BIRMINGHAM City University

Data Acquisition and Analysis using SPI memory

(with specific reference to the PIC)

Data acquisition to SPI memory

Why?

…. Because we can’t trail a cable around the maze without difficulty and without affecting the performance of the mouse. The same holds true for any mobile data acquisition. Other methods of transmiting data have their own problems…..

How?

1. Write data at each δt to non-volatile memory with an SPI interface.

2. Upload the data to a PC/laptop for analysis after a run.

BIRMINGHAM City University

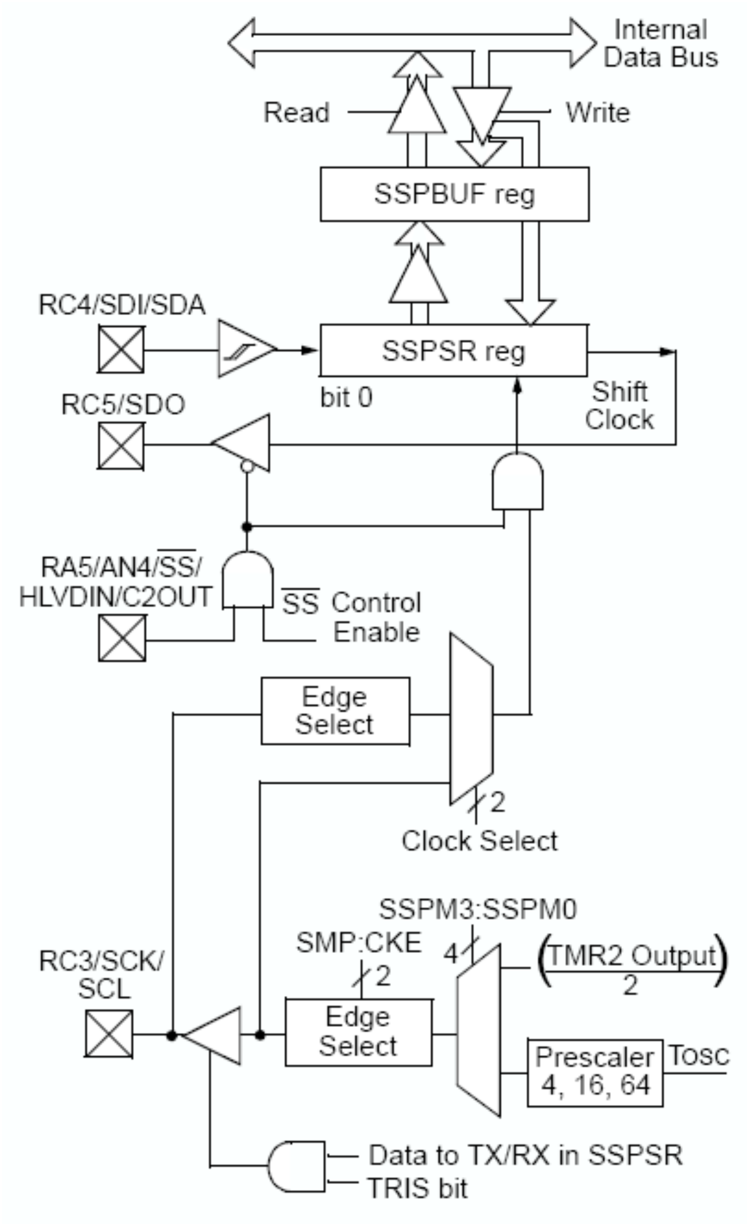
SPI: Serial Peripheral Interface

SPI requires 3 lines: SCK, SDO, SDI to transfer bidirectional data.

If two or more devices share the bus, each device must have its own select line (CS) – i.e. ‘hard-addressing’.

SPI is therefore a very easy system to implement, either using software bit-banging, or using the PIC MSSP module.

SPI on the PIC18FXXX

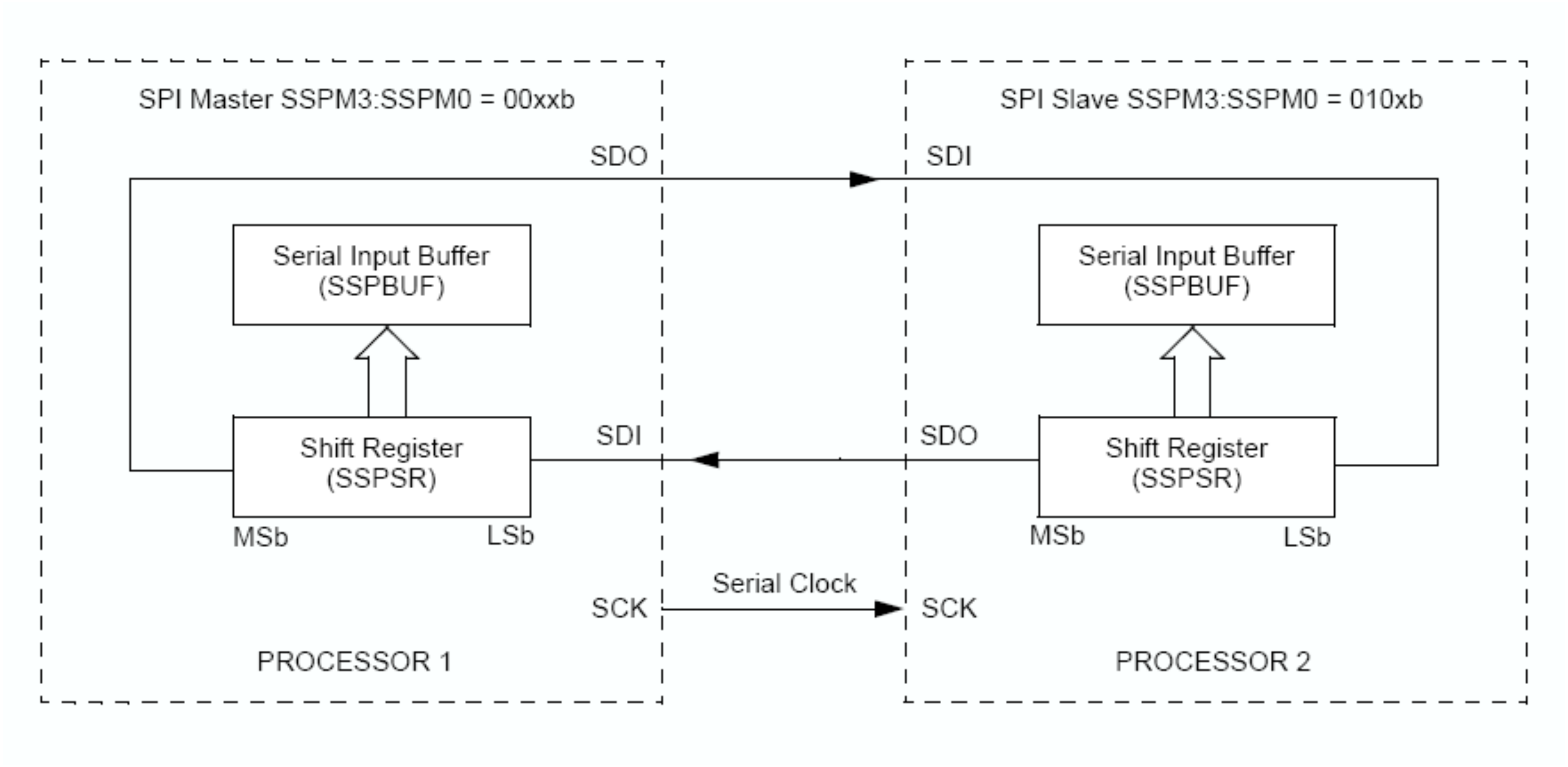


Three pins are used for data transfer:

- Serial Data Out (SDOx) (RC5/RD4)
- Serial Data In (SDIx) (RC4/RD5)
- Serial Clock (SCKx) (RC3/RD6)

Port C is normally used for SPI - Port D is used on 100 pin devices.

SPI ‘Data Exchange’ Mechanism


SPI Master SSPM3:SSPM0 = 00xxb
SPI Slave SSPM3:SSPM0 = 010xb
SDO
SDI
Serial Input Buffer (SSPBUF)
Serial Input Buffer (SSPBUF)
SDI
SDO
Shift Register (SSPSR)
Shift Register (SSPSR)
MSb
LSb
MSb
LSb
Serial Clock
SCK
SCK
PROCESSOR 1
PROCESSOR 2

As a data bit is clocked out from the Master into the Slave, a data bit is clocked out from the Slave into the Master. The Master controls the clock, and so controls the rate as well as the number of bits clocked out.

The PIC18FXXX SPI Registers

Each MSSP module has four registers for SPI mode operation. These are:

- Control Register 1 (SSPCON1)
- Status Register (SSPSTAT)
- Serial Receive/Transmit Buffer Register (SSPBUF)
- Shift Register (SSPSR)

The last register is not directly accessible - only the first 3 are used by the programmer.

BIRMINGHAM City University

PIC18FXXX Control Registers for SPI

SSPSTAT: MSSP STATUS REGISTER (SPI MODE)

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE[(1)]	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF
bit 7							bit 0

I2C mode only

bit 7 **SMP:** Sample bit

SPI Master mode:

1 = Input data sampled at end of data output time

0 = Input data sampled at middle of data output time

SPI Slave mode:

SMP must be cleared when SPI is used in Slave mode.

bit 6 **CKE:** SPI Clock Select bit[(1)]

1 = Transmit occurs on transition from active to Idle clock state

0 = Transmit occurs on transition from Idle to active clock state

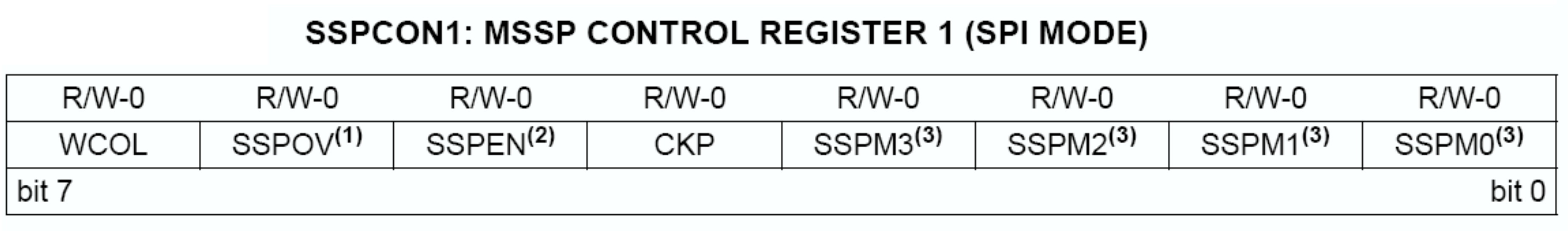
bit 0 **BF:** Buffer Full Status bit (Receive mode only)

1 = Receive complete, SSPBUF is full

0 = Receive not complete, SSPBUF is empty

Note 1: Polarity of clock state is set by the CKP bit (SSPCON1<4>).

Control Registers for SPI



SSPCON1: MSSP CONTROL REGISTER 1 (SPI MODE)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV[1]	SSPEN[2]	CKP	SSPM3[3]	SSPM2[3]	SSPM1[3]	SSPM0[3]
bit 7							bit 0

bit 7 **WCOL:** Write Collision Detect bit

bit 6 **SSPOV:** Receive Overflow Indicator bit (Slave only)

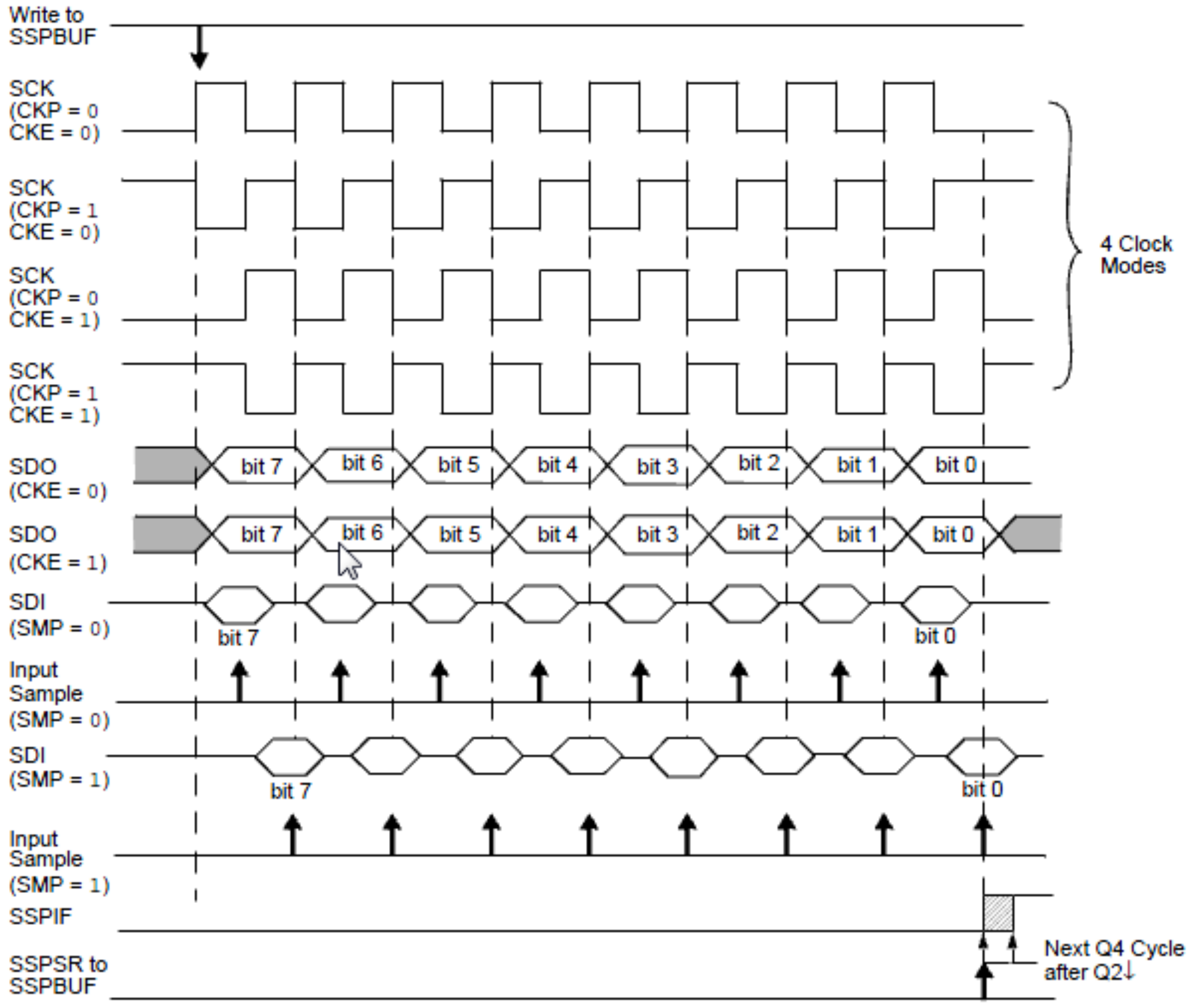
bit 5 **SSPEN:** Master Synchronous Serial Port Enable bit

bit 4 **CKP:** Clock Polarity Select bit

bit 3-0 **SSPM3:SSPM0:** Master Synchronous Serial Port Mode Select bits
0101 = SPI Slave mode, clock = SCK pin, SS pin control disabled
0100 = SPI Slave mode, clock = SCK pin, SS pin control enabled
0011 = SPI Master mode, clock = TMR2 output/2
0010 = SPI Master mode, clock = FOSC/64
0001 = SPI Master mode, clock = FOSC/16
0000 = SPI Master mode, clock = FOSC/4

(Refer to 18F4520 DATA sheet for more detail)

Data Transfer Protocols



The SPI protocol as specified by the 18F4520 datasheet is simply a byte exchange protocol.

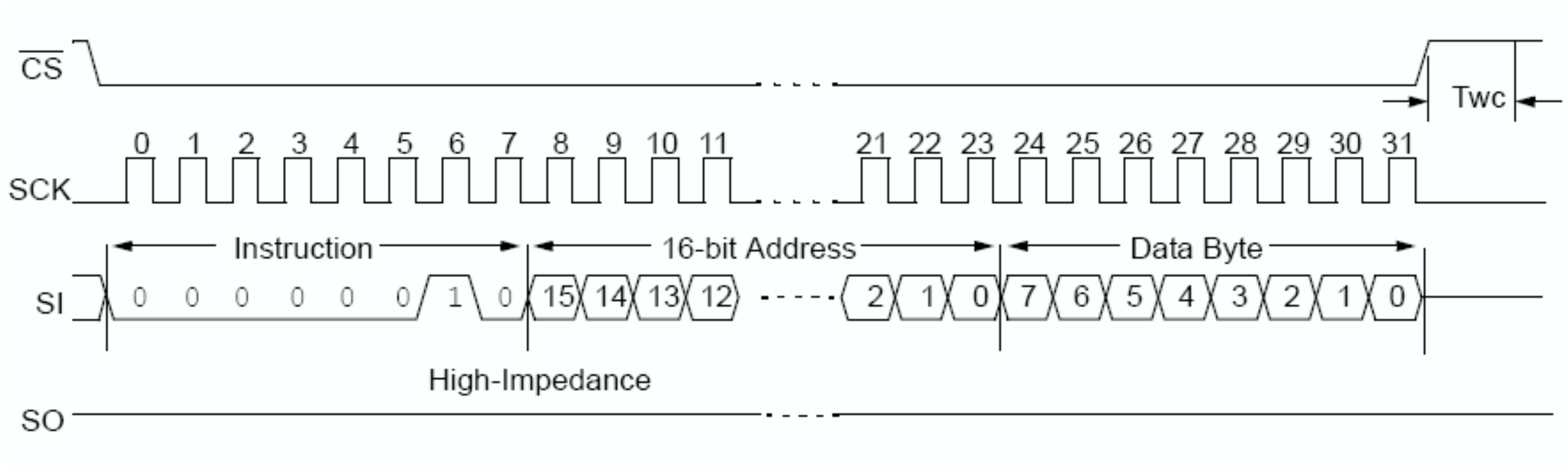
It functions at the lowest level of data transfer – 8 bits of data are exchanged with the slave.

There is no meaning attached to the data at this level.

BIRMINGHAM City University

Data Transfer Protocols: Higher Level

To interface to other devices using SPI it is most often necessary to implement a higher-level protocol that uses the byte-exchange mechanism of SPI, but that strings bytes together into a frame, each byte of which has a specific meaning.



For example, the diagram above shows a byte that corresponds to a Write instruction, followed by a 16-bit address (2 bytes for high and low), and finally the data to be written to that address. A program must be written to implement this higher level command protocol, using low-level SPI functions.

C18 Compiler Support for SPI

The C18 compiler provides a library of C functions to help us to initialise SPI mode, and to read and write to SSPBUF (G:\MCC18\doc\periph-lib)

Function	Description
`CloseSPI`	**Disable the SSP module used for SPI™ communications.**
`DataRdySPI`	**Determine if a new value is available from the SPI buffer.**
`getcSPI`	**Read a byte from the SPI bus.**
`getsSPI`	**Read a string from the SPI bus.**
`OpenSPI`	**Initialize the SSP module used for SPI communications.**
`putcSPI`	**Write a byte to the SPI bus.**
`putsSPI`	**Write a string to the SPI bus.**
`ReadSPI`	**Read a byte from the SPI bus.**
`WriteSPI`	**Write a byte to the SPI bus.**

BIRMINGHAM City University

Example C18 function: OpenSPI()

General form: void OpenSPI(unsigned char ***sync_mode***,
unsigned char ***bus_mode***,
unsigned char ***smp_phase***);

sync_mode	SPI_FOSC_4	SPI Master mode, clock = Fosc/4
	SPI_FOSC_16	SPI Master mode, clock = Fosc/16
	SPI_FOSC_64	SPI Master mode, clock = Fosc/64
	SPI_FOSC_TMR2	SPI Master mode, clock = TMR2 output/2
	SLV_SSON	SPI Slave mode, /SS pin control enabled
	SLV_SSOFF	SPI Slave mode, /SS pin control disabled

bus_mode	MODE_00	Setting for SPI bus Mode 0,0
	MODE_01	Setting for SPI bus Mode 0,1
	MODE_10	Setting for SPI bus Mode 1,0
	MODE_11	Setting for SPI bus Mode 1,1
smp_phase	SMPEND	Input data sample at end of data out
	SMPMID	Input data sample at middle of data out

Standard SPI Mode Terminology	Control Bits State	
	CKP	CKE
0, 0	0	1
0, 1	0	0
1, 0	1	1
1, 1	1	0

Idle state = LOW

TX on active to idle

For example, initialising the eeprom:

```
void EE_Init(void)
{
        CSEEPROM = INACTIVE;
        OpenSPI(SPI_FOSC_4,MODE_00,SMPMID); //Set up SPI
}
```

BIRMINGHAM City University

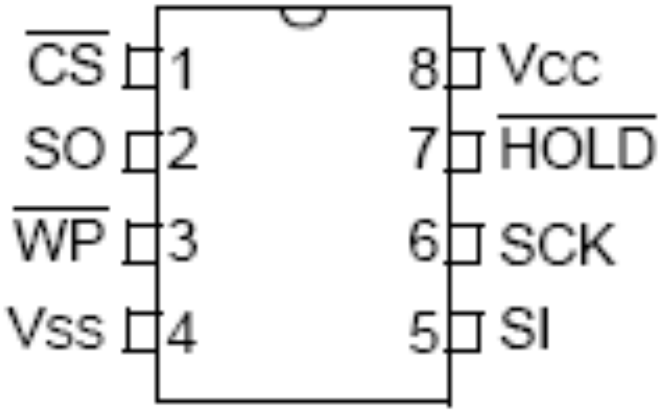
SPI Serial EEPROM

The Microchip 25XX256 are 256 KBit Serial Electrically Erasable PROMs with the following features:

- 32,768 x 8-bit Organization
- 64-Byte Page
- Self-Timed Erase and Write Cycles (5 ms max.)
- Vcc range 1.8-5.5V
- Access to the device is controlled through a Chip Select (CS) input.

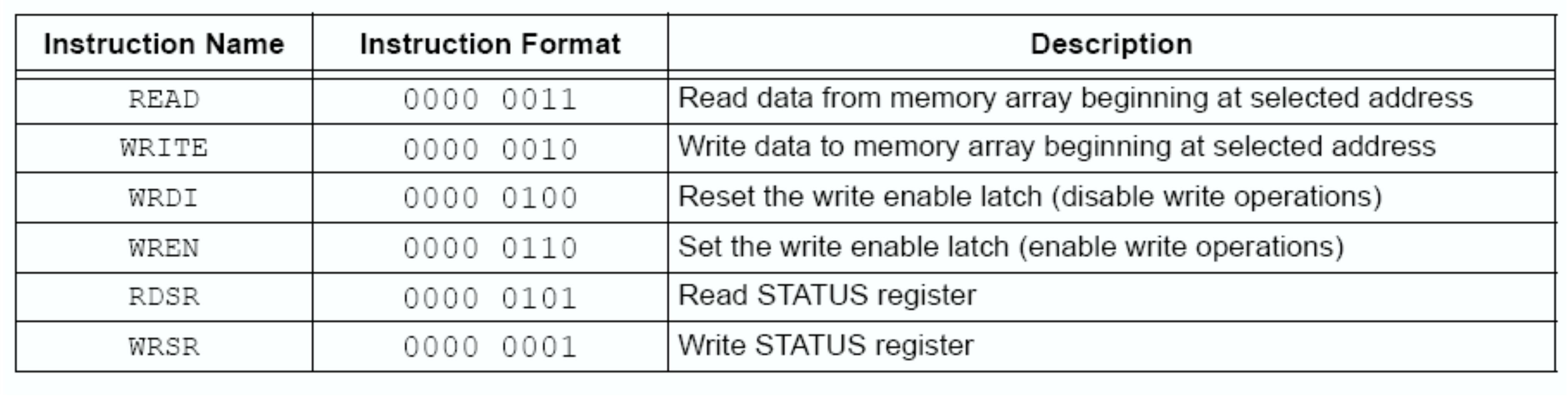
Pin Function Table

Name	Function
$\overline{\text{CS}}$	Chip Select Input
SO	Serial Data Output
$\overline{\text{WP}}$	Write-Protect
Vss	Ground
SI	Serial Data Input
SCK	Serial Clock Input
$\overline{\text{HOLD}}$	Hold Input
Vcc	Supply Voltage



Data Transfer Protocol

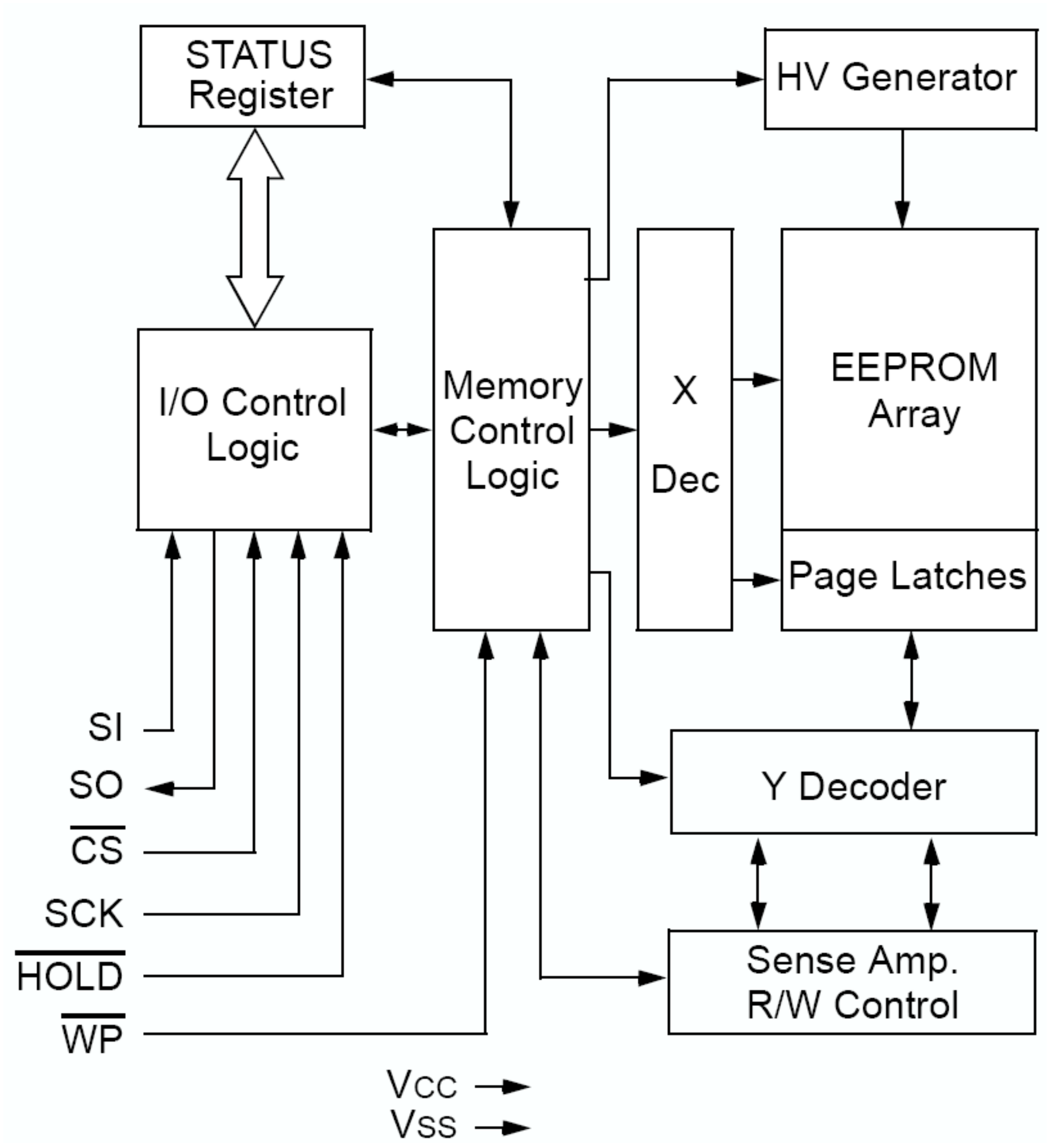
The 25XX256 contains an 8-bit instruction register. The first byte of a data transfer sequence will be the instruction.



Instruction Name	Instruction Format	Description
READ	0000 0011	Read data from memory array beginning at selected address
WRITE	0000 0010	Write data to memory array beginning at selected address
WRDI	0000 0100	Reset the write enable latch (disable write operations)
WREN	0000 0110	Set the write enable latch (enable write operations)
RDSR	0000 0101	Read STATUS register
WRSR	0000 0001	Write STATUS register

BIRMINGHAM City University

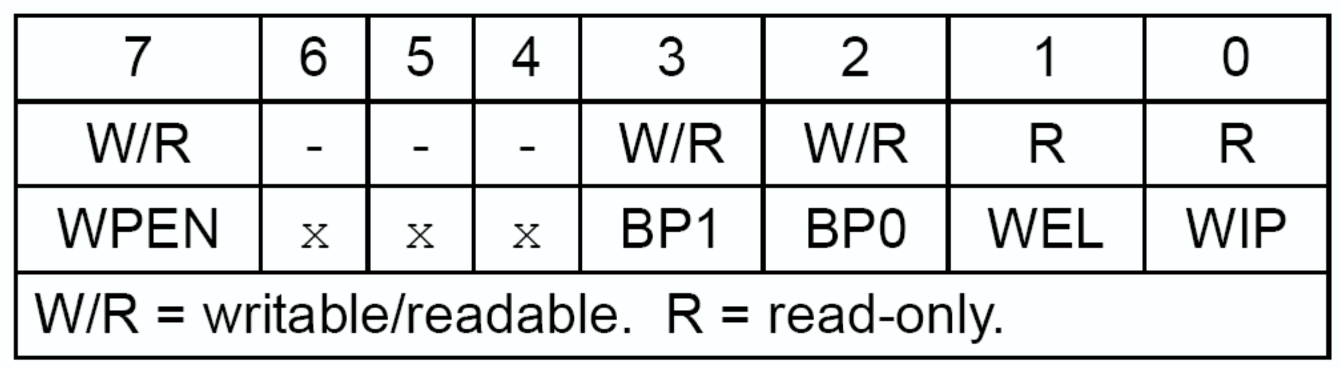
Internal Structure of the 25LC256L



BIRMINGHAM City University

The STATUS REGISTER

Status Register



7	6	5	4	3	2	1	0
W/R	-	-	-	W/R	W/R	R	R
WPEN	x	x	x	BP1	BP0	WEL	WIP
W/R = writable/readable. R = read-only.							

The **Write-In-Process (WIP)** bit indicates whether the 25XX256 is busy with a write operation.

The **Block Protection (BP0 and BP1)** bits indicate which blocks are currently write-protected.

The **Write Enable Latch (WEL)** bit indicates the status of the write enable latch and is read-only.

The **Write-In-Process (WIP)** bit indicates whether the 25XX256 is busy with a write operation.

READ STATUS REGISTER sequence

Example command sequence:

The WIP bit in the status register must be checked prior to writing a byte to ensure a write cycle is not in progress.

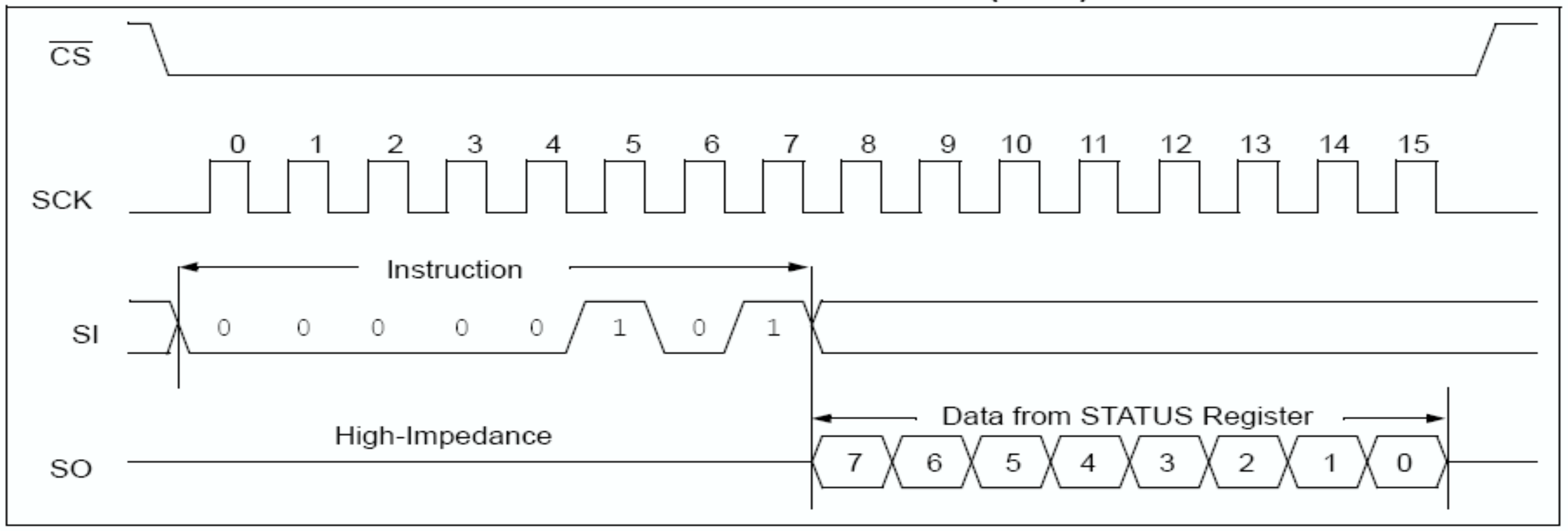
The status register is read by issuing the RDSR command (0x05) – we must wait until WIP is 0 (no Write In Progress) before starting a byte write operation.

CS => LOW enabling the device
Issue RDSR command (0x05)
Read SSPBUF and test WIP (bit 0)
CS => HIGH [this is optional]

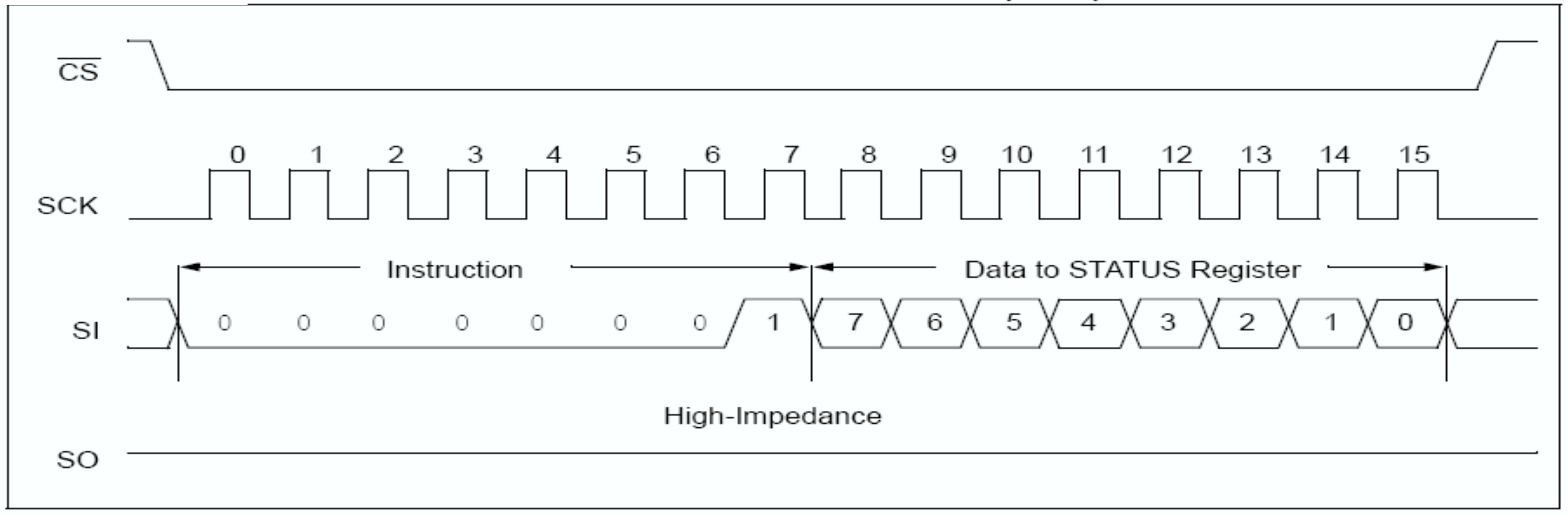
BIRMINGHAM City University

STATUS REGISTER sequences

READ STATUS REGISTER TIMING SEQUENCE (RDSR)



WRITE STATUS REGISTER TIMING SEQUENCE (WRSR)

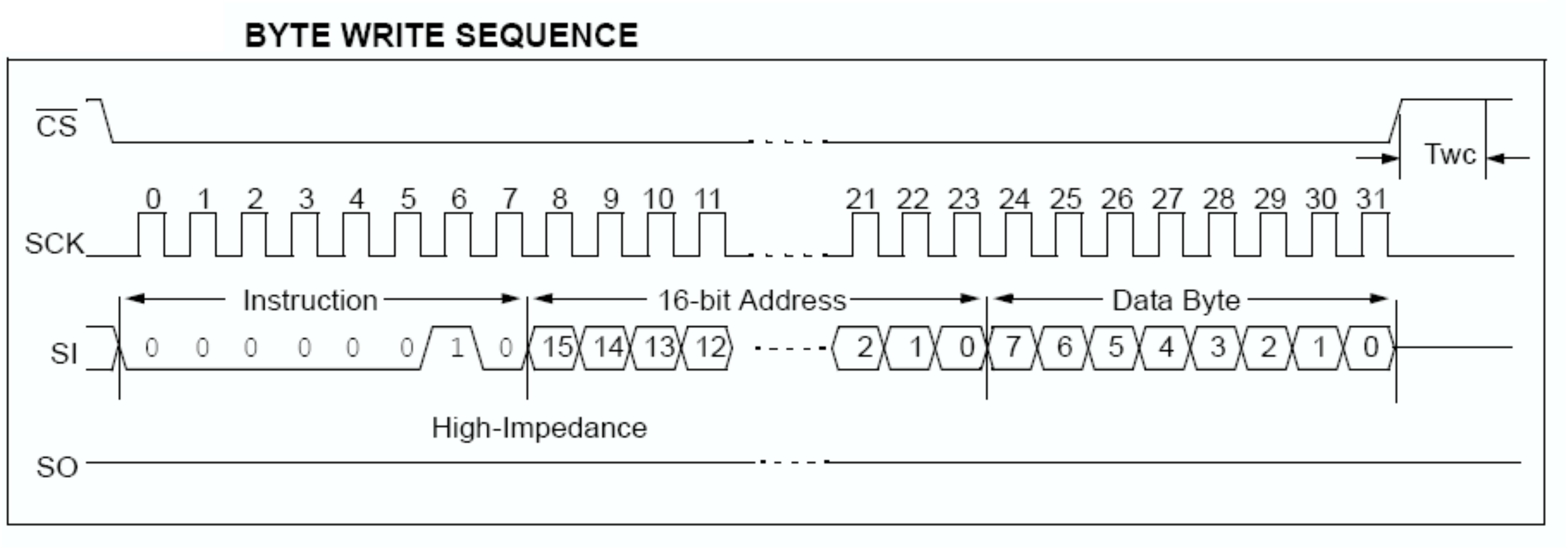


Byte Write sequence

- CS ➔ LOW enabling the device
- Set the Write Enable latch :
  - Issue the WREN command (0x06).
  - CS ➔ HIGH sets the WREN latch.
- CS ➔ LOW re-enabling device in write mode
- Issue WRITE command (0x02)
- Issue 16 bit address
- Issue data byte
- CS ➔ HIGH to indicate byte write sequence is complete, and the write cycle should now start.

The write cycle takes ~5ms.

Byte write sequence



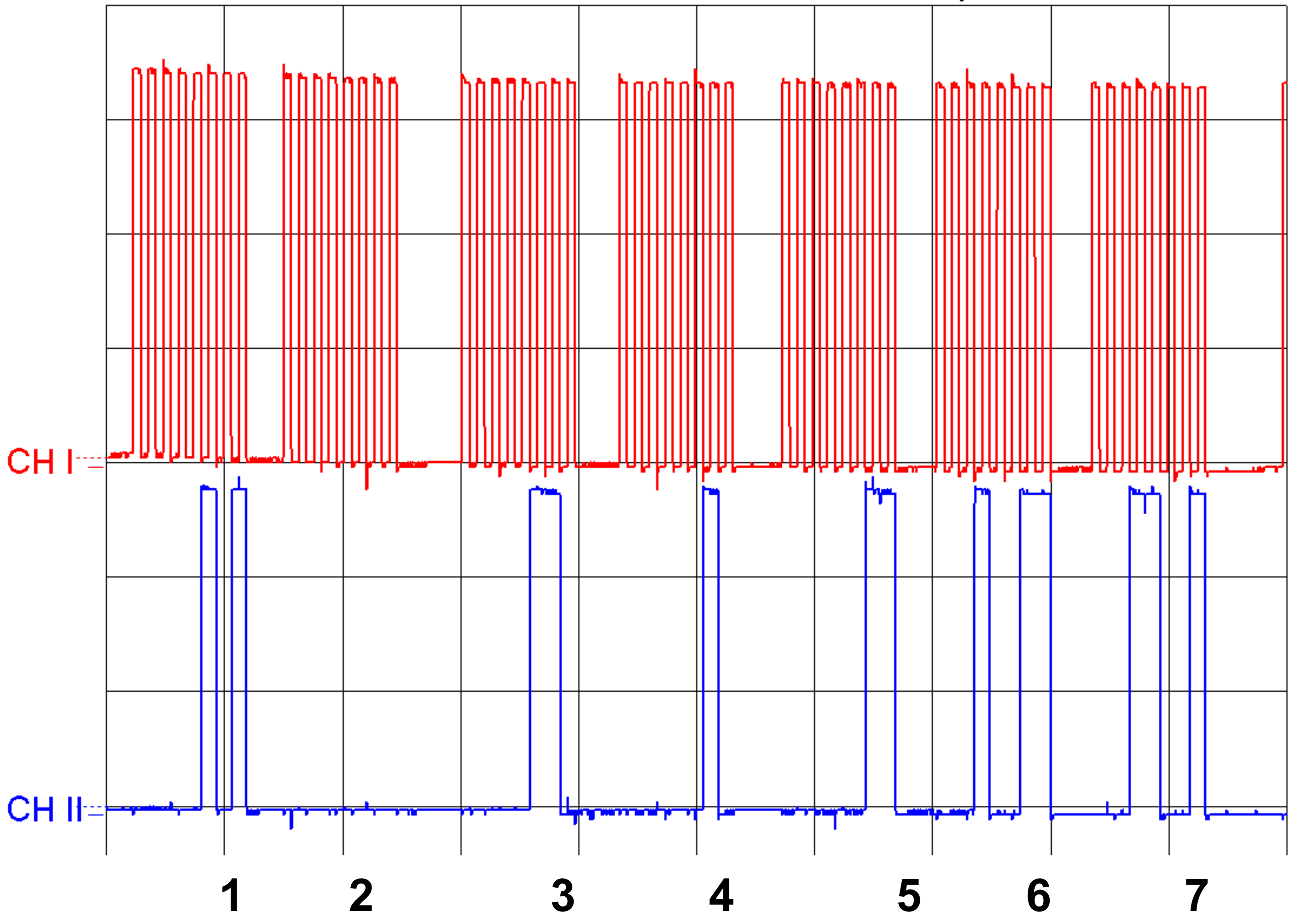
Writing data one byte-write sequence at a time, with the associated 5ms write-cycle delay, would not be particularly efficient for data acquisition purposes, as we are normally dealing with streamed data.

The Page Write sequence helps to overcome this.

Byte Write sequence

Measure time: 11:34:31
Measure date: 21/07/2009

CH1: 1.147V/DIV DC CH2: 1.353V/DIV DC TB A: 20 μs TR: EXT-NR



1. The STATUS instruction 0x05 is sent
2. The second set of clock pulses is the read of the STATUS register (val=0).
3. WREN instruction 0x06 is sent
4. WRITE instruction 0x02 is sent
5. MS Byte of the 15 bit address is sent
6. LS Byte the 15 bit address is sent
7. Data byte sent

```
CH I :  Cursor I: .138V        Cursor II: 3.854V   Diff. I-II: -3.7165V
CH II:  Cursor I: .054V        Cursor II: .000V    Diff. I-II: .0541V
dt: 204.500 µs        1/dt: 4.890 kHz
```

Page Write sequence

A PAGE is 64 bytes – up to one page can be written to the SPI eeprom before a write cycle must be initiated ………i.e. before CS ➔ HIGH

This is useful as the ~5ms write cycle time will only occur once, after the 64th byte is written.

A page write is achieved by continuing to issue data bytes after the first one described above while keeping the CS’ pin low.

The page write is efficient because:

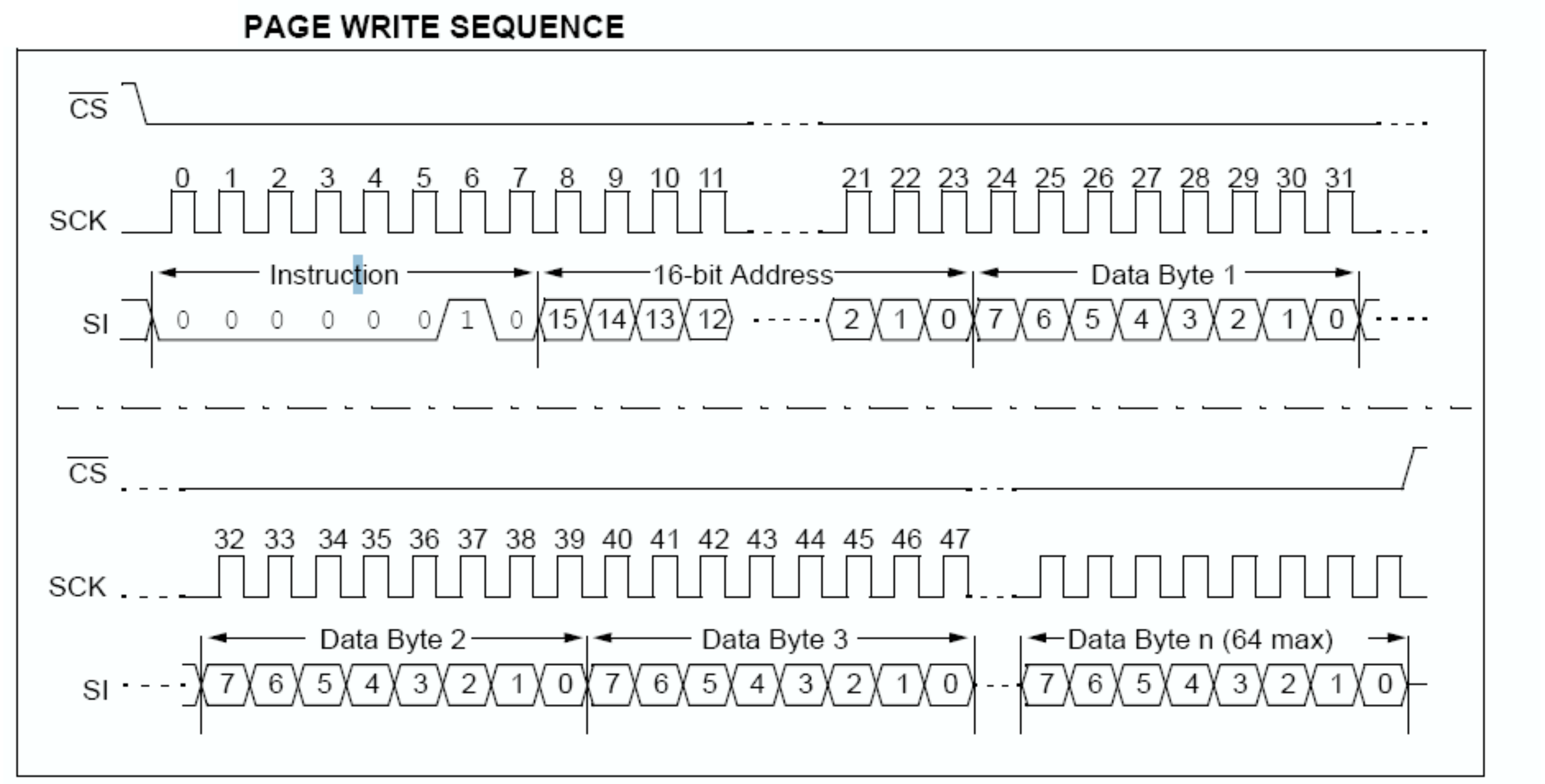
Only one address required for 1-64 bytes – the internal address is automatically incremented

Only one WRITE command required for 1-64 bytes

CS remains low for the entire sequence.

BIRMINGHAM City University

Page Write sequence

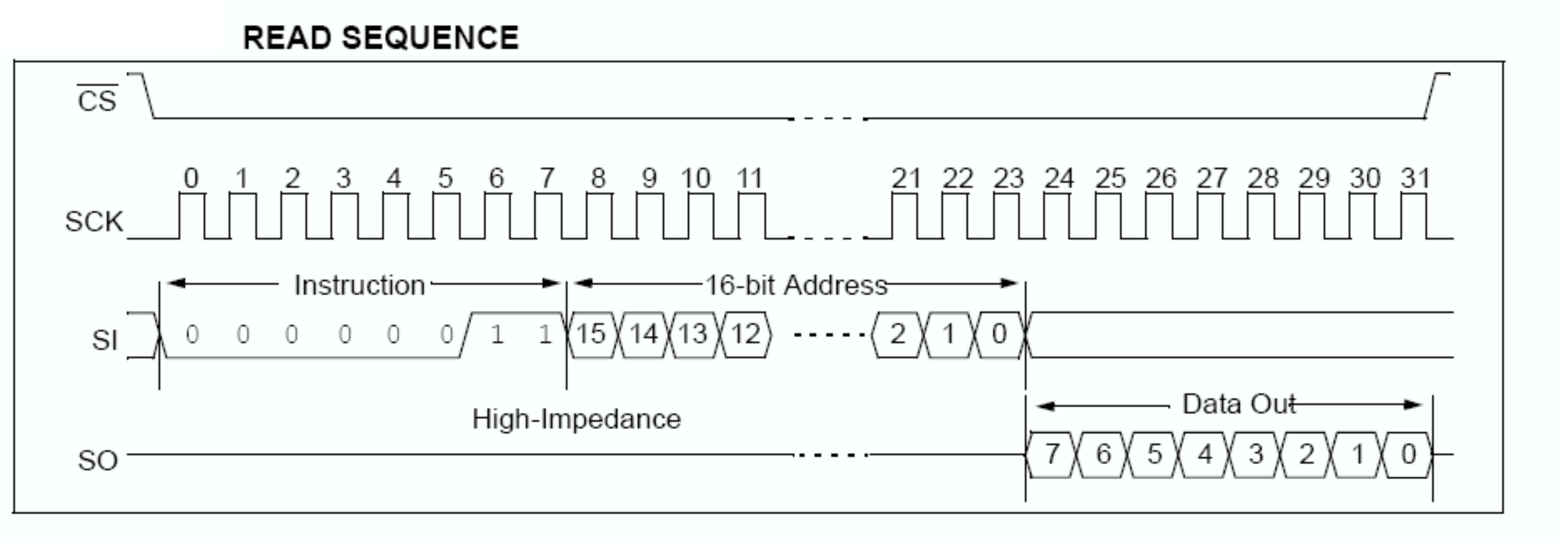


Byte read sequence

- The read sequence is performed in a similar way
- The RDSR register is checked by issuing the status command (0x05).
- After the RDSR is checked the CS’ pin must be pulled low enabling the chip.
- The read instruction (0x03) is sent followed by a 16 bit address which indicates where to read from.
- On the next set of clock pulses the data stored at the chosen address is clocked out of the shift register.
- The CS’ pin is pulled high to indicate the read is complete.
- A sequential read can be achieved by continuing to issue clock pulses after the first described above. As long as CS’ is kept low the EEPROM will auto increment it’s address pointer each set of pulses and a byte will be shifted out. Sequential reads will loop back to 0x00 when the address overflows.

BIRMINGHAM City University

Byte read sequence



Any number of bytes can be read back sequentially from the specified address – there is no page limit for READ. The internal address is auto-incremented on each read.

Non-sequential addressing would require a new read sequence for each address.

BIRMINGHAM City University

Speed of SPI EEPROM data transfer

The maximum clock rate for the Microchip 25aa256 is 10MHz.

A one-byte transfer consists of command, address, data = 4 bytes = 32bits.

This gives a nominal rate of 3.2us for 1 byte of data or 312.5 KBytes/second.

However, the 5ms write cycle would reduce this to just under 200 Bytes/sec!

The Page Write lets us write up to 64 bytes before a write cycle is required.

i.e. a page frame would be 64 + command + address = 67 bytes = 536 bits

➔ 53.6 us, which gives 5.054ms for 64 bytes = 12.7KBytes/sec.

This betters RS232 at 115200 baud ….. but it is burst-mode data transfer rather than continuous. This makes it appropriate for data sampling at greater than 5ms intervals …..

SPI FRAM: Ferromagnetic Random Access Memory

FM25256B : *256Kb FRAM Serial 5V Memory*

256K bit Ferroelectric Nonvolatile RAM

- Organized as 32,768 x 8 bits
- Virtually Unlimited Endurance (10^{14} Cycles)
- 10 Year Data Retention
- NoDelay™ Writes
- Advanced High-Reliability Ferroelectric Process

Very Fast Serial Peripheral Interface - SPI

- Up to 20 MHz Frequency
- Direct Hardware Replacement for EEPROM
- SPI Mode 0 & 3 (CPOL, CPHA=0,0 & 1,1)

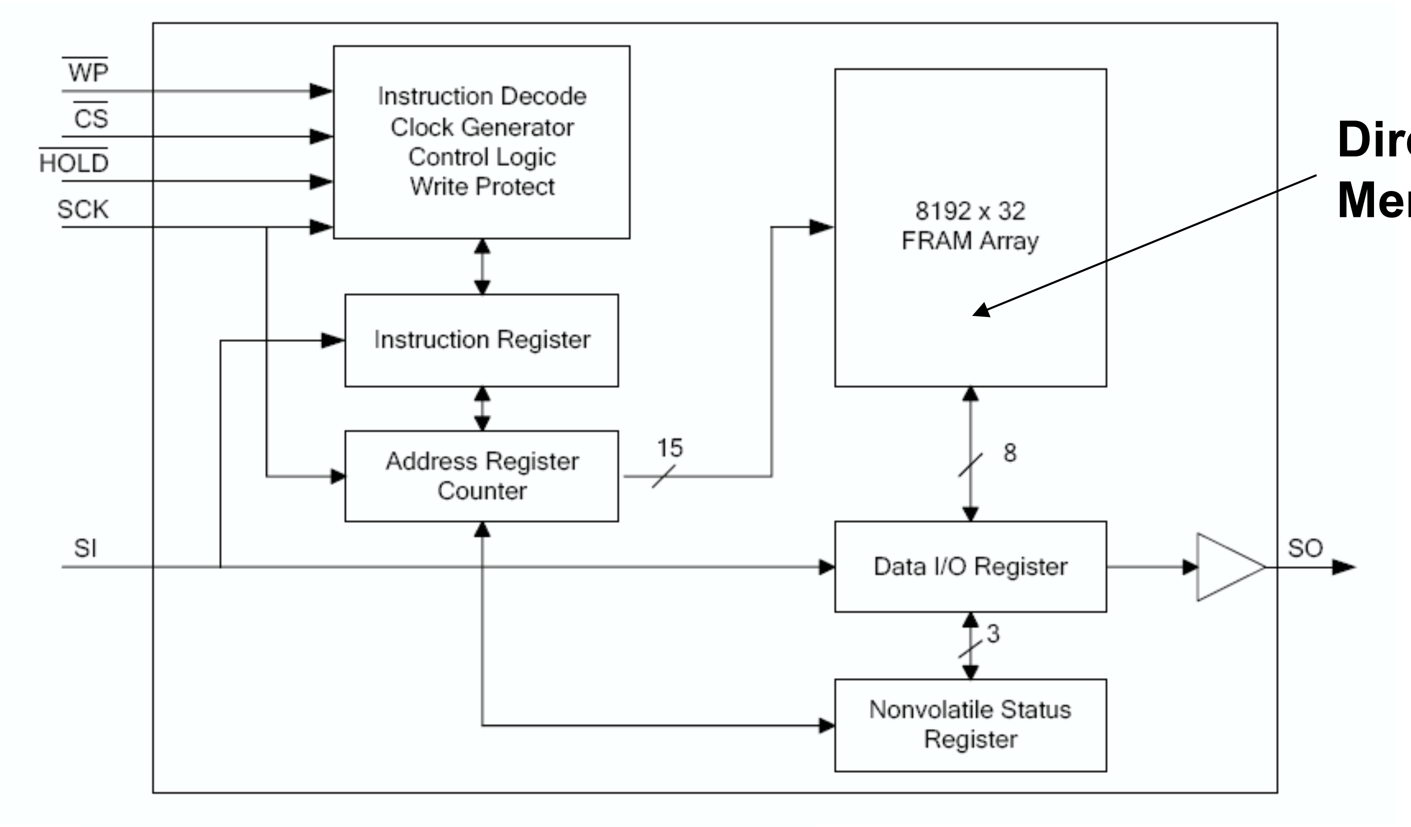
Write Protection Scheme

- Hardware Protection
- Software Protection

Wide Operating Range

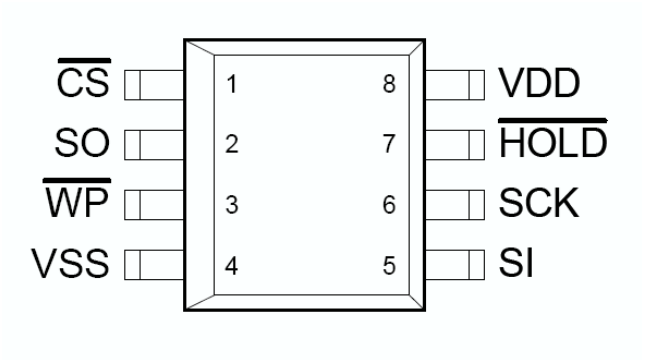
- Wide Voltage Operation 4.0V – 5.5V

FRAM Internal Structure



Direct write to Memory

Pin Name	Function
/CS	Chip Select
/WP	Write Protect
/HOLD	Hold
SCK	Serial Clock
SI	Serial Data Input
SO	Serial Data Output
VDD	Supply Voltage (4.0 to 5.5V)
VSS	Ground



Comparison of FRAM vs. EEPROM

Unlike serial EEPROMs, the FM25256B performs write operations at bus speed - i.e. No write delays are incurred. The next bus cycle may commence immediately without the need for data polling.

This means no 64-byte page, and no 5ms Write-cycle ……!!

The FM25256B is also twice as fast as the 25AA256 … running at 20MHz, thereby doubling the burst data rate of the EEPROM and also maintaining this rate continuously.

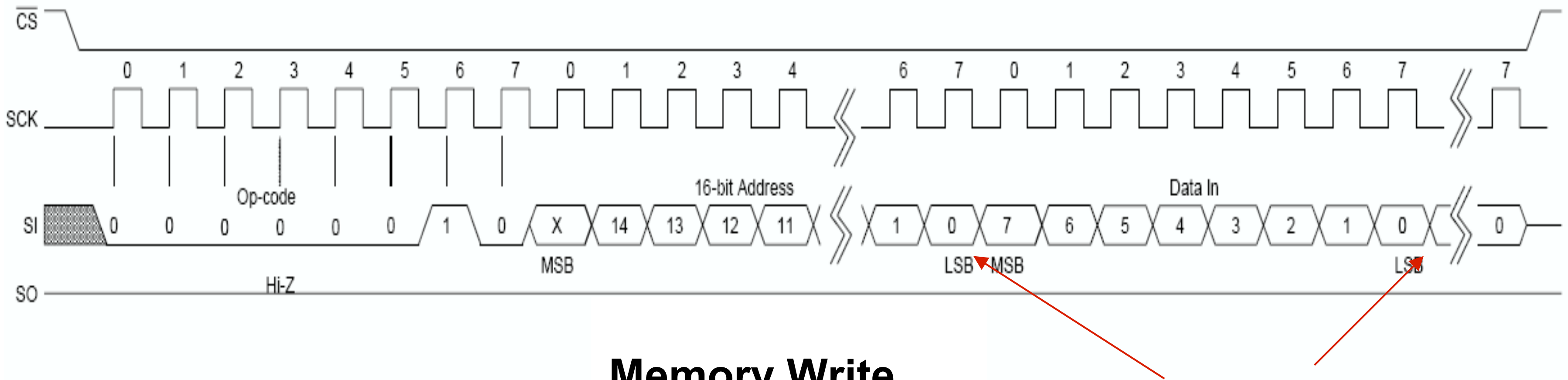
For sequential data acquisition, 32Kbytes can be written at full speed, with no delays.

This gives us a maximum data rate of 2.5MBytes/sec.

The FRAM is pin compatible AND code compatible with serial EEPROM

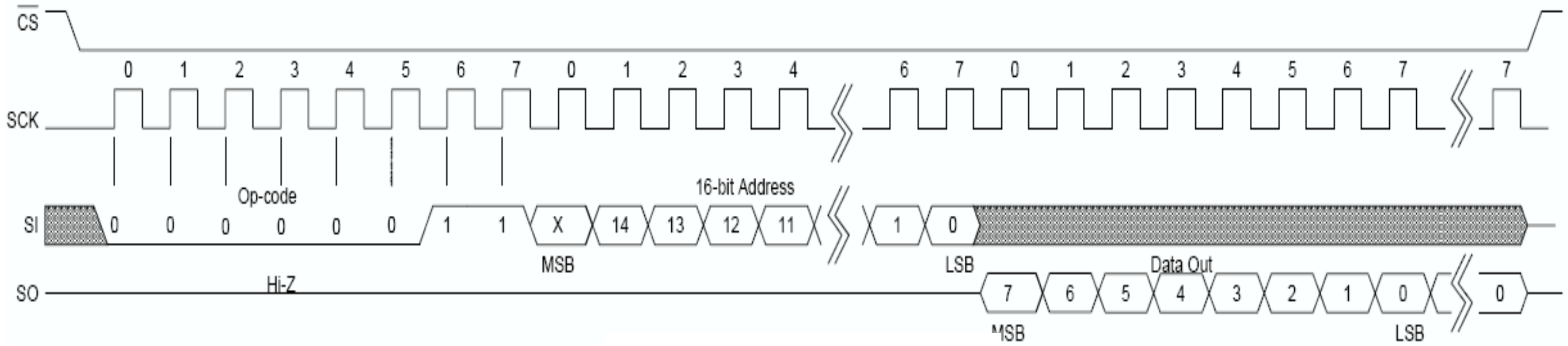
(Disadvantage?: a price factor of 3X ….)

FRAM Write and Read Cycles



Memory Write

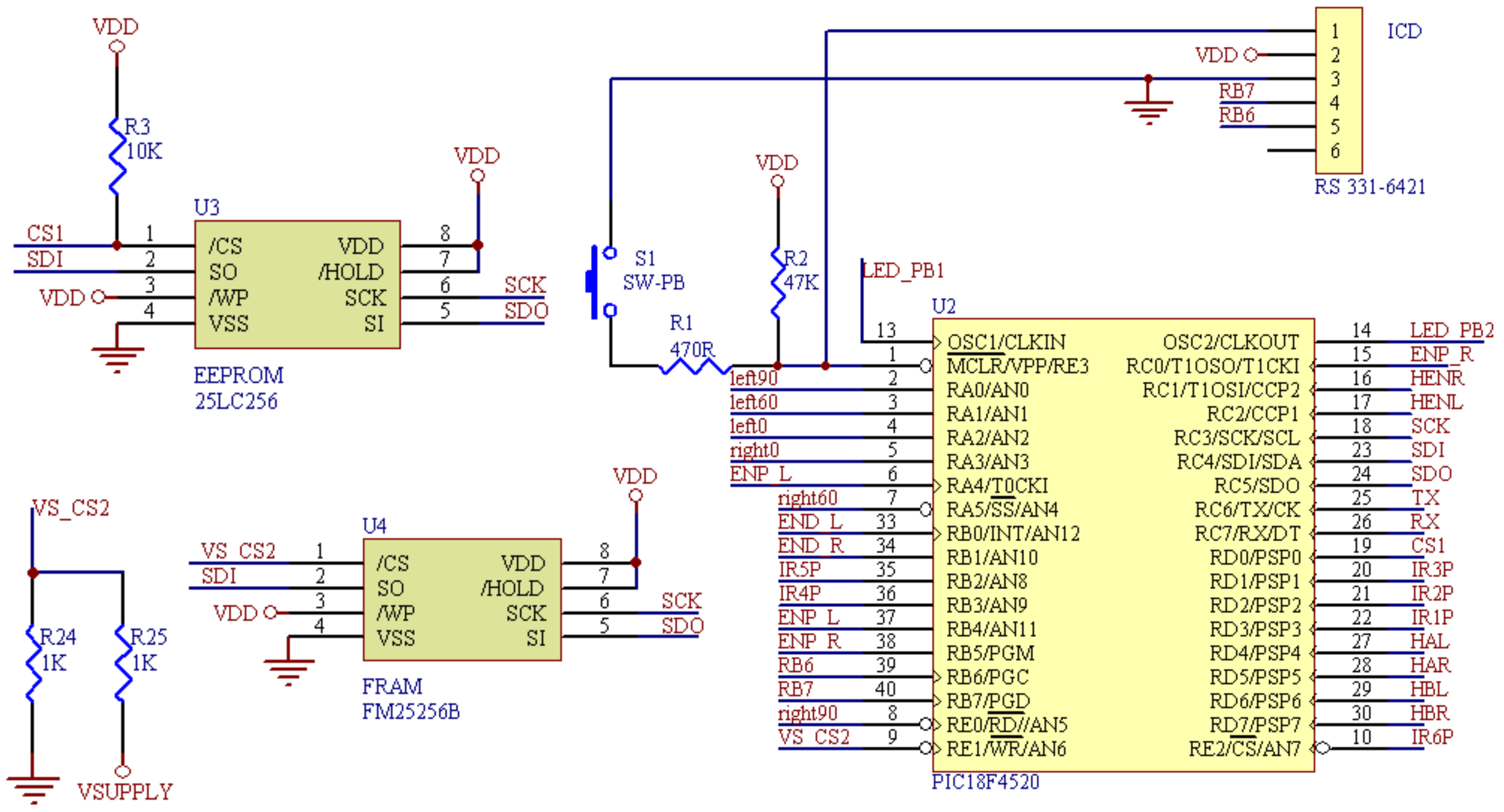
No Delay, No Write Cycle



Memory Read

BIRMINGHAM City University

MyTEE-mouse SPI Memory



Summary

FM25256B SPI FRAM out-performs 25LC256 SPI EEPROM ….

It can be written to at 20MHz vs the 10MHz of the 25LC256
It has virtually unlimited read/write cycles (>10^{14})
It does not require a 5ms write cycle to complete the write operation

25LC256 SPI EEPROM out-performs FM25256B SPI FRAM …

It has a data retention of > 200 years vs. >10 years for the FM25256B

BIRMINGHAM City University

Using SPI memory with C

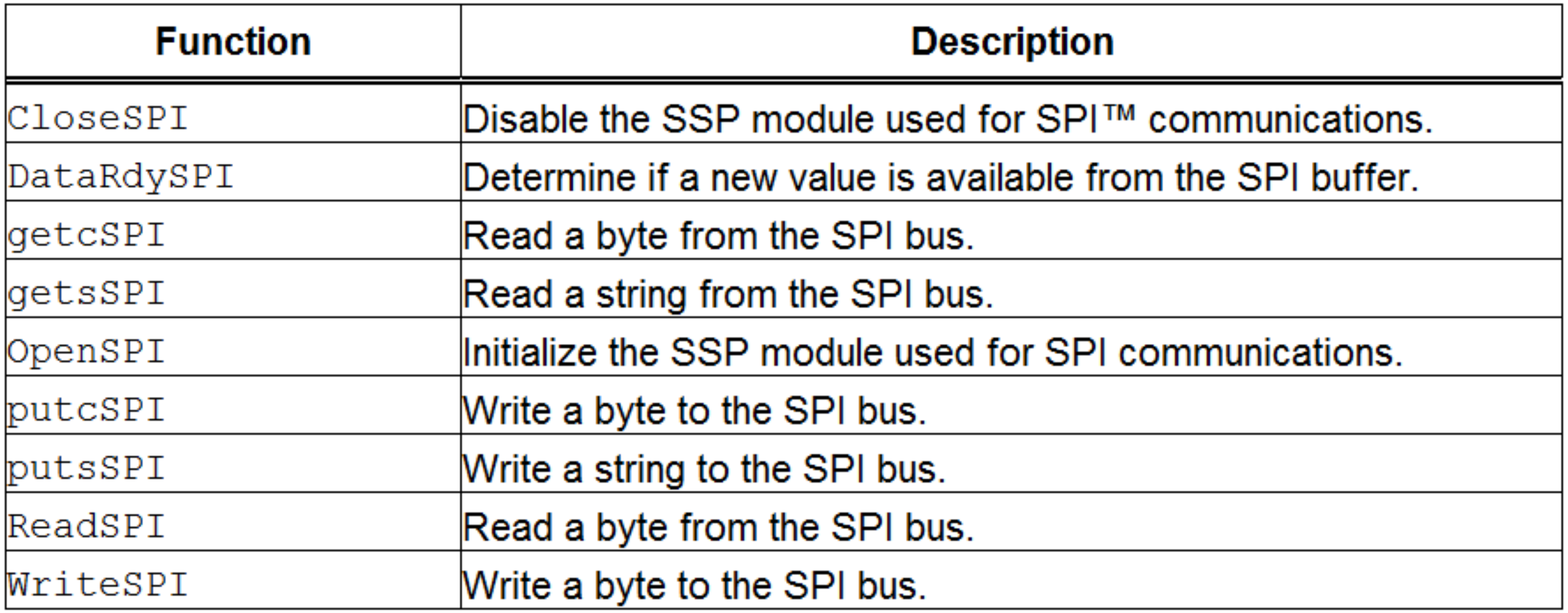
There are several layers of software required to use the FRAM for data acquisition and retrieval.

1. Low-level SPI functions - provided by MCC18 peripheral libraries
2. High-level SPI functions specific to FRAM - written by the user
3. Application-specific functions - written by the user

BIRMINGHAM City University

SPI Functions provided by MCC18: Level 1

MCC18 provides functions for the initialisation of the SPI interface, and writing or reading through SPI.



Function	Description
`CloseSPI`	Disable the SSP module used for SPI™ communications.
`DataRdySPI`	Determine if a new value is available from the SPI buffer.
`getcSPI`	Read a byte from the SPI bus.
`getsSPI`	Read a string from the SPI bus.
`OpenSPI`	Initialize the SSP module used for SPI communications.
`putcSPI`	Write a byte to the SPI bus.
`putsSPI`	Write a string to the SPI bus.
`ReadSPI`	Read a byte from the SPI bus.
`WriteSPI`	Write a byte to the SPI bus.

BIRMINGHAM City University

Functions for FRAM on MyTEEmouse: Level 2

FRAM specific functions must be written by the user – the following functions are provided in FramSPI.INC.

FR_Init	: Initialise the MSSP for SPI
FR_WriteEnable	: Enable the FRAM for writing data
FR_WriteDisable	: Disable the FRAM for writing data
FR_ByteWrite	: Write a single byte to FRAM
FR_ByteRead	: Read a single byte from FRAM
FR_BlockWrite	: Write a number of bytes from PIC memory to FRAM
FR_BlockRead	: Read a number of bytes from FRAM to PIC memory

FRAM Application-specific functions: Level 3

Level 3 functions are for:

- Transfer of a number of mixed data types to and from FRAM
- Upload of the record to the PC via the RS232 port.

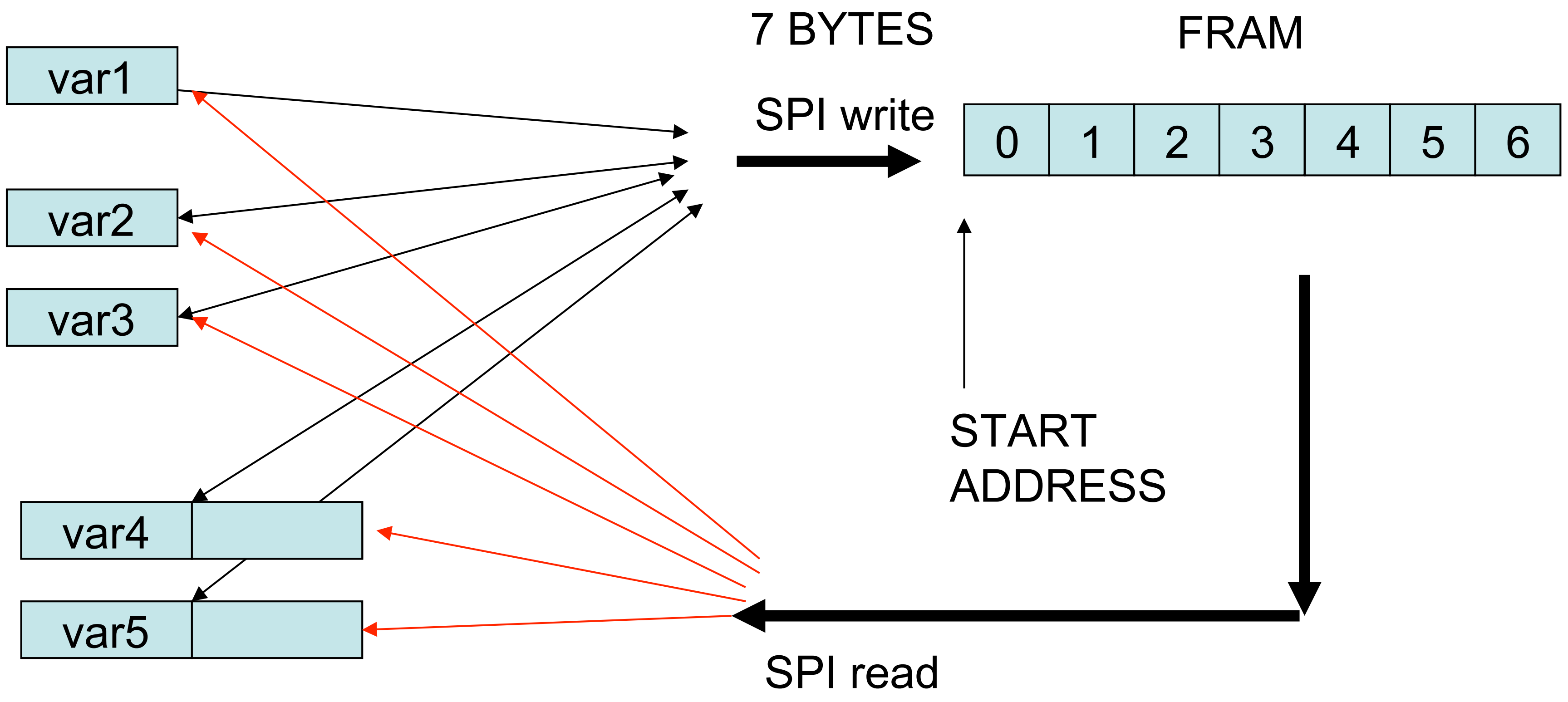
Issues

FRAM is 8-bit memory

Data must be written one byte at a time

Writing/reading mixed data types (char, int, long etc.) can be messy

Streaming bytes and maintaining data type



We have to maintain the data type information that is held only *by association* once the data has been written in byte format.

Streaming bytes and maintaining data type

We could have a number of functions that each write a specific data type

SPIWRchar() , SPIWRint(), SPIWRlong(), SPIWRfloat() and so on

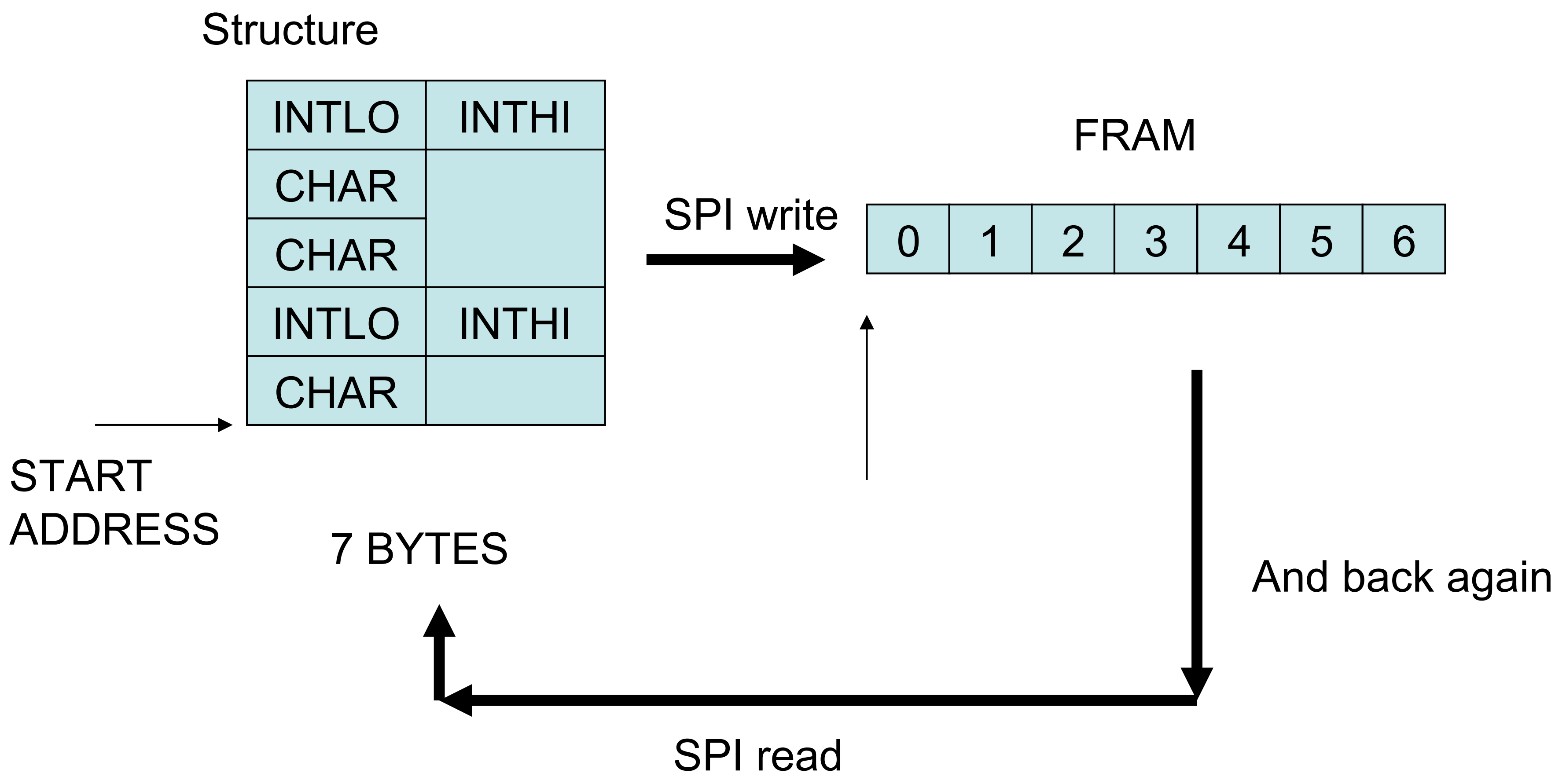
OR

Use a **STRUCTURE** to define the variable names and types

Transfer this as a block of data bytes - this we will call a **RECORD**

Read the record back into the same structure - this 'rebuilds' the type information.

Streaming bytes and maintaining data type



Streaming bytes from/to a structure maintains the data type information ….

BIRMINGHAM City University

Using Structures

Structures and Arrays are closely related –

An array is a collection of variables of the same type referenced under one name. Arrays are used to keep related information in the same place.

A structure is a collection of variables of different types referenced under one name. Structures are used to keep related information in the same place.

```
volatile struct {
        signed int leftpos;
        signed int rightpos;
        signed int v_current;
} record;
```

If record.leftpos = 0x1234, record.rightpos = 0x1245, record.v_current = 5

The compiler will create the following in memory, starting at &record:

34 12 45 12 00 05   ….. Hexadecimal low-high byte format

i.e. it can be considered as an array of byte values.

BIRMINGHAM City University

Using Structures

```
volatile struct {
        signed int leftpos;
        signed int rightpos;
        signed int v_current;
} record;
```

We access the start of the structure using: **&record**

& is a pointer operator returning the address of the start of the structure

We can determine the length using: **recordsize = sizeof(record)**

And so we have the start address and length of a byte array.

Continuous data acquisition

- At each sample time, a RECORD of data is written to FRAM, using a counted loop (start address and length known).
- A record counter is incremented ➔ total number of records
- A memory counter is incremented ➔ total amount of the data
- When the FRAM is 'full' no more data is written (could use wrap-around memory)

The files that will allow you to implement this are:

FramSPI.inc	: Level 2
FramRecordRW.inc	: Level 3 (see next slide)

Level 3 functions

```
void InitRecord()

        initialise the SPI connection to FRAM
        initialise variables

void WriteRecord()
        copy program variables to structure variables
        write the structure to FRAM as a record
        increment pointers and store no. of records

void UpLoad ()
        for all records
                read record to structure
                export structure variables to COM port using printf()
```

Controlling the upload

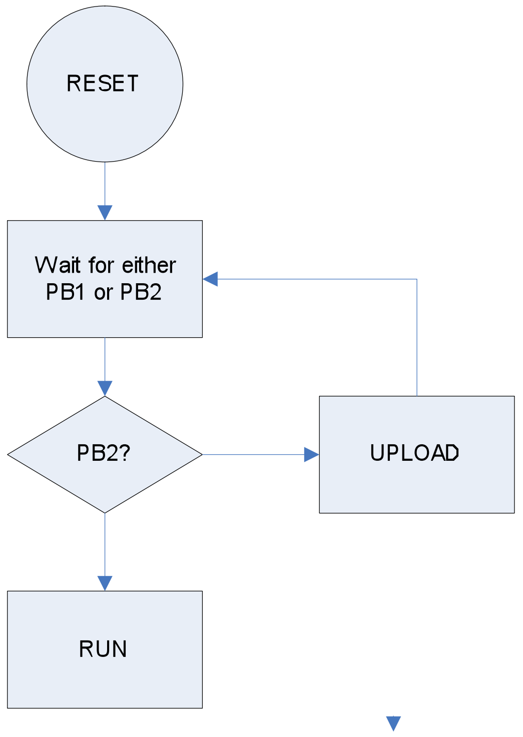
We need some way to control when upload to the PC happens.

MyTEEmouse has push-buttons ……

wait for PB1 or PB2

If PB2 do upload

else run main program



Level 2 and 3 source code

C source code will be available for download via the micromousonline web site.

DEMO