VS10XX APPNOTE: CONNECTING SPI BUSES

VS1011, VS1003, VS1033, VS1053, VS1063, VS1073

Revision History			
Rev.	**Date**	**Author**	**Description**
1.10	2025-04-01	HH	Added VS1073.
1.01	2012-12-11	HH	Corrected DREQ-less documentation.
1.00	2012-11-21	HH	Initial version.

Contents

List of Figures

1 Introduction to VS10xx SPI Buses

VLSI Solution's slave VS10xx ICs are typically connected to a microcontroller using two SPI buses. These buses, which share clock and data pins, are called SCI (Serial Control Interface) and SDI (Serial Data Interface). This document presents seven different ways to connect VS10xx to a host microcontroller with SPI, with example code and SPI waveform examples for each way.

This document is applicable to VS1011 (see note below), VS1003, VS1033, VS1053, VS8053, VS1063, and VS1073.

Before reading this document, read the datasheet of the IC you are using, and particularly chapters *SPI Buses*, *Serial Data Interface (SDI)*, *Serial Control Interface (SCI)*, and *SCI Registers*.

In the examples of this document SCI_MODE register bits SM_DACT (clock polarity) and SM_SDIORD (bit order in SDI bus) are both at their default value 0, and SM_SDINEW is set to 1 in all examples.

Note: In VS1011 the default value for SCI_MODE register bit SM_SDINEW is 0. Set it to 1! (Example codes do this for you.)

1.1 Reading DREQ Without a Dedicated Pin

Some of the examples in this document (Chapters 2.3, 2.6, and 2.7) don't connect the DREQ pin. Instead they read the DREQ status through the SCI bus. This section tells how to do that.

1.1.1 Reading DREQ without DREQ Pin with VS1073 and VS1063

With VS1073 and VS1063, the following algorithm can be used to read DREQ status:

```
/* Method specifically for VS1073 and VS1063 */
int ReadDreq(void) {
  WriteSci(SCI_WRAMADDR, 0xc0df);
  return (ReadSci(SCI_WRAM) < 40) ? 0 : 1;
}
```

1.1.2 Reading DREQ without DREQ Pin with Older VS10XX ICs

If using an older IC than VS1063a, you must define an IC-dependent value for DREQ_ADDR from the table below. With VS1063a, you can use either this method, or the method as shown in Chapter 1.1.1 (recommended).

Values for DREQ_ADDR for different VS10XX ICs	
IC	**Definition**
VS1011e	#define DREQ_ADDR 0x40c4
VS1003b	#define DREQ_ADDR 0x595e
VS1033d	#define DREQ_ADDR 0x5a53
VS1053b	#define DREQ_ADDR 0x5b17
VS8053b	#define DREQ_ADDR 0x5b17
VS1063a	#define DREQ_ADDR 0x5b23

Then, you can read DREQ with the following algorithm:

```
  /* Method specifically for others than VS1073.
     You have to have correct definition for DREQ_ADDR! */
int ReadDreq(void) {
  WriteSci(SCI_WRAMADDR, DREQ_ADDR); // DREQ_ADDR depends on the IC
  return ReadSci(SCI_WRAM);
}
```

2 Seven Different Ways of Connecting SPI Buses SCI and SDI

This Chapter presents seven different ways for connecting the two VS10xx SPI buses, SCI and SDI. Each of the ways have their own pros and cons, which are listed, along with example pseudocode of how to implement data transfers that are compatible with the hardware. Also waveforms are presented for each case.

All of the following functions assume there the following datatypes and SPI support functions exist on the microcontroller, and that the following initializations have been done:

```
// Microcontroller definitions and initialization

typedef unsigned char u_int8;
typedef unsigned short u_int16;

void WriteSpiByte(u_int8 data);           // You need to provide this
u_int8 ReadSpiByte(void);                 // You need to provide this
void SetGpioLow(int registerName);        // You need to provide this
void SetGpioHigh(int registerName);       // You need to provide this
void WaitMicroseconds(int microseconds);  // You need to provide this
void WriteSci(u_int8 addr, u_int16 data);                  // Examples in this document
void WriteSciMulti(u_int8 addr, const u_int16 *data, int nWords);// Ex in this document
u_int16 ReadSci(u_int8 addr);                              // Examples in this document
void ReadSciMulti(u_int8 addr, u_int16 *data, int nWords); // Examples in this document
int WriteSdi(const u_int8 *data, u_int16 bytes);           // Examples in this document

/* Some SCI registers */
#define SCI_MODE      0
#define SCI_WRAM      6
#define SCI_WRAMADDR  7

/* Some SCI_MODE bits */
#define SM_SDISHARE_B 10
#define SM_SDINEW_B   11
#define SM_SDISHARE  (1<<SM_SDISHARE_B)
#define SM_SDINEW    (1<<SM_SDINEW_B)

void InitVS10xx(void) {
  SetGpioLow(GPIO1); // Reset VS10xx
  SetGpioHigh(GPIO3); // VS10xx xCS high
  SetGpioHigh(GPIO4); // VS10xx xDCS high (only needed if xDCS used)
  SetGpioHigh(GPIO5); // Other SPI device xCS high
  SetGpioHigh(GPIO1); // VS10xx out of reset
  // SCI_MODE_INIT_VAL separately defined for every case
  WriteSci(SCI_MODE, SCI_MODE_INIT_VAL);
  // If DREQ pin is not used, add a delay that is long enough.
  // See Datasheet Chapter Switching "Characteristics - Boot Initialization"
  // for how long to wait.
}
```

2.1 VS10xx as Only Device, 7 Pins

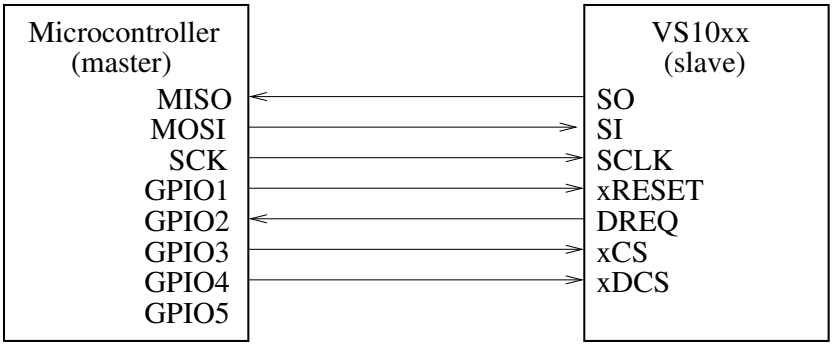

Figure 1: VS10xx as only device, 7-pin connection.

Figure 1 presents how to connect VS10xx to a microcontroller using 7 pins when it is the only device on the bus.

```
#define SCI_MODE_INIT_VAL SM_SDINEW
```

Pros:

- Straightforward implementation.

Cons:

- Requires 7 pins.

2.1.1 Microcontroller SCI Operations: VS10xx as Only Device, 7 Pins

```
void WriteSci(u_int8 addr, u_int16 data) {
  while (GetGpio(GPIO2) == 0); // Wait until DREQ is high

  SetGpioLow(GPIO3);       // Activate xCS
  WriteSpiByte(2);         // Write command code
  WriteSpiByte(addr);      // SCI register number
  WriteSpiByte((u_int8)(data >> 8));
  WriteSpiByte((u_int8)(data & 0xFF));
  SetGpioHigh(GPIO3);      // Deactivate xCS
}

u_int16 ReadSci(u_int8 addr) {
  u_int16 res;

  while (GetGpio(GPIO2) == 0); // Wait until DREQ is high

  SetGpioLow(GPIO3);       // Activate xCS
  WriteSpiByte(3);         // Read command code
  WriteSpiByte(addr);      // SCI register number
  res = (u_int16)ReadSpiByte() << 8;
  res |= ReadSpiByte();
  SetGpioHigh(GPIO3);      // Deactivate xCS
  return res;
}

/* VS1053, VS1063, VS1073 only. */
void WriteSciMulti(u_int8 addr, const u_int16 *data, int nWords) {
  while (GetGpio(GPIO2) == 0); // Wait until DREQ is high

  SetGpioLow(GPIO3);       // Activate xCS
  WriteSpiByte(2);         // Write command code
  WriteSpiByte(addr);      // SCI register number
  while (nWords--) {
    WriteSpiByte((u_int8)(*data   >> 8));
    WriteSpiByte((u_int8)(*data++ & 0xFF));
  }
  SetGpioHigh(GPIO3);      // Deactivate xCS
}

/* VS1073 only. */
void ReadSciMulti(u_int8 addr, u_int16 *data, int nWords) {
  while (GetGpio(GPIO2) == 0); // Wait until DREQ is high

  SetGpioLow(GPIO3);       // Activate xCS
  WriteSpiByte(3);         // Read command code
  WriteSpiByte(addr);      // SCI register number
  while (nWords--) {
    u_int16 t = (u_int16)ReadSpiByte() << 8;
    *data++ = t | ReadSpiByte();
  }
  SetGpioHigh(GPIO3);      // Deactivate xCS
}
```

2.1.2 Microcontroller SDI Operations: VS10xx as Only Device, 7 Pins

```
int WriteSdi(const u_int8 *data, u_int8 bytes) {
  u_int8 i;

  if (bytes > 32) return -1;   // Error: Too many bytes to transfer!

  while (GetGpio(GPIO2) == 0); // Wait until DREQ is high

  SetGpioLow(GPIO4);           // Activate xDCS
  for (i=0; i<bytes; i++) {
    WriteSpiByte(*data++);
  }
  SetGpioHigh(GPIO4);          // Dectivate xDCS

  return 0;                    // Ok
}
```

2.1.3 SCI Write Waveform: VS10xx as Only Device, 7 Pins

SCI_MODE register bits SM_DACT and SM_SDIORD are both at their default value 0.

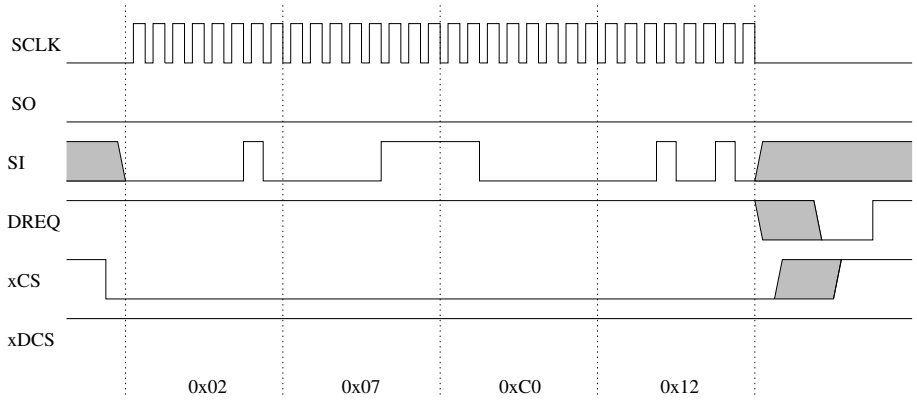

Figure 2: WriteSci(SCI_WRAMADDR, 0xC012)

Figure 2 presents an SCI write operation where we write 0xC012 to SCI_WRAMADDR.

First, we must wait until DREQ is high (or, take care that we don't send commands faster than VS10xx can process). We turn xCS low. Then we send the write command byte 0x02, followed by address SCI_WRAMADDR, which is 0x07. Then the 16-bit data value 0xC012 is sent. After all 32 bits have been clocked out, we turn xCS high. VS10xx will turn DREQ low for the time it takes to process the write command. When DREQ is again high, VS10xx is again capable of receiving data and/or commands.

SCI Multiple Write operation is similar, except for the actual data part (in this example 0xC0, 0x12), which is repeated to write multiple new values for the register.

2.1.4 SCI Read Waveform: VS10xx as Only Device, 7 Pins

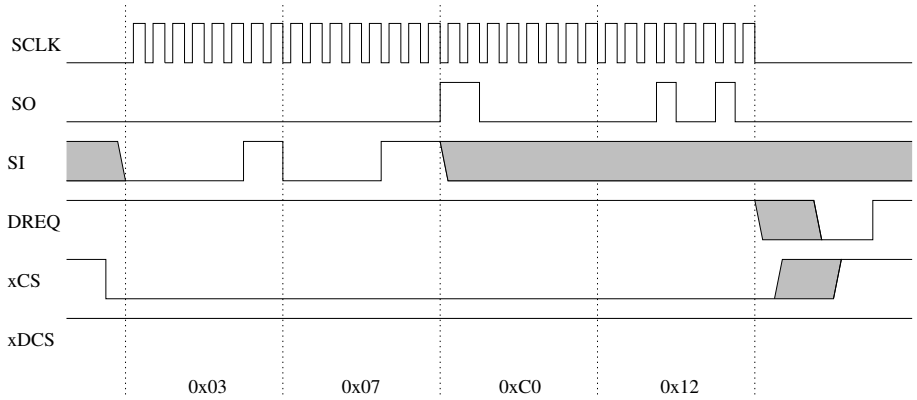

Figure 3: ReadSci(SCI_WRAMADDR) (result 0xC012)

In Figure 3 we read from the same register SCI_WRAMADDR we just wrote to. The operation is otherwise similar to the write command, except that the data is provided by VS10xx to SO. We should now get the same value we just write to the same register, in this case 0xC012.

SCI Multiple Read operation is similar, except for the actual data part (in this example 0xC0, 0x12), which is repeated to get multiple new values for the register.

2.1.5 SDI Write Waveform: VS10xx as Only Device, 7 Pins

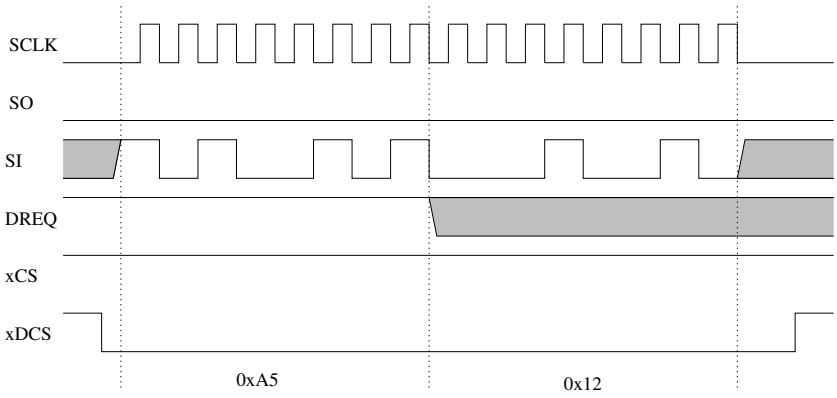

Figure 4: WriteSdi(data, 2): VS10xx as only device, 7 Pins

Figure 4 presents a 2-byte SDI write operation with data = 0xA5, 0x12. The SDI operation starts by turning xDCS low. Then the two bytes are sent. DREQ may go low while the transfer is going on, but as long as there are not more than 32 bytes in one transfer, this is normal. After the operation, xDCS is turned high.

2.2 VS10xx as Only Device, 6 Pins without xDCS

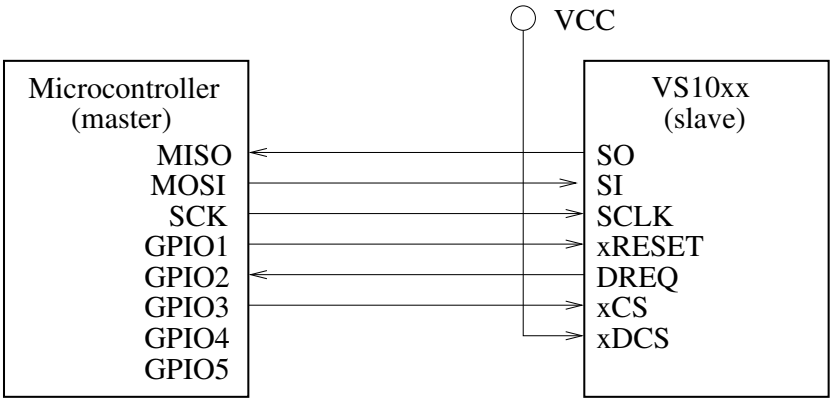

Figure 5: VS10xx as only device, 6-pin connection without xDCS.

In Figure 5 xDCS has been omitted. To be able to send data to SDI, register SCI_MODE bit SM_SDISHARED must be set. Then VS10xx's SDI bus will be active whenever xCS is high.

```
#define SCI_MODE_INIT_VAL (SM_SDINEW | SM_SDISHARED)
```

Pros:

- One pin less than worst case.

Cons:

- It is not possible to run "empty" SPI cycles: all SPI operations go either to the SCI or SDI bus (unless you temporarily deactivate SM_SDISHARED).
- Any SPI clock glitches while not doing SPI operations will go to the SDI bus (unless you temporarily deactivate SM_SDISHARED).

2.2.1 Microcontroller SCI Operations: VS10xx as Only Device, 6 Pins

```
void WriteSci(u_int8 addr, u_int16 data) {
  while (GetGpio(GPIO2) == 0);                      // Wait until DREQ is high

  SetGpioLow(GPIO3);       // Activate xCS
  WriteSpiByte(2);         // Write command code
  WriteSpiByte(addr);      // SCI register number
  WriteSpiByte((u_int8)(data >> 8));
  WriteSpiByte((u_int8)(data & 0xFF));
  SetGpioHigh(GPIO3);      // Deactivate xCS
}

u_int16 ReadSci(u_int8 addr) {
  u_int16 res;

  while (GetGpio(GPIO2) == 0);                      // Wait until DREQ is high

  SetGpioLow(GPIO3);       // Activate xCS
  WriteSpiByte(3);         // Read command code
  WriteSpiByte(addr);      // SCI register number
  res = (u_int16)ReadSpiByte() << 8;
  res |= ReadSpiByte();
  SetGpioHigh(GPIO3);      // Deactivate xCS
  return res;
}

/* VS1053, VS1063, VS1073 only. */
void WriteSciMulti(u_int8 addr, const u_int16 *data, int nWords) {
  while (GetGpio(GPIO2) == 0); // Wait until DREQ is high

  SetGpioLow(GPIO3);       // Activate xCS
  WriteSpiByte(2);         // Write command code
  WriteSpiByte(addr);      // SCI register number
  while (nWords--) {
    WriteSpiByte((u_int8)(*data   >> 8));
    WriteSpiByte((u_int8)(*data++ & 0xFF));
  }
  SetGpioHigh(GPIO3);      // Deactivate xCS
}

/* VS1073 only. */
void ReadSciMulti(u_int8 addr, u_int16 *data, int nWords) {
  while (GetGpio(GPIO2) == 0); // Wait until DREQ is high

  SetGpioLow(GPIO3);       // Activate xCS
  WriteSpiByte(3);         // Read command code
  WriteSpiByte(addr);      // SCI register number
  while (nWords--) {
    u_int16 t = (u_int16)ReadSpiByte() << 8;
    *data++ = t | ReadSpiByte();
  }
  SetGpioHigh(GPIO3);      // Deactivate xCS
}
```

2.2.2 Microcontroller SDI Operations: VS10xx as Only Device, 6 Pins

```
int WriteSdi(const u_int8 *data, u_int8 bytes) {
  u_int8 i;

  if (bytes > 32) return -1;   // Error: Too many bytes to transfer!

  while (GetGpio(GPIO2) == 0); // Wait until DREQ is high

  SetGpioHigh(GPIO3);          // Deactivate xCS = activate SDI
  for (i=0; i<bytes; i++) {
    WriteSpiByte(*data++);
  }
                               // xCS left deactivated

  return 0;                    // Ok
}
```

2.2.3 SCI Write Waveform: VS10xx as Only Device, 6 Pins

The waveform is the same as in Chapter 2.1.3.

2.2.4 SCI Read Waveform: VS10xx as Only Device, 6 Pins

The waveform is the same as in Chapter 2.1.4.

2.2.5 SDI Write Waveform: VS10xx as Only Device, 6 Pins

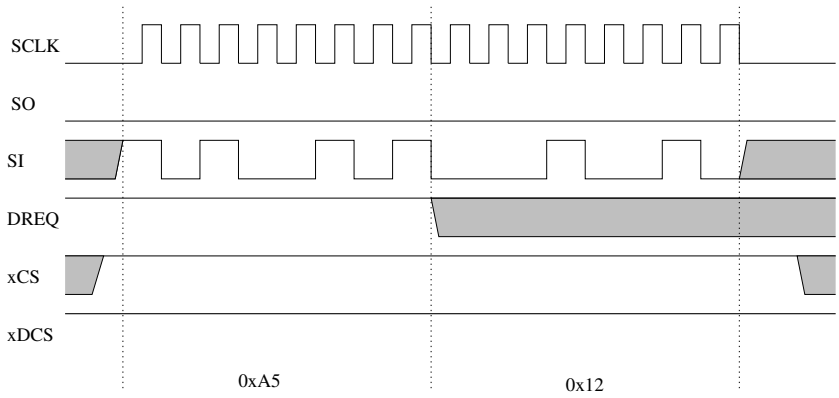

Figure 6: WriteSdi(data, 2): VS10xx as only device, 6 Pins

Figure 6 presents a 2-byte SDI write operation with data = 0xA5, 0x12. The SDI operation starts by making sure xCS is high. Then the two bytes are sent. DREQ may go low while the transfer is going on, but as long as there are not more than 32 bytes in one transfer, this is normal. Nothing needs to be done after the operation.

2.3 VS10xx as Only Device, 5 Pins

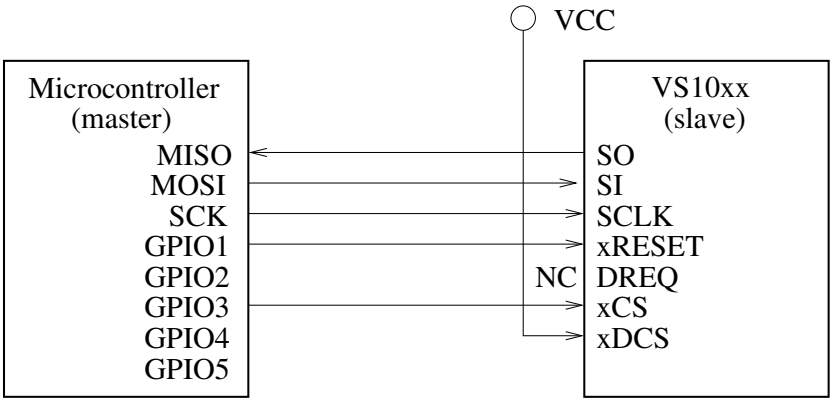

Figure 7: VS10xx as only device, 5-pin connection.

In Figure 7 both xDCS and DREQ been omitted. To be able to send data to SDI, register SCI_MODE bit SM_SDISHARED must be set. Then VS10xx's SDI bus will be active whenever xCS is high. DREQ status needs to be read through the SCI bus.

```
#define SCI_MODE_INIT_VAL (SM_SDINEW | SM_SDISHARED)
```

Pros:

- Minimum number of pins used.

Cons:

- Checking DREQ through SCI forces the user to add a delay after each of the potentially slow SCI register write (SCI_CLOCKF, SCI_AUDATA, SCI_AIADDR) operations. Otherwise some SCI write may be missed by VS10xx. Check Chapter *SCI Registers*, table *SCI registers*, field *Write Time* in the datasheet for your IC for exact number of clock cycles.
- Requires at least 8 bytes of extra data to be transferred for each SDI operation (one WriteSci() and one ReadSci()), creating an overhead of at least 25 %.
- Cannot be interrupt driven with rising edge of DREQ.
- It is not possible to run "empty" SPI cycles: all SPI operations go either to the SCI or SDI bus (unless you temporarily deactivate SM_SDISHARED).
- Any SPI clock glitches while not doing SPI operations will go to the SDI bus (unless you temporarily deactivate SM_SDISHARED).

2.3.1 Microcontroller SCI Operations: VS10xx as Only Device, 5 Pins

```
void WriteSci(u_int8 addr, u_int16 data) {
                            // Can't check DREQ, hopefully VS10xx is ready

  SetGpioLow(GPIO3);        // Activate xCS
  WriteSpiByte(2);          // Write command code
  WriteSpiByte(addr);       // SCI register number
  WriteSpiByte((u_int8)(data >> 8));
  WriteSpiByte((u_int8)(data & 0xFF));
  SetGpioHigh(GPIO3);       // Deactivate xCS
}

u_int16 ReadSci(u_int8 addr) {
  u_int16 res;
                            // Can't check DREQ, hopefully VS10xx is ready

  SetGpioLow(GPIO3);        // Activate xCS
  WriteSpiByte(3);          // Read command code
  WriteSpiByte(addr);       // SCI register number
  res = (u_int16)ReadSpiByte() << 8;
  res |= ReadSpiByte();
  SetGpioHigh(GPIO3);       // Deactivate xCS
  return res;
}

/* VS1053, VS1063, VS1073 only. */
void WriteSciMulti(u_int8 addr, const u_int16 *data, int nWords) {
                            // Can't check DREQ, hopefully VS10xx is ready

  SetGpioLow(GPIO3);        // Activate xCS
  WriteSpiByte(2);          // Write command code
  WriteSpiByte(addr);       // SCI register number
  while (nWords--) {
    WriteSpiByte((u_int8)(*data   >> 8));
    WriteSpiByte((u_int8)(*data++ & 0xFF));
  }
  SetGpioHigh(GPIO3);       // Deactivate xCS
}

/* VS1073 only. */
void ReadSciMulti(u_int8 addr, u_int16 *data, int nWords) {
                            // Can't check DREQ, hopefully VS10xx is ready

  SetGpioLow(GPIO3);        // Activate xCS
  WriteSpiByte(3);          // Read command code
  WriteSpiByte(addr);       // SCI register number
  while (nWords--) {
    u_int16 t = (u_int16)ReadSpiByte() << 8;
    *data++ = t | ReadSpiByte();
  }
  SetGpioHigh(GPIO3);       // Deactivate xCS
}
```

2.3.2 Microcontroller SDI Operations: VS10xx as Only Device, 5 Pins

```
int WriteSdi(const u_int8 *data, u_int8 bytes) {
  u_int8 i;
  u_int16 dreq;

  if (bytes > 32) return -1;    // Error: Too many bytes to transfer!

  do {                          // Wait until DREQ is high
    dreq = ReadDreq();
    if (!dreq) {                // If DREQ low, wait 1 millisecond
      WaitMicroseconds(1000);   // before next read so that SCI
    }                           // registers aren't read all the time
  } while (!dreq);

                                // ReadSci left xCS deactivated = SDI active
  for (i=0; i<bytes; i++) {
    WriteSpiByte(*data++);
  }
                                // xCS left deactivated

  return 0;                     // Ok
}
```

2.3.3 SCI Write Waveform: VS10xx as Only Device, 5 Pins

The waveform is the same as in Chapter 2.1.3.

2.3.4 SCI Read Waveform: VS10xx as Only Device, 5 Pins

The waveform is the same as in Chapter 2.1.4.

2.3.5 SDI Write Waveform: VS10xx as Only Device, 5 Pins

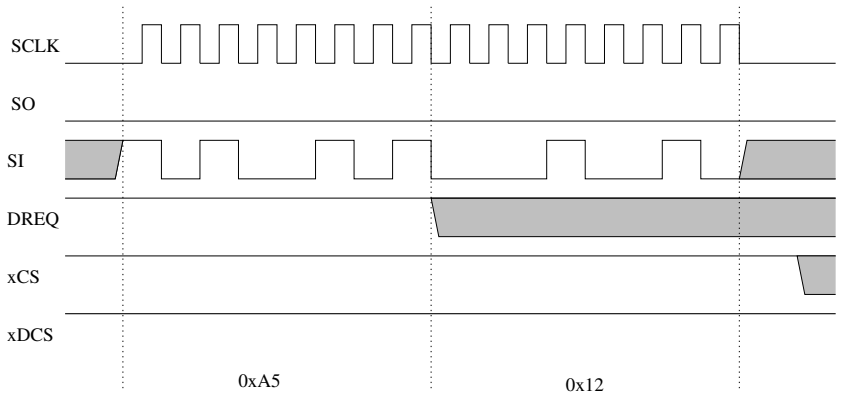

Figure 8: Transfer loop of WriteSdi(data, 2): VS10xx as only device, 5 pins

Figure 8 presents the for() transfer loop of a short, 2-byte SDI write operation with data = 0xA5, 0x12. The DREQ checking part of the function has already turned xCS high. So the only thing that happens is that the two bytes are sent. Although unconnected to the microcontroller, DREQ may go low while the transfer is going on, but as long as there are not more than 32 bytes in one transfer, this is normal. Nothing needs to be done after the operation.

2.4 VS10xx as One SPI Device of Many, 7 Pins

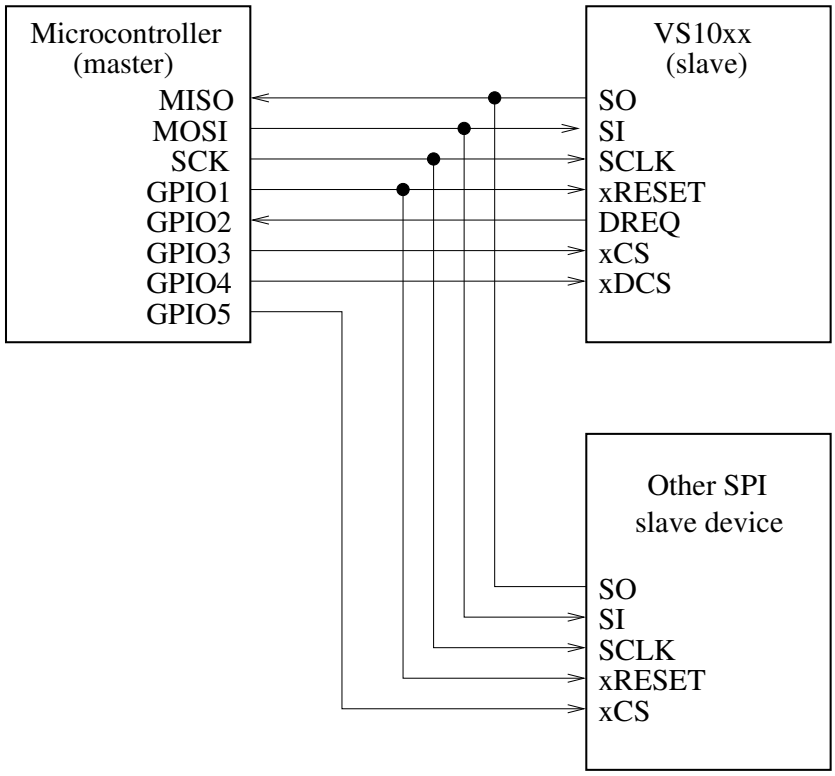

Figure 9: VS10xx as one SPI device of many, 7-pin connection.

Figure 9 presents how to connect VS10xx to a microcontroller using 7 pins when it there are more than one device on the bus. From the point of view of VS10xx, this is similar to the 7-pin connection in Chapter 2.1.

```
#define SCI_MODE_INIT_VAL SM_SDINEW
```

Pros:

- Straightforward implementation.

Cons:

- Requires 7 pins.

2.4.1 Microcontroller SCI and SDI Operations and Waveforms: VS10xx as One SPI Device, 7 Pins

SCI and SDI operations, as well as SCI and SDI waveforms, are exactly the same as in Chapter 2.1.

2.5 VS10xx as One SPI Device of Many, 6 Pins without xDCS

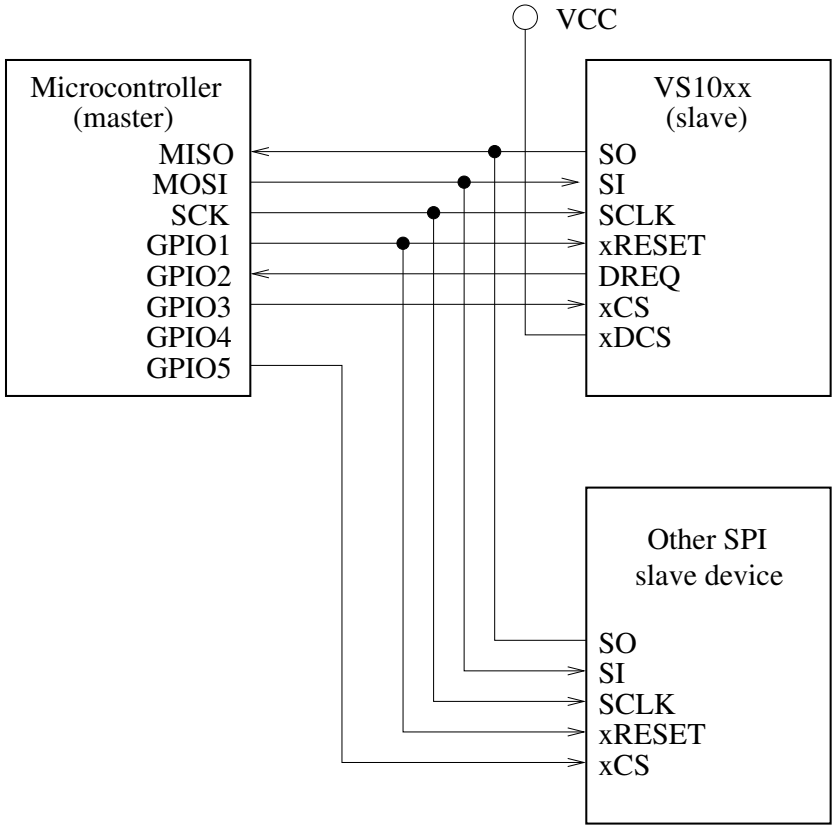

Figure 10: VS10xx as one SPI device of many, 6-pin connection without xDCS.

In Figure 10 xDCS has been omitted. To be able to send data to SDI, bit SM_SDISHARED must be switched on and off in register SCI_MODE. VS10xx's SDI will be active when SM_SDISHARED is high and xCS is high.

```
#define SCI_MODE_INIT_VAL SM_SDINEW
```

Pros:

- One pin less than worst case.

Cons:

- Requires twiddling with SM_SDISHARED.
- Each SDI write operation requires 8 extra bytes of data to be transferred (two additional SciWrite() operations to modify register SCI_MODE), creating an overhead of at least 25 %.

2.5.1 Microcontroller SCI Operations: VS10xx as One SPI Device, 6 Pins without xDCS

```
void WriteSci(u_int8 addr, u_int16 data) {
  while (GetGpio(GPIO2) == 0); // Wait until DREQ is high

  SetGpioLow(GPIO3);       // Activate xCS
  WriteSpiByte(2);         // Write command code
  WriteSpiByte(addr);      // SCI register number
  WriteSpiByte((u_int8)(data >> 8));
  WriteSpiByte((u_int8)(data & 0xFF));
  SetGpioHigh(GPIO3);      // Deactivate xCS
}

u_int16 ReadSci(u_int8 addr) {
  u_int16 res;

  while (GetGpio(GPIO2) == 0); // Wait until DREQ is high

  SetGpioLow(GPIO3);       // Activate xCS
  WriteSpiByte(3);         // Read command code
  WriteSpiByte(addr);      // SCI register number
  res = (u_int16)ReadSpiByte() << 8;
  res |= ReadSpiByte();
  SetGpioHigh(GPIO3);      // Deactivate xCS
  return res;
}

/* VS1053, VS1063, VS1073 only. */
void WriteSciMulti(u_int8 addr, const u_int16 *data, int nWords) {
  while (GetGpio(GPIO2) == 0); // Wait until DREQ is high

  SetGpioLow(GPIO3);       // Activate xCS
  WriteSpiByte(2);         // Write command code
  WriteSpiByte(addr);      // SCI register number
  while (nWords--) {
    WriteSpiByte((u_int8)(*data   >> 8));
    WriteSpiByte((u_int8)(*data++ & 0xFF));
  }
  SetGpioHigh(GPIO3);      // Deactivate xCS
}

/* VS1073 only. */
void ReadSciMulti(u_int8 addr, u_int16 *data, int nWords) {
  while (GetGpio(GPIO2) == 0); // Wait until DREQ is high

  SetGpioLow(GPIO3);       // Activate xCS
  WriteSpiByte(3);         // Read command code
  WriteSpiByte(addr);      // SCI register number
  while (nWords--) {
    u_int16 t = (u_int16)ReadSpiByte() << 8;
    *data++ = t | ReadSpiByte();
  }
  SetGpioHigh(GPIO3);      // Deactivate xCS
}
```

2.5.2 Microcontroller SDI Operations: VS10xx as One SPI Device, 6 Pins without xDCS

```
int WriteSdi(const u_int8 *data, u_int8 bytes) {
  u_int8 i;

  if (bytes > 32) return -1;   // Error: Too many bytes to transfer!

  while (GetGpio(GPIO2) == 0); // Wait until DREQ is high

                               // Set shared chip select mode. Activates SDI.
  WriteSci(SCI_MODE, SCI_MODE_INIT_VAL | SM_SDISHARED);
  for (i=0; i<bytes; i++) {
    WriteSpiByte(*data++);
  }
                               // Clear shared chip select mode. Deactivates SDI.
  WriteSci(SCI_MODE, SCI_MODE_INIT_VAL);

  return 0;                    // Ok
}
```

2.5.3 SCI Write Waveform: VS10xx as One SPI Device, 6 Pins without xDCS

The waveform is the same as in Chapter 2.1.3.

2.5.4 SCI Read Waveform: VS10xx as One SPI Device, 6 Pins without xDCS

The waveform is the same as in Chapter 2.1.4.

2.5.5 SDI Write Waveform: VS10xx as One SPI Device, 6 Pins without xDCS

The waveform is the same as in Chapter 2.2.5.

2.6 VS10xx as One SPI Device of Many, 6 Pins without DREQ

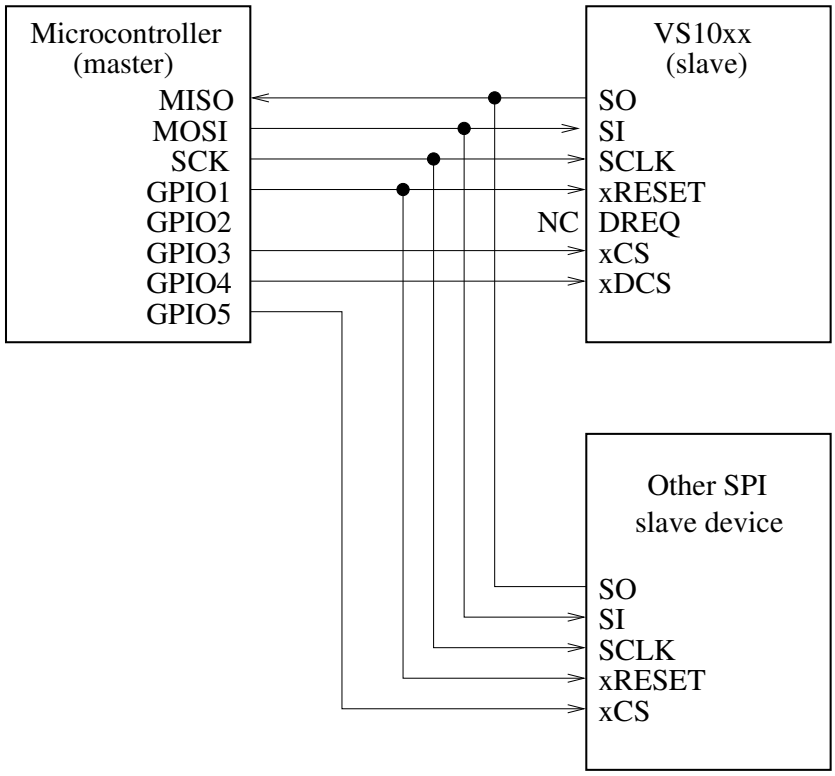

Figure 11: VS10xx as one SPI device of many, 6-pin connection without DREQ.

Figure 11 presents one way how to connect VS10xx to a microcontroller using 6 pins when it there are more than one device on the bus. The difference with the 7-pin connection is the removal of the DREQ signal.

```
#define SCI_MODE_INIT_VAL SM_SDINEW
```

Pros:

- One pin less than worst case.

Cons:

- Checking DREQ through SCI forces the user to add a delay after each of the potentially slow SCI register write (SCI_CLOCKF, SCI_AUDATA, SCI_AIADDR) operations. Otherwise some SCI write may be missed by VS10xx. Check Chapter *SCI Registers*, table *SCI registers*, field *Write Time* in the datasheet for your IC for exact number of clock cycles.
- Requires at least 8 bytes of extra data to be transferred for each SDI operation (one WriteSci() and one ReadSci()), creating an overhead of at least 25 %.
- Cannot be interrupt driven with rising edge of DREQ.

2.6.1 Microcontroller SCI Operations: VS10xx as One SPI Device, 6 Pins without DREQ

```
void WriteSci(u_int8 addr, u_int16 data) {
                            // Can't check DREQ, hopefully VS10xx is ready

  SetGpioLow(GPIO3);        // Activate xCS
  WriteSpiByte(2);          // Write command code
  WriteSpiByte(addr);       // SCI register number
  WriteSpiByte((u_int8)(data >> 8));
  WriteSpiByte((u_int8)(data & 0xFF));
  SetGpioHigh(GPIO3);       // Deactivate xCS
}

u_int16 ReadSci(u_int8 addr) {
  u_int16 res;
                            // Can't check DREQ, hopefully VS10xx is ready

  SetGpioLow(GPIO3);        // Activate xCS
  WriteSpiByte(3);          // Read command code
  WriteSpiByte(addr);       // SCI register number
  res = (u_int16)ReadSpiByte() << 8;
  res |= ReadSpiByte();
  SetGpioHigh(GPIO3);       // Deactivate xCS
  return res;
}

/* VS1053, VS1063, VS1073 only. */
void WriteSciMulti(u_int8 addr, const u_int16 *data, int nWords) {
                            // Can't check DREQ, hopefully VS10xx is ready

  SetGpioLow(GPIO3);        // Activate xCS
  WriteSpiByte(2);          // Write command code
  WriteSpiByte(addr);       // SCI register number
  while (nWords--) {
    WriteSpiByte((u_int8)(*data   >> 8));
    WriteSpiByte((u_int8)(*data++ & 0xFF));
  }
  SetGpioHigh(GPIO3);       // Deactivate xCS
}

/* VS1073 only. */
void ReadSciMulti(u_int8 addr, u_int16 *data, int nWords) {
                            // Can't check DREQ, hopefully VS10xx is ready

  SetGpioLow(GPIO3);        // Activate xCS
  WriteSpiByte(3);          // Read command code
  WriteSpiByte(addr);       // SCI register number
  while (nWords--) {
    u_int16 t = (u_int16)ReadSpiByte() << 8;
    *data++ = t | ReadSpiByte();
  }
  SetGpioHigh(GPIO3);       // Deactivate xCS
}
```

2.6.2 Microcontroller SDI Operations: VS10xx as One SPI Device, 6 Pins without DREQ

```
int WriteSdi(const u_int8 *data, u_int8 bytes) {
  u_int8 i;
  u_int16 dreq;

  if (bytes > 32) return -1;   // Error: Too many bytes to transfer!
  }

  do {                         // Wait until DREQ is high
    dreq = ReadDreq();
    if (!dreq) {               // If DREQ low, wait 1 millisecond
      WaitMicroseconds(1000);  // before next read so that SCI
    }                          // registers aren't read all the time
  } while (!dreq);

  SetGpioLow(GPIO4);           // Activate xDCS
  for (i=0; i<bytes; i++) {
    WriteSpiByte(*data++);
  }
  SetGpioHigh(GPIO4);          // Deactivate xDCS

  return 0;                    // Ok
}
```

2.6.3 SCI Write Waveform: VS10xx as One SPI Device, 6 Pins without DREQ

The waveform is the same as in Chapter 2.1.3.

2.6.4 SCI Read Waveform: VS10xx as One SPI Device, 6 Pins without DREQ

The waveform is the same as in Chapter 2.1.4.

2.6.5 SDI Write Waveform: VS10xx as One SPI Device, 6 Pins without DREQ

Figure 4 on Page 11 presents the for() transfer loop for a 2-byte SDI write operation with data = 0xA5, 0x12. The DREQ checking part has been omitted from the figure. The SDI operation starts by turning xDCS low. Then the two bytes are sent. Although unconnected to the microcontroller, DREQ may go low while the transfer is going on, but as long as there are not more than 32 bytes in one transfer, this is normal. After the operation, xDCS is turned high.

2.7 VS10xx as One SPI Device of Many, 5 Pins

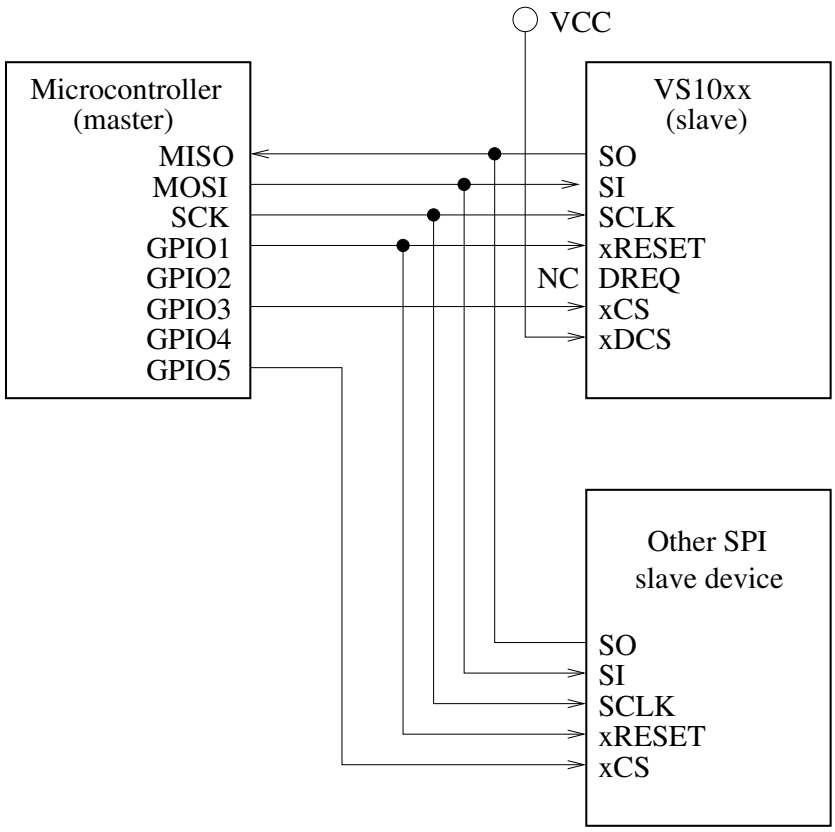

Figure 12: VS10xx as one SPI device of many, 5-pin connection.

In Figure 12 both xDCS and DREQ been omitted. To be able to send data to SDI, bit SM_SDISHARED must be activated in register SCI_MODE, and DREQ status needs to be read through the SCI bus. Now VS10xx's SDI bus will be active when xCS is high.

```
#define SCI_MODE_INIT_VAL SM_SDINEW
```

Pros:

- Minimum number of pins used.

Cons:

- Checking DREQ through SCI forces the user to add a delay after each of the potentially slow SCI register write (SCI_CLOCKF, SCI_AUDATA, SCI_AIADDR) operations. Otherwise some SCI write may be missed by VS10xx. Check Chapter *SCI Registers*, table *SCI registers*, field *Write Time* in the datasheet for your IC for exact number of clock cycles.
- Requires at least 16 bytes of extra data to be transferred for each SDI operation (at least one WriteSci() and one ReadSci() to check DREQ, then two WriteSci()'s to change SCI_MODE), creating an overhead of at least 50 %.
- Requires twiddling with SM_SDISHARED.
- Cannot be interrupt driven with rising edge of DREQ.

2.7.1 Microcontroller SCI Operations: VS10xx as One SPI Device, 5 Pins

```
void WriteSci(u_int8 addr, u_int16 data) {
                            // Can't check DREQ, hopefully VS10xx is ready

  SetGpioLow(GPIO3);        // Activate xCS
  WriteSpiByte(2);          // Write command code
  WriteSpiByte(addr);       // SCI register number
  WriteSpiByte((u_int8)(data >> 8));
  WriteSpiByte((u_int8)(data & 0xFF));
  SetGpioHigh(GPIO3);       // Deactivate xCS
}

u_int16 ReadSci(u_int8 addr) {
  u_int16 res;
                            // Can't check DREQ, hopefully VS10xx is ready

  SetGpioLow(GPIO3);        // Activate xCS
  WriteSpiByte(3);          // Read command code
  WriteSpiByte(addr);       // SCI register number
  res = (u_int16)ReadSpiByte() << 8;
  res |= ReadSpiByte();
  SetGpioHigh(GPIO3);       // Deactivate xCS
  return res;
}

/* VS1053, VS1063, VS1073 only. */
void WriteSciMulti(u_int8 addr, const u_int16 *data, int nWords) {
                            // Can't check DREQ, hopefully VS10xx is ready

  SetGpioLow(GPIO3);        // Activate xCS
  WriteSpiByte(2);          // Write command code
  WriteSpiByte(addr);       // SCI register number
  while (nWords--) {
    WriteSpiByte((u_int8)(*data   >> 8));
    WriteSpiByte((u_int8)(*data++ & 0xFF));
  }
  SetGpioHigh(GPIO3);       // Deactivate xCS
}

/* VS1073 only. */
void ReadSciMulti(u_int8 addr, u_int16 *data, int nWords) {
                            // Can't check DREQ, hopefully VS10xx is ready

  SetGpioLow(GPIO3);        // Activate xCS
  WriteSpiByte(3);          // Read command code
  WriteSpiByte(addr);       // SCI register number

  while (nWords--) {
    u_int16 t = (u_int16)ReadSpiByte() << 8;
    *data++ = t | ReadSpiByte();
  }
  SetGpioHigh(GPIO3);       // Deactivate xCS
}
```

2.7.2 Microcontroller SDI Operations: VS10xx as One SPI Device, 5 Pins

```
int WriteSdi(const u_int8 *data, u_int8 bytes) {
  u_int8 i;
  u_int16 dreq;

  if (bytes > 32) return -1;   // Error: Too many bytes to transfer!

  do {                         // Wait until DREQ is high
    dreq = ReadDreq();
    if (!dreq) {               // If DREQ low, wait 1 millisecond
      WaitMicroseconds(1000);  // before next read so that SCI
    }                          // registers aren't read all the time
  } while (!dreq);

                               // Set shared chip select mode. Activates SDI.
  WriteSci(SCI_MODE, SCI_MODE_INIT_VAL | SM_SDISHARED);
  for (i=0; i<bytes; i++) {
    WriteSpiByte(*data++);
  }
                               // Clear shared chip select mode. Deactivates SDI.
  WriteSci(SCI_MODE, SCI_MODE_INIT_VAL);

  return 0;                    // Ok
}
```

2.7.3 SCI Write Waveform: VS10xx as One SPI Device, 5 Pins

The waveform is the same as in Chapter 2.1.3.

2.7.4 SCI Read Waveform: VS10xx as One SPI Device, 5 Pins

The waveform is the same as in Chapter 2.1.4.

2.7.5 SDI Write Waveform: VS10xx as One SPI Device, 5 Pins

The waveform is the same as in Chapter 2.3.5.

3 Latest Version Changes

Version 1.10, 2025-04-01

- Added multiple write SCI function SciWriteMulti() for VS1073, VS1063, and VS1053 to all examples.
- Added multiple read SCI function SciReadMulti() for VS1073 to all examples.
- Added VS1073 tro Chapter 1.1, *Reading DREQ Without a Dedicated Pin*, and renamed the Chapter.

Version 1.01, 2012-12-01

- The DREQ read address was incorrect. It has been replaced with a chip-specific address which is presented in a new Chapter 1.1, *DREQ Address DREQ_ADDR*.
- Some prototypes for ReadSci() were `u_int16 ReadSci(u_int16 addr)` instead of `u_int16 ReadSci(u_int8 addr)`. Corrected.

Version 1.00, 2012-11-21

Initial version.

4 Contact Information

VLSI Solution Oy
Entrance G, 2nd floor
Hermiankatu 8
FI-33720 Tampere
FINLAND

URL: http://www.vlsi.fi/
Phone: +358-50-462-3200
Commercial e-mail: sales@vlsi.fi

For technical support or suggestions regarding this document, please participate at
http://www.vsdsp-forum.com/
For confidential technical discussions, contact
support@vlsi.fi

For technical questions or suggestions regarding this document, please participate at
`http://www.vsdsp-forum.com/`
or contact
`support@vlsi.fi`.