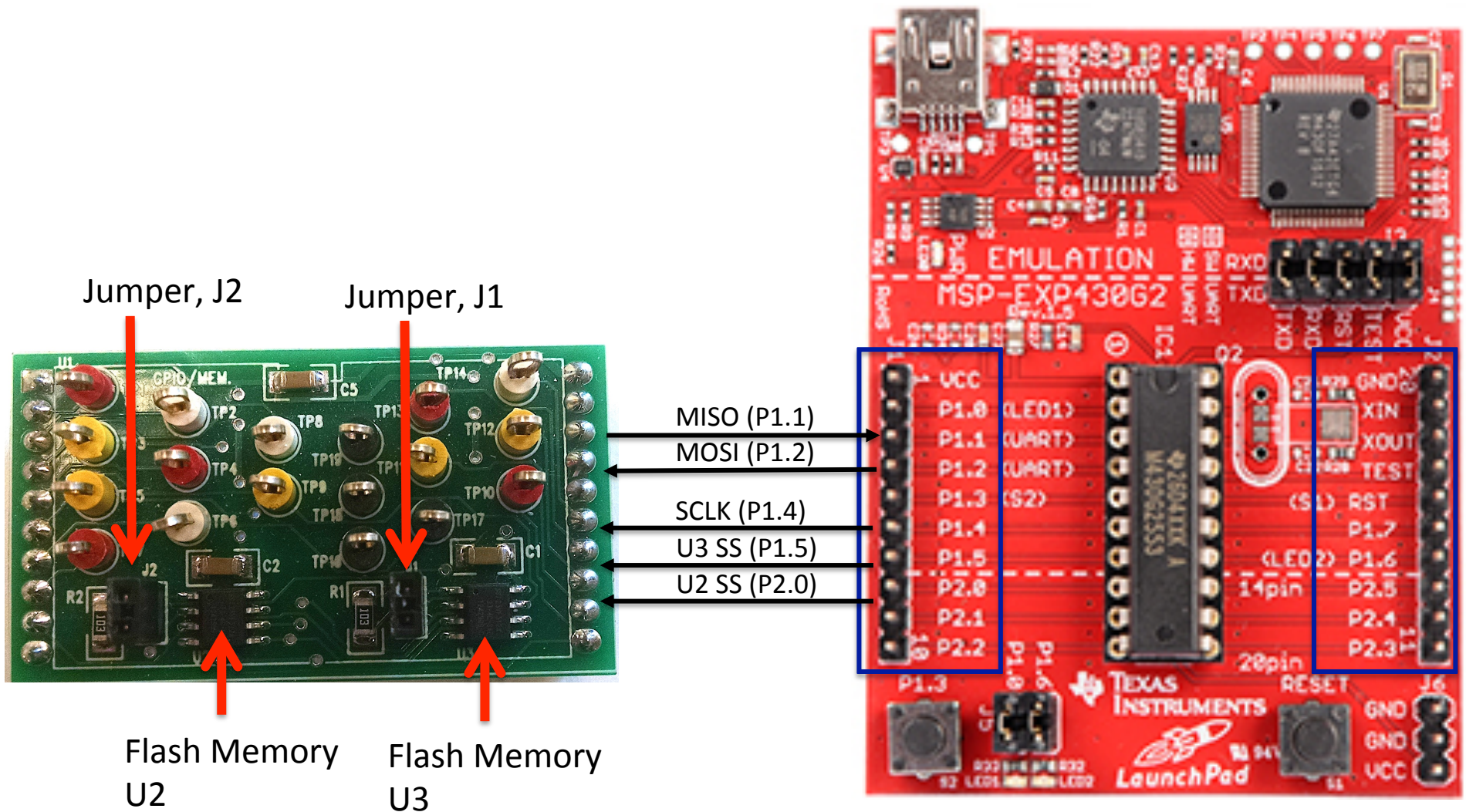


# SPI Interfacing to Serial Flash Memory

# GPIO/Serial Flash Header Board



# Sending Information Using SPI

```
/*  
 * This function initializes all hardware and port pins to support SPI.  
 */  
void InitializeSPI();
```

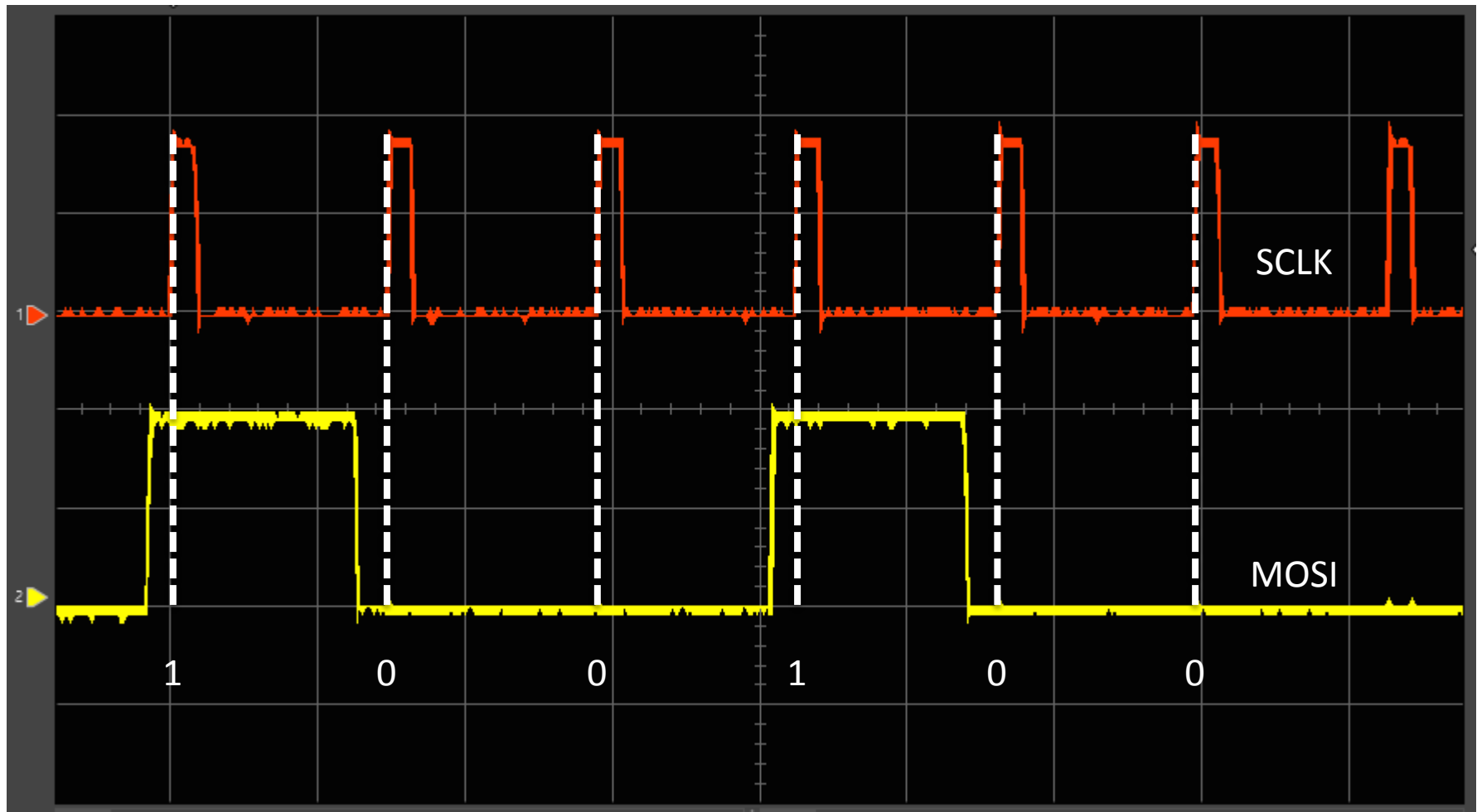
```
/*  
 * This function sends the byte, SendValue, using SPI.  
 */  
void SPISendByte(unsigned char SendValue);
```

```
/*  
 * This function receives a byte using SPI.  
 *  
 * Return Value: The byte that is received over SPI.  
 */  
unsigned char SPIReceiveByte();
```

# Sending Information Using SPI

```
Create local copy of SendValue;  
for (each bit in SendValue) {  
    if (MSB equals 1)  
        set MOSI;  
    else  
        reset MOSI;  
    Left-Shift local copy of SendValue by 1;  
  
    // SCLK: 0->1->0  
    Toggle SCLK twice;  
}
```

# Milestone 1 Verification



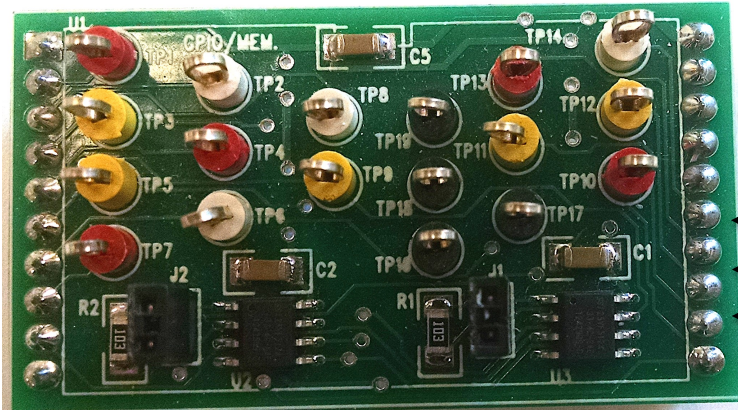
# Receiving Information Using SPI

```
/*  
 * This function initializes all hardware and port pins to support SPI.  
 */  
void InitializeSPI();  
  
/*  
 * This function sends the byte, SendValue, using SPI.  
 */  
void SPISendByte(unsigned char SendValue);  
  
/*  
 * This function receives a byte using SPI.  
 *  
 * Return Value: The byte that is received over SPI.  
 */  
unsigned char SPIReceiveByte();
```

# Receiving Information Using SPI

```
Initialize ReceiveValue = 0;  
for (each bit in received byte) {  
  
    Left-shift current value of ReceiveValue by 1;  
    Then, OR ReceiveValue with MISO;  
  
    // SCLK: 0->1->0  
    Toggle SCLK twice;  
}
```

# Reading MISO



MISO (P1.1)

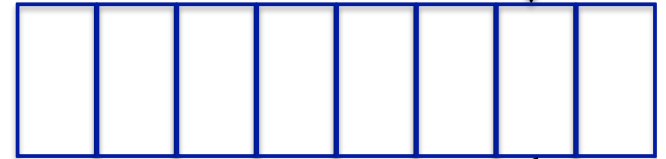
Bit value from MISO is stored in bit 1 of P1IN

SCLK (P1.4)

U3 SS (P1.5)

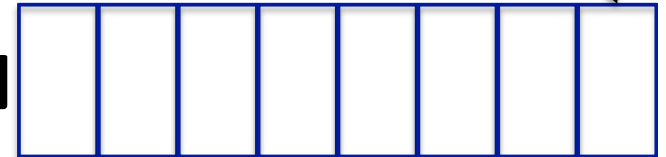
U2 SS (P2.0)

**P1IN**



Shift bit to known bit location, like LSB

**Value read from P1IN**





# Verification

```
// Begin by reading the ID for each flash memory. The value should be 0xBF48 for both.
ID_U3 = ReadFlashMemoryID(FLASH_MEMORY_U3); ←
if (ID_U3 != 0xBF48) {
    TestHasNotFailed[TestNumber] = FALSE;
    TestNumber++;
}
ID_U2 = ReadFlashMemoryID(FLASH_MEMORY_U2); ←
if (ID_U2 != 0xBF48) {
    TestHasNotFailed[TestNumber] = FALSE;
    TestNumber++;
}

// Next, turn on block protection, and then read status register for each flash memory.
// The value for each should be 0x0C.
SetBlockProtection(FULL, FLASH_MEMORY_U3);
SetBlockProtection(FULL, FLASH_MEMORY_U2);
StatusRegister_U3 = ReadFlashMemoryStatusRegister(FLASH_MEMORY_U3); ←
if (StatusRegister_U3 != 0x0C) {
    TestHasNotFailed[TestNumber] = FALSE;
    TestNumber++;
}
StatusRegister_U2 = ReadFlashMemoryStatusRegister(FLASH_MEMORY_U2); ←
if (StatusRegister_U2 != 0x0C) {
    TestHasNotFailed[TestNumber] = FALSE;
    TestNumber++;
}
```




# Final Verification

- The checksum test writes a pseudo-random test pattern to each memory location, and then reads the contents to confirm that every memory location can be correctly accessed

```
// Next, write pseudo-random data to each flash, and then read it to confirm flash read/write.  
ChecksumFlashMemoryTest(FLASH_MEMORY_U3,BYTE_PROGRAM,CHIP_ERASE,INITIAL_LFSR_STATE,  
    &LFSRStateAfterRead_U3,&LFSRStateAfterWrite_U3);  
if (LFSRStateAfterRead_U3 != LFSRStateAfterWrite_U3) {  
    TestHasNotFailed[TestNumber] = FALSE;  
}  
TestNumber++;  
ChecksumFlashMemoryTest(FLASH_MEMORY_U2,AAI_PROGRAM,~CHIP_ERASE,INITIAL_LFSR_STATE,  
    &LFSRStateAfterRead_U2,&LFSRStateAfterWrite_U2);  
if (LFSRStateAfterRead_U2 != LFSRStateAfterWrite_U2) {  
    TestHasNotFailed[TestNumber] = FALSE;  
}  
TestNumber++;
```


# Final Verification

- The confirmation is accomplished by comparing the two variables passed to the function (using pointers)

```
// Next, write pseudo-random data to each flash, and then read it to confirm flash read/write.
CheckSumFlashMemoryTest(FLASH_MEMORY_U3,BYTE_PROGRAM,CHIP_ERASE,INITIAL_LFSR_STATE,
    &LFSRStateAfterRead_U3,&LFSRStateAfterWrite_U3);
if (LFSRStateAfterRead_U3 != LFSRStateAfterWrite_U3) { 
    TestHasNotFailed[TestNumber] = FALSE;
}
TestNumber++;
CheckSumFlashMemoryTest(FLASH_MEMORY_U2,AAI_PROGRAM,~CHIP_ERASE,INITIAL_LFSR_STATE,
    &LFSRStateAfterRead_U2,&LFSRStateAfterWrite_U2);
if (LFSRStateAfterRead_U2 != LFSRStateAfterWrite_U2) {
    TestHasNotFailed[TestNumber] = FALSE;
}
TestNumber++;
```


# Final Verification

- One variable represents the final state of the LFSR used to generate the pseudo-random patterns after writing to each memory location

```
// Next, write pseudo-random data to each flash, and then read it to confirm flash read/write.
CheckSumFlashMemoryTest(FLASH_MEMORY_U3,BYTE_PROGRAM,CHIP_ERASE,INITIAL_LFSR_STATE,
    &LFSRStateAfterRead_U3,&LFSRStateAfterWrite_U3);
if (LFSRStateAfterRead_U3 != LFSRStateAfterWrite_U3) { 
    TestHasNotFailed[TestNumber] = FALSE;
}
TestNumber++;
CheckSumFlashMemoryTest(FLASH_MEMORY_U2,AAI_PROGRAM,~CHIP_ERASE,INITIAL_LFSR_STATE,
    &LFSRStateAfterRead_U2,&LFSRStateAfterWrite_U2);
if (LFSRStateAfterRead_U2 != LFSRStateAfterWrite_U2) {
    TestHasNotFailed[TestNumber] = FALSE;
}
TestNumber++;
```


# Final Verification

- The other variable represents the final state of the LFSR after the contents of each memory location is read

```
// Next, write pseudo-random data to each flash, and then read it to confirm flash read/write.
CheckSumFlashMemoryTest(FLASH_MEMORY_U3,BYTE_PROGRAM,CHIP_ERASE,INITIAL_LFSR_STATE,
    &LFSRStateAfterRead_U3,&LFSRStateAfterWrite_U3);
if (LFSRStateAfterRead_U3 != LFSRStateAfterWrite_U3) { 
    TestHasNotFailed[TestNumber] = FALSE;
}
TestNumber++;
CheckSumFlashMemoryTest(FLASH_MEMORY_U2,AAI_PROGRAM,~CHIP_ERASE,INITIAL_LFSR_STATE,
    &LFSRStateAfterRead_U2,&LFSRStateAfterWrite_U2);
if (LFSRStateAfterRead_U2 != LFSRStateAfterWrite_U2) {
    TestHasNotFailed[TestNumber] = FALSE;
}
TestNumber++;
```


# Final Verification

- If the read/write operations were successful, the values should be equal

```
// Next, write pseudo-random data to each flash, and then read it to confirm flash read/write.
CheckSumFlashMemoryTest(FLASH_MEMORY_U3,BYTE_PROGRAM,CHIP_ERASE,INITIAL_LFSR_STATE,
    &LFSRStateAfterRead_U3,&LFSRStateAfterWrite_U3);
if (LFSRStateAfterRead_U3 != LFSRStateAfterWrite_U3) { 
    TestHasNotFailed[TestNumber] = FALSE;
}
TestNumber++;
CheckSumFlashMemoryTest(FLASH_MEMORY_U2,AAI_PROGRAM,~CHIP_ERASE,INITIAL_LFSR_STATE,
    &LFSRStateAfterRead_U2,&LFSRStateAfterWrite_U2);
if (LFSRStateAfterRead_U2 != LFSRStateAfterWrite_U2) {
    TestHasNotFailed[TestNumber] = FALSE;
}
TestNumber++;
```




Name	Type	Value	Location
(x)= ID_U2	unsigned int	0xBF48 (Hex)	0x03F2
(x)= ID_U3	unsigned int	0xBF48 (Hex)	0x03F0
(x)= LFSRStateAfterRead_U2	unsigned int	0x0362 (Hex) ←	0x03EC
(x)= LFSRStateAfterRead_U3	unsigned int	0x0362 (Hex) ←	0x03E8
(x)= LFSRStateAfterWrite_U2	unsigned int	0x0362 (Hex) ←	0x03EE
(x)= LFSRStateAfterWrite_U3	unsigned int	0x0362 (Hex) ←	0x03EA
(x)= NumberOfTestsPassed	unsigned char	.	Register R13
(x)= StatusRegister_U2	unsigned char	0x0C (Hex)	0x03F5
(x)= StatusRegister_U3	unsigned char	0x0C (Hex)	0x03F4
▶  TestHasNotFailed	unsigned char[6]	0x03F6	0x03F6
(x)= TestNumber	unsigned char	.	Register R10

Name	Type	Value	Location
(x)= ID_U2	unsigned int	0xBF48 (Hex)	0x03F2
(x)= ID_U3	unsigned int	0xBF48 (Hex)	0x03F0
(x)= LFSRStateAfterRead_U2	unsigned int	0x0362 (Hex)	0x03EC
(x)= LFSRStateAfterRead_U3	unsigned int	0x0362 (Hex)	0x03E8
(x)= LFSRStateAfterWrite_U2	unsigned int	0x0362 (Hex)	0x03EE
(x)= LFSRStateAfterWrite_U3	unsigned int	0x0362 (Hex)	0x03EA
(x)= NumberOfTestsPassed	unsigned char	.	Register R13
(x)= StatusRegister_U2	unsigned char	0x0C (Hex)	0x03F5
(x)= StatusRegister_U3	unsigned char	0x0C (Hex)	0x03F4
▶  TestHasNotFailed	unsigned char[6]	0x03F6	0x03F6
(x)= TestNumber	unsigned char	.	Register R10

Memory Browser ✕

0x300

0x3b0 - 0x300(+0xb0) <Memory Rendering 2> 

### 16-Bit Hex - TI Style

0x03B0 stack

```
0x03B0 9802 8C04 0858 2908 0A04 C80C 0002 C950 C62A FFFE C500 0000 FFFE 03EC 0003 C322
```

```
0x03D0  0303 06C4 0002 0001 03EA BAFE 0060 03E8 0005 C140 03EC 03EE 0362 0362 0362 0362
```

```
0x03F0  BF48 BF48 0C0C 0101 0101 0101 0002 C92A 0000 0000 0000 0000 0000 0000 0000 0000
```

[illegible]



# Final Verification

```
// Next, write pseudo-random data to each flash, and then read it to confirm flash read/write.
ChecksumFlashMemoryTest(FLASH_MEMORY_U3,BYTE_PROGRAM,CHIP_ERASE,INITIAL_LFSR_STATE,
    &LFSRStateAfterRead_U3,&LFSRStateAfterWrite_U3);
if (LFSRStateAfterRead_U3 != LFSRStateAfterWrite_U3) {
    TestHasNotFailed[TestNumber] = FALSE;
}
TestNumber++;
ChecksumFlashMemoryTest(FLASH_MEMORY_U2,AAI_PROGRAM,~CHIP_ERASE,INITIAL_LFSR_STATE,
    &LFSRStateAfterRead_U2,&LFSRStateAfterWrite_U2);
if (LFSRStateAfterRead_U2 != LFSRStateAfterWrite_U2) {
    TestHasNotFailed[TestNumber] = FALSE;
}
TestNumber++;

// If all tests were successful, then indicate this by toggling the green LED.
// Otherwise, toggle the red LED.
for (NumberOfTestsPassed = 0, TestNumber = 0; TestNumber < NUMBER_OF_TESTS; TestNumber++)
    NumberOfTestsPassed += TestHasNotFailed[TestNumber];
while (TRUE) {
    if (NumberOfTestsPassed == NUMBER_OF_TESTS) {
        TOGGLE_LED2;
    }
    else {
        TOGGLE_LED1;
    }
    _delay_cycles(1000000);
}
```

When testing U3, byte programming and full chip-erase are tested.

# Final Verification

```
// Next, write pseudo-random data to each flash, and then read it to confirm flash read/write.
CheckSumFlashMemoryTest(FLASH_MEMORY_U3,BYTE_PROGRAM,CHIP_ERASE,INITIAL_LFSR_STATE,
    &LFSRStateAfterRead_U3,&LFSRStateAfterWrite_U3);
if (LFSRStateAfterRead_U3 != LFSRStateAfterWrite_U3) {
    TestHasNotFailed[TestNumber] = FALSE;
}
TestNumber++;
CheckSumFlashMemoryTest(FLASH_MEMORY_U2,AAI_PROGRAM,~CHIP_ERASE,INITIAL_LFSR_STATE,
    &LFSRStateAfterRead_U2,&LFSRStateAfterWrite_U2);
if (LFSRStateAfterRead_U2 != LFSRStateAfterWrite_U2) {
    TestHasNotFailed[TestNumber] = FALSE;
}
TestNumber++;

// If all tests were successful, then indicate this by toggling the green LED.
// Otherwise, toggle the red LED.
for (NumberOfTestsPassed = 0, TestNumber = 0; TestNumber < NUMBER_OF_TESTS; TestNumber++)
    NumberOfTestsPassed += TestHasNotFailed[TestNumber];
while (TRUE) {
    if (NumberOfTestsPassed == NUMBER_OF_TESTS) {
        TOGGLE_LED2;
    }
    else {
        TOGGLE_LED1;
    }
    _delay_cycles(1000000);
}
```

When testing U2, AAI programming and sector/block-erase are tested.

# Final Verification

```
// Next, write pseudo-random data to each flash, and then read it to confirm flash read/write.
CheckSumFlashMemoryTest(FLASH_MEMORY_U3,BYTE_PROGRAM,CHIP_ERASE,INITIAL_LFSR_STATE,
    &LFSRStateAfterRead_U3,&LFSRStateAfterWrite_U3);
if (LFSRStateAfterRead_U3 != LFSRStateAfterWrite_U3) {
    TestHasNotFailed[TestNumber] = FALSE;
}
TestNumber++;
CheckSumFlashMemoryTest(FLASH_MEMORY_U2,AAI_PROGRAM,~CHIP_ERASE,INITIAL_LFSR_STATE,
    &LFSRStateAfterRead_U2,&LFSRStateAfterWrite_U2);
if (LFSRStateAfterRead_U2 != LFSRStateAfterWrite_U2) {
    TestHasNotFailed[TestNumber] = FALSE;
}
TestNumber++;

// If all tests were successful, then indicate this by toggling the green LED.
// Otherwise, toggle the red LED.
for (NumberOfTestsPassed = 0, TestNumber = 0; TestNumber < NUMBER_OF_TESTS; TestNumber++)
    NumberOfTestsPassed += TestHasNotFailed[TestNumber];
while (TRUE) {
    if (NumberOfTestsPassed == NUMBER_OF_TESTS) {
        TOGGLE_LED2;
    }
    else {
        TOGGLE_LED1;
    }
    _delay_cycles(1000000);
}
```

Finally, if all of the tests are passed, the the **green** LED will flash to indicate as such.

Otherwise, the **red** LED will blink