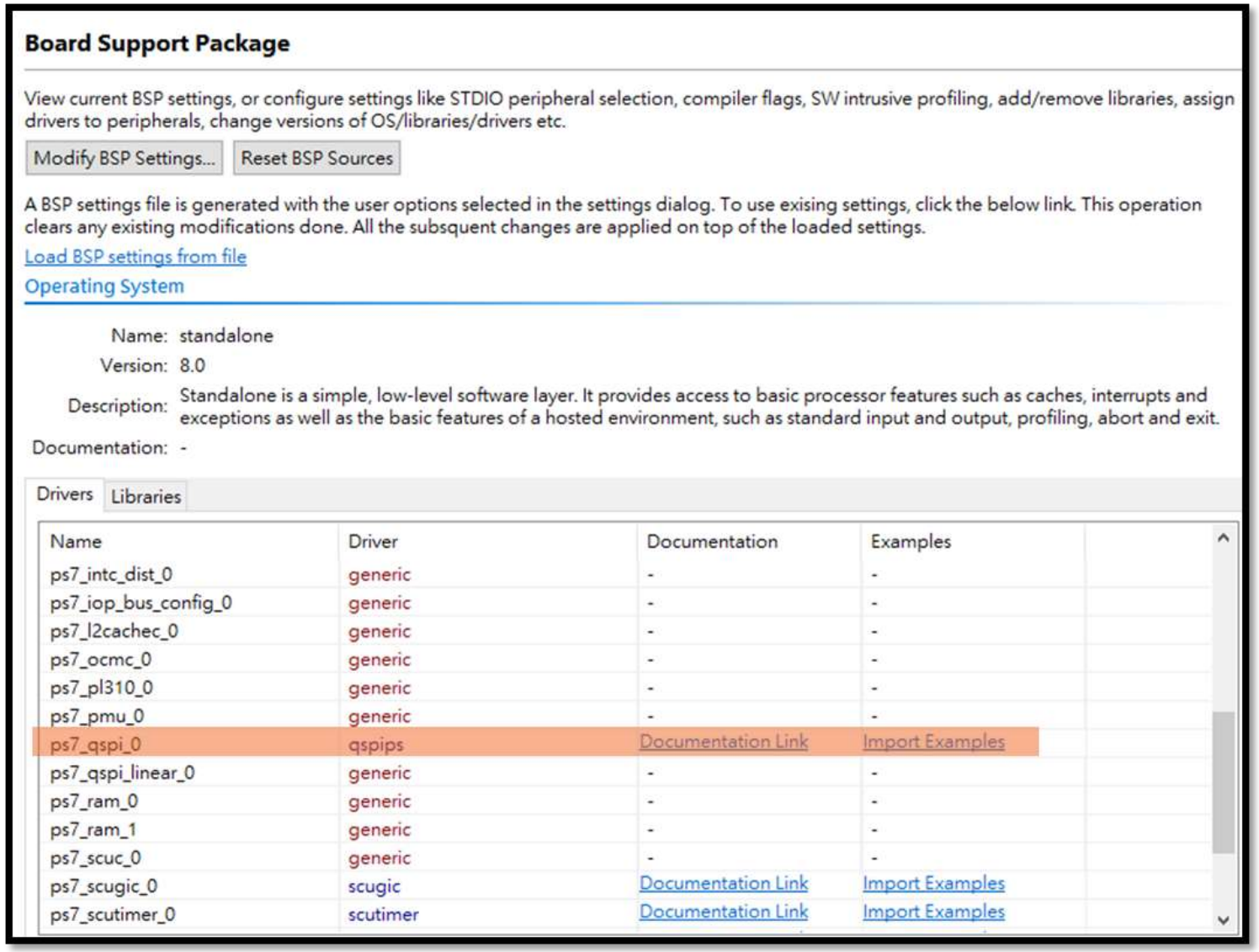


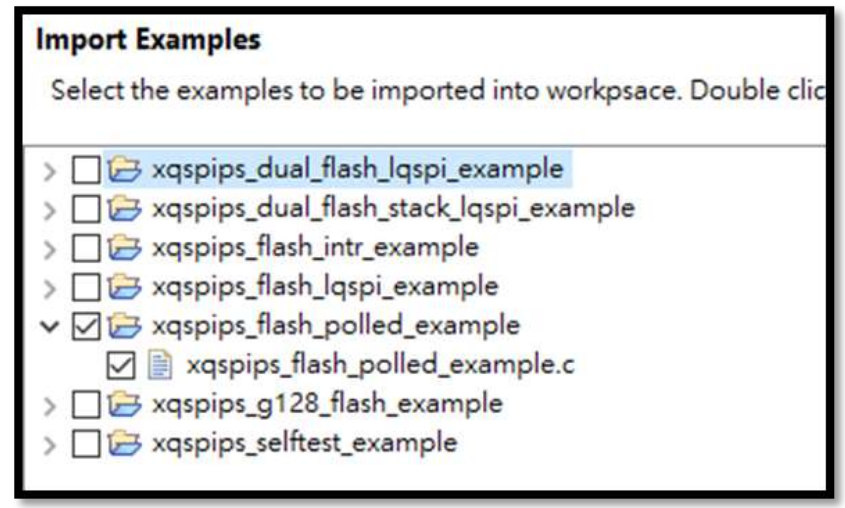


Only has one SPI flash in the ZC702.



Vitis


Board Support Package
View current BSP settings, or configure settings like STDIO peripheral selection, compiler flags, SW intrusive profiling, add/remove libraries, assign drivers to peripherals, change versions of OS/libraries/drivers etc.
Modify BSP Settings...
Reset BSP Sources
A BSP settings file is generated with the user options selected in the settings dialog. To use exising settings, click the below link. This operation clears any existing modifications done. All the subsquent changes are applied on top of the loaded settings.
Load BSP settings from file
Operating System
Name: standalone
Version: 8.0
Description: Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.
Documentation: -
Drivers
Libraries
Name
Driver
Documentation
Examples
ps7_intc_dist_0 generic - -
ps7_iop_bus_config_0 generic - -
ps7_l2cachec_0 generic - -
ps7_ocmc_0 generic - -
ps7_pl310_0 generic - -
ps7_pmu_0 generic - -
ps7_qspi_0 qspips Documentation Link Import Examples
ps7_qspi_linear_0 generic - -
ps7_ram_0 generic - -
ps7_ram_1 generic - -
ps7_scuc_0 generic - -
ps7_scugic_0 scugic Documentation Link Import Examples
ps7_scutimer_0 scutimer Documentation Link Import Examples


Import Examples
Select the examples to be imported into workpsace. Double clic
xqspips_dual_flash_lqspi_example
xqspips_dual_flash_stack_lqspi_example
xqspips_flash_intr_example
xqspips_flash_lqspi_example
xqspips_flash_polled_example
xqspips_flash_polled_example.c
xqspips_g128_flash_example
xqspips_selftest_example

```
/***************************** Include Files *********************************/

#include "xparameters.h"    /* SDK generated parameters */
#include "xqspips.h"        /* QSPI device driver */
#include "xil_printf.h"
```

```
/*
 * The following constants map to the XPAR parameters created in the
 * xparameters.h file. They are defined here such that a user can easily
 * change all the needed parameters in one place.
 */
#define QSPI_DEVICE_ID      XPAR_XQSPIPS_0_DEVICE_ID

/*
 * The following constants define the commands which may be sent to the FLASH
 * device.
 */
#define WRITE_STATUS_CMD    0x01
#define WRITE_CMD       0x02
#define READ_CMD        0x03
#define WRITE_DISABLE_CMD   0x04
#define READ_STATUS_CMD     0x05
#define WRITE_ENABLE_CMD    0x06
#define FAST_READ_CMD       0x0B
#define DUAL_READ_CMD       0x3B
#define QUAD_READ_CMD       0x6B
#define BULK_ERASE_CMD      0xC7
#define SEC_ERASE_CMD       0xD8
#define READ_ID         0x9F
```

CMD of FSM
The command refer UG256

All define need to reference the Flash spec

```
/*
 * The following constants define the offsets within a FlashBuffer data
 * type for each kind of data.  Note that the read data offset is not the
 * same as the write data because the QSPI driver is designed to allow full
 * duplex transfers such that the number of bytes received is the number
 * sent and received.
 */
#define COMMAND_OFFSET      0 /* FLASH instruction */
#define ADDRESS_1_OFFSET    1 /* MSB byte of address to read or write */
#define ADDRESS_2_OFFSET    2 /* Middle byte of address to read or write */
#define ADDRESS_3_OFFSET    3 /* LSB byte of address to read or write */
#define DATA_OFFSET     4 /* Start of Data for Read/Write */
#define DUMMY_OFFSET        4 /* Dummy byte offset for fast, dual and quad
                     * reads
                     */
#define DUMMY_SIZE      1 /* Number of dummy bytes for fast, dual and
                     * quad reads
                     */
#define RD_ID_SIZE      4 /* Read ID command + 3 bytes ID response */
#define BULK_ERASE_SIZE     1 /* Bulk Erase command size */
#define SEC_ERASE_SIZE      4 /* Sector Erase command + Sector address */
```

Dummy only in fast/Dual/Quad Read mode

```
/*
 * The following constants specify the extra bytes which are sent to the
 * FLASH on the QSPI interface, that are not data, but control information
 * which includes the command and address
 */
#define OVERHEAD_SIZE       4
```

CMD + address = 4

```
122 /*
123  * The following constants specify the page size, sector size, and number of
124  * pages and sectors for the FLASH.  The page size specifies a max number of
125  * bytes that can be written to the FLASH with a single transfer.
126  */
127 #define SECTOR_SIZE     0x10000
128 #define NUM_SECTORS     0x100
129 #define NUM_PAGES       0x10000
130 #define PAGE_SIZE       256
131
132 /* Number of flash pages to be written.*/
133 #define PAGE_COUNT      16
134
135 /* Flash address to which data is ot be written.*/
136 #define TEST_ADDRESS        0x00055000
137 #define UNIQUE_VALUE        0x05
138 /*
139  * The following constants specify the max amount of data and the size of the
140  * the buffer required to hold the data and overhead to transfer the data to
141  * and from the FLASH.
142  */
143 #define MAX_DATA        (PAGE_COUNT * PAGE_SIZE)
```

Like the initial value,

```
145 /************************** Type Definitions *****************************/
146
147 /***************** Macros (Inline Functions) Definitions *****************/
148
149 /************************** Function Prototypes ******************************/
150
151 void FlashErase(XQspiPs *QspiPtr, u32 Address, u32 ByteCount);
152
153 void FlashWrite(XQspiPs *QspiPtr, u32 Address, u32 ByteCount, u8 Command);
154
155 void FlashRead(XQspiPs *QspiPtr, u32 Address, u32 ByteCount, u8 Command);
156
157 int FlashReadID(void);
158
159 void FlashQuadEnable(XQspiPs *QspiPtr);
160
161 int QspiFlashPolledExample(XQspiPs *QspiInstancePtr, u16 QspiDeviceId);
162
```

```
/*
 * The following variable allows a test value to be added to the values that
 * are written to the FLASH such that unique values can be generated to
 * guarantee the writes to the FLASH were successful
 */
int Test = 5;

```

```
/*
 * The following variables are used to read and write to the flash and they
 * are global to avoid having large buffers on the stack
 */
u8 ReadBuffer[MAX_DATA + DATA_OFFSET + DUMMY_SIZE];
u8 WriteBuffer[PAGE_SIZE + DATA_OFFSET];

```

```
int main(void)
{
    int Status;

    xil_printf("QSPI FLASH Polled Example Test \r\n");

    /* Run the Qspi Interrupt example.*/
    Status = QspiFlashPolledExample(&QspiInstance, QSPI_DEVICE_ID);
    if (Status != XST_SUCCESS) {
        xil_printf("QSPI FLASH Polled Example Test Failed\r\n");
        return XST_FAILURE;
    }

    xil_printf("Successfully ran QSPI FLASH Polled Example Test\r\n");
    return XST_SUCCESS;
}

```

與累加的參數有關

```
229 ****************************************************************************/
230 int QspiFlashPolledExample(XQspiPs *QspiInstancePtr, u16 QspiDeviceId)
231 {
232     int Status;
233     u8 *BufferPtr;
234     u8 UniqueValue;
235     int Count;
236     int Page;
237     XQspiPs_Config *QspiConfig;
238
239     /* Initialize the QSPI driver so that it's ready to use*/
240     QspiConfig = XQspiPs_LookupConfig(QspiDeviceId);
241     if (QspiConfig == NULL) {
242         return XST_FAILURE;
243     }
244
245     Status = XQspiPs_CfgInitialize(QspiInstancePtr, QspiConfig,
246                     QspiConfig->BaseAddress);
247     if (Status != XST_SUCCESS) {
248         return XST_FAILURE;
249     }
```

Initial the qspi

```
251     /* Perform a self-test to check hardware build*/
252     Status = XQspiPs_SelfTest(QspiInstancePtr);
253     if (Status != XST_SUCCESS) {
254         return XST_FAILURE;
255     }
```

自行測試(可加可不加，本實驗後面就會對讀寫進行檢測)

```
257     /*
258      * Initialize the write buffer for a pattern to write to the FLASH
259      * and the read buffer to zero so it can be verified after the read,
260      * the test value that is added to the unique value allows the value
261      * to be changed in a debug environment to guarantee
262      */
263     for (UniqueValue = UNIQUE_VALUE, Count = 0; Count < PAGE_SIZE;
264          Count++, UniqueValue++) {
265         WriteBuffer[DATA_OFFSET + Count] = (u8)(UniqueValue + Test);
266     }
267     memset(ReadBuffer, 0x00, sizeof(ReadBuffer));
```

對WIRTE BUFFER 進行初始化以及賦值
剛進來時候，DATA OFFSET 的第四個數據 寫入5+5 = 10; count 加一次。
WIRTE BUFFER 內容進行重新賦值

READBUFFER 清零，清零 方面對我們WRITE BUFFER進行對比

```
269     /*
270      * Set Manual Start and Manual Chip select options and drive HOLD_B
271      * pin high.
272      */
273     XQspiPs_SetOptions(QspiInstancePtr, XQSPIPS_MANUAL_START_OPTION |
274             XQSPIPS_FORCE_SSELECT_OPTION |
275             XQSPIPS_HOLD_B_DRIVE_OPTION);
276
277     /* Set the prescaler for QSPI clock*/
278     XQspiPs_SetClkPrescaler(QspiInstancePtr, XQSPIPS_CLK_PRESCALE_8);
279
280     /* Assert the FLASH chip select.*/
281     XQspiPs_SetSlaveSelect(QspiInstancePtr);
282
283
284     FlashReadID();
285
286     FlashQuadEnable(QspiInstancePtr);
287
288     /* Erase the flash.*/
289     FlashErase(QspiInstancePtr, TEST_ADDRESS, MAX_DATA);
290
291     /*
292      * Write the data in the write buffer to the serial FLASH a page at a
293      * time, starting from TEST_ADDRESS
294      */
295     for (Page = 0; Page < PAGE_COUNT; Page++) {
296         FlashWrite(QspiInstancePtr, (Page * PAGE_SIZE) + TEST_ADDRESS,
297                 PAGE_SIZE, WRITE_CMD);
298     }
299
```

把flash暫存器 設置成手動模式 HOLD_B 需要拉高

FLASH CLOCK 進行分頻，一開始VIVADO 設置200M，這裡將他除8

片選信號有效

FLASH寫入數據只能從 1 寫入0 , 不能把0寫入1, 如果要0寫入1 需要進行擦除，這裡的擦除只針對FLASH測試的範圍

朝每個PAGE 寫入數據

```
300     /*
301      * Read the contents of the FLASH from TEST_ADDRESS, using Normal Read
302      * command. Change the prescaler as the READ command operates at a
303      * lower frequency.
304      */
305     FlashRead(QspiInstancePtr, TEST_ADDRESS, MAX_DATA, READ_CMD);
306
307     /*
308      * Setup a pointer to the start of the data that was read into the read
309      * buffer and verify the data read is the data that was written
310      */
311     BufferPtr = &ReadBuffer[DATA_OFFSET];
312
313     for (UniqueValue = UNIQUE_VALUE, Count = 0; Count < MAX_DATA;
314          Count++, UniqueValue++) {
315         if (BufferPtr[Count] != (u8)(UniqueValue + Test)) {
316             return XST_FAILURE;
317         }
318     }
319
320     /*
321      * Read the contents of the FLASH from TEST_ADDRESS, using Fast Read
322      * command
323      */
324     memset(ReadBuffer, 0x00, sizeof(ReadBuffer));
325     FlashRead(QspiInstancePtr, TEST_ADDRESS, MAX_DATA, FAST_READ_CMD);
326
327     /*
328      * Setup a pointer to the start of the data that was read into the read
329      * buffer and verify the data read is the data that was written
330      */
331     BufferPtr = &ReadBuffer[DATA_OFFSET + DUMMY_SIZE];
332
333     for (UniqueValue = UNIQUE_VALUE, Count = 0; Count < MAX_DATA;
334          Count++, UniqueValue++) {
335         if (BufferPtr[Count] != (u8)(UniqueValue + Test)) {
336             return XST_FAILURE;
337         }
338     }
```

讀出的數據 放入READBUFFER 在FLASHREAD函數裡，指標指向一個BUFFER 指向一個數據

指標指向一個BUFFER 指向一個數據

對READ的數據 對 下次的數據做比較，對回傳錯誤

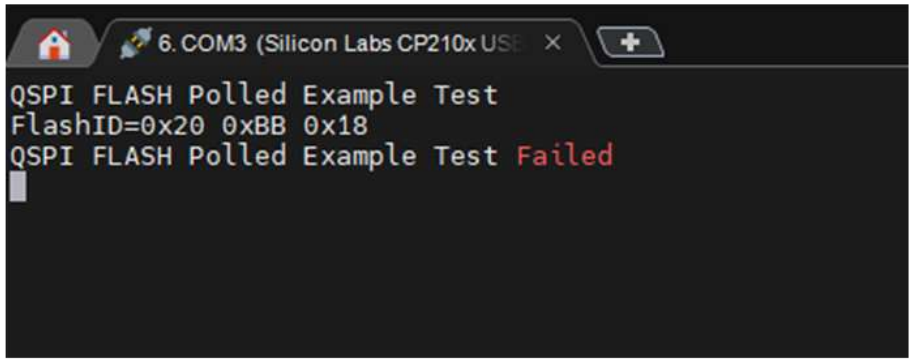
```
300     /*
301      * Read the contents of the FLASH from TEST_ADDRESS, using Normal Read
302      * command. Change the prescaler as the READ command operates at a
303      * lower frequency.
304      */
305     FlashRead(QspiInstancePtr, TEST_ADDRESS, MAX_DATA, READ_CMD);
306
```

修改讀取速度的命令

```
300     /*
301      * Read the contents of the FLASH from TEST_ADDRESS, using Normal Read
302      * command. Change the prescaler as the READ command operates at a
303      * lower frequency.
304      */
305 //  FlashRead(QspiInstancePtr, TEST_ADDRESS, MAX_DATA, READ_CMD);
306     FlashRead(QspiInstancePtr, TEST_ADDRESS, MAX_DATA, QUAD_READ_CMD);
307
```

後面內容都不需要了，只是模式上的差別320行 以後都不需要了

```
320
321     /*
322      * Read the contents of the FLASH from TEST_ADDRESS, using Fast Read
323      * command
324      */
325     memset(ReadBuffer, 0x00, sizeof(ReadBuffer));
326     FlashRead(QspiInstancePtr, TEST_ADDRESS, MAX_DATA, FAST_READ_CMD);
327
328     /*
329      * Setup a pointer to the start of the data that was read into the read
```



debug

```
411     /*
412      * Read the contents of the FLASH from TEST_ADDRESS, using Normal Read
413      * command. Change the prescaler as the READ command operates at a
414      * lower frequency.
415      */
416     FlashRead(QspiInstancePtr, TEST_ADDRESS, MAX_DATA, READ_CMD);
417
418     /*
419      * Setup a pointer to the start of the data that was read into the read
420      * buffer and verify the data read is the data that was written
421      */
422     BufferPtr = &ReadBuffer[DATA_OFFSET];
423
424     for (UniqueValue = UNIQUE_VALUE, Count = 0; Count < MAX_DATA;
425          Count++, UniqueValue++) {
426         if (BufferPtr[Count] != (u8)(UniqueValue + Test)) {
427             return XST_FAILURE;
428         }
429     }
430
```

```
471     /*
472      * Read the contents of the FLASH from TEST_ADDRESS, using Quad Read
473      * command
474      */
475     memset(ReadBuffer, 0x00, sizeof(ReadBuffer));
476     FlashRead(QspiInstancePtr, TEST_ADDRESS, MAX_DATA, QUAD_READ_CMD);
477
478     /*
479      * Setup a pointer to the start of the data that was read into the read
480      * buffer and verify the data read is the data that was written
481      */
482     BufferPtr = &ReadBuffer[DATA_OFFSET + DUMMY_SIZE];
483
484     for (UniqueValue = UNIQUE_VALUE, Count = 0; Count < MAX_DATA;
485          Count++, UniqueValue++) {
486         if (BufferPtr[Count] != (u8)(UniqueValue + Test)) {
487             return XST_FAILURE;
488         }
489     }
```

P因為我們用QUAD_READ_CMD的模式，所以他的BUFFER的偏移不同，需要在後面在加個DUMMY_SIZE

```
300     /*
301      * Read the contents of the FLASH from TEST_ADDRESS, using Normal Read
302      * command. Change the prescaler as the READ command operates at a
303      * lower frequency.
304      */
305 //  FlashRead(QspiInstancePtr, TEST_ADDRESS, MAX_DATA, READ_CMD);
306     FlashRead(QspiInstancePtr, TEST_ADDRESS, MAX_DATA, QUAD_READ_CMD);
307
308     /*
309      * Setup a pointer to the start of the data that was read into the read
310      * buffer and verify the data read is the data that was written
311      */
312 //  BufferPtr = &ReadBuffer[DATA_OFFSET];
313     BufferPtr = &ReadBuffer[DATA_OFFSET + DUMMY_SIZE];
314     for (UniqueValue = UNIQUE_VALUE, Count = 0; Count < MAX_DATA;
315          Count++, UniqueValue++) {
316         if (BufferPtr[Count] != (u8)(UniqueValue + Test)) {
317             return XST_FAILURE;
318         }
319     }
```

```
QSPI FLASH Polled Example Test
FlashID=0x20 0xBB 0x18
Successfully ran QSPI FLASH Polled Example Test
```


AMD