# Software Parts & Peripherals

## Callable Software functions extracted from the ECH

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## Introduction

These functions have been extracted from the Embedded Control Handbook, and modified into callable functions, with parameters to customize their actual use. While these functions originated from the referenced app note, they have been modified to meet several goals:

1. Use standard symbols as defined in 16C5x.inc
2. Flatten out to call at most one level of sub function.
3. Parameterize to customize the operation of the function.
4. rename function to more clearly reflect the function.

Cautions when using the libraries:

1. Normal paging concerns apply. See AN581 for details on implementing long calls.
2. ORG instructions should be used to prevent functions from running across page boundaries.
3. Internal symbols are prefixed with underscores to reduce duplicate symbol names.
4. Part must be specified prior to including these functions

Cost of each function
Each function includes a table that shows the cost of using a function in terms of Program space (EEPROM), Clock cycles, file registers, Stack levels (including the call to the function) and IO pins.

Tools Requirements:
MPASM V2.00 (or later)
MPLINK V1.00 (or later)

# Software Parts & Peripherals

## Callable Software functions extracted from the ECH

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## Serial Communication UART

The SWUART12 functions provide functions to send and receive data on the UART port. When the lines are connected up to a voltage level converter such as the MAX232, these routines provide RS232 communication capability. Normal RS232 communication uses 8 data bits and 1 stop bit, sent LSB first.

The library uses the generic pic16c5x designation. This allows flexibility within the 12 bit family. However it the routine needs to communicate on PORTC, the designation must be changed to p16c55 or p16c57 as appropriate. Otherwise the assembler will flag an error on all references to PORTC.

<PIC/MAX232 circuit diagram>

This library must be reassembled to change any of the communication parameters:
CLOCKSPEED (4, 8, 10, or 20 MHz)
BAUDRATE (2400, 4800 or 9600)
TX_PIN
RX_PIN
UART_PORT
MODE (Data sent MSB first or LSB first)
STOP_BITS (1 or 2)
NBITS (number of data bits, 7 or 8)

These are #defines at the top of the SWUART12.ASM file.
UART_PORT must resolve to 5, 6 or 7 (PortA, PortB, or PortC)
TX_PIN and RX_PIN must resolve to 0-7 inclusive

Currently the following combinations of CLOCKSPEED and BAUDRATE are supported:

| | 2400 | 4800 | 9600 |
|---|---|---|---|
| 4 MHz | OK | OK | OK |
| 8 MHz | Not Supported | OK | OK |
| 10 MHz | Not Supported | OK | OK |
| 20 MHz | Not Supported | Not Supported | OK |

For reliable communications, an external crystal is recommended for accurate timing,

Library Size

| **EEPROM** | **Clock cycles** | **File Registers** | **Stack levels** | **IO Pins** |
|---|---|---|---|---|
| $51_{10}$ | * | 4 | 1 | 2 |

* Depends on Clockspeed and Baudrate.

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## OpenUART

Function Opens a port for serial transmission (sets the line high and TRISs the port).

Include File swuart12.inc

Options see above

Syntax call OpenUART

Remarks OpenUART build a TRIS value for the port such that the RX Pin is an input, the TX Pin is an output and the rest of the bits are set by the TRIS value.

Return Value None

Limitations RX Pin and TX Pin must be on the same port.

Error Conditions Baudrate/Clockspeed combination is checked at assembly time. If the combination is not supported, an error message is generated.

See Also AN510

Example:

```
include <swuart12.inc>
…
call    OpenUART            ; set up the UART Port.
…
movlw   ‘A’
call    putcUART            ; Transmit char
…
```

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## getcUART

Function Reads a byte from the software UART.

Include File swuart12.inc

Options See above

Syntax call getcUART

Remarks

- Receiving port must be opened prior to calling function. (See OpenUART)
- If a startbit is detected, getcUART keeps control of processor until character is received.
- Delays are handled as software loops. The delay constants are setup automatically as a function of baud rate and clock speed. At normal operating temperatures the RC oscillator is sufficiently accurate for 2400 baud, however, for faster communication speeds or over a temperature range, a crystal oscillator is required.

Return Value Returns 0 in the Carry bit (Status,C) if nothing was found on the port. If a character was detected, a 1 is returned in the Carry bit and the returned character is returned in RcvReg.

Limitations Parity not supported

Error Conditions None

See Also AN510

Example:

```
include <swuart12.inc>
…
call    getcUART            ; receive
btfss   STATUS,C            ; was a byte received?
Goto    no_char             ; no…
movf    RcvReg,w            ; yes - process character

…
```

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## putcUART

Function Sends a byte out the software UART.

Include File swuart12.inc

Options see above

Syntax load byte to transmit into W register,
call putcUART

Remarks

- Port must be opened prior to calling function. (See OpenUART)
- putcUART keeps control of processor until character is sent.
- Due to the read-modify-write nature of the BSF and BCF instructions, the remaining bits on the port may have changed during the course of the transmission. For more information, see paragraph 5.6.1 “Bi-Directional Data Port” of the 16c5x and enhanced 16c5x datasheets. (Paragraph 5.4.1 of the 12c50x datasheet).
- Delays are handled as software loops. The delay constants are setup automatically as a function of baud rate and clock speed. At normal operating temperatures the RC oscillator is sufficiently accurate for 2400 baud, however, for faster communication speeds or over a temperature range, a crystal oscillator is required.

Return Value None

Limitations Parity bit is not supported

Error Conditions None

See Also AN510

Example:

```
include <swuart12.inc>
…
movlw   ‘A’             ; load character into Wreg
call    putcUART        ; transmit
…
```

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

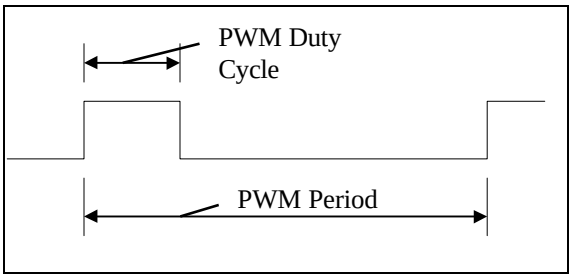
## PWM (Pulse Width Modulation)

These functions set up a state machine that will toggle an output pin. The amount of time the pin is high is proportional to the ratio of the **dutycycle** to the **period** Both of these parameters should be set prior to initiating the PWM. Period is an 8 bit integer value 0-255. Duty cycle is also an 8 bit integer value 0-Period. If dutycycle is greater than or equal to the period, the output will remain high 100% of the time.

This PWM implementation depends getting called from the main program at a regular interval. Each waveform consists of a number of timeslices. Total time for each PWM waveform is given by the number of timeslices multiplied by the size of each timeslice. For example, if we had 32 timeslices and each timeslice is 50uS, each waveform is 1600uS = 625Hz. PWMperiod specifies the number of timeslices that make up a full wave.

For better resolution, increase the number of time slices in a waveform (increase PWMPeriod). However as the PWMperiod increases, frequency decreases (assuming the timeslice remains constant). To keep frequency constant and increase resolution, we must shrink the timeslices by calling the PWM Function more often. This can be done by calling the function more often, or by running at a higher clock rate.

The duty cycle tells how many timeslices out of the whole will be high. The remaining slices will be low. If the DutyCycle says 6 timeslices are high and the period is defined as 32 timeslices, the wave will be high for 6 out of every 32 timeslices for an 18.75% duty cycle. Working backwards, given that the controlling program wants a 41% duty cycle, 41% of 32 = 13.12.




At the maximum rated speed of the by12 parts (20 MHz), Servicing each timeslice costs about 10 Cycles, which means these functions can handle approximately 500,000 timeslices per second. At maximum resolution (Period = 256) frequency works out to 3.9KHz. However if lower resolution is acceptable (period = 32) frequency comes up to 31Khz. Balancing the tradeoffs is left as an exercise to the engineer.

Configuring the library for your application:

1. Select the PORT and PIN to use (PWMPIN).
2. Select the TRIS value for the rest of the port. Since the TRIS register must be assigned as a whole, the OpenSWPWM function needs to know how to set the rest of the bits. The bits for the PWMpin will be set as an output.

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## OpenSWPWM

Function Macro to define the PWM Port and pin, and TRIS the port register for output on the specified pin.

Include File SWPWM12.INC

Options none

Syntax OpenSWPWM
PWMport - name of the output port to use
PWMpin - identifies the pin within the output port.
TRIS - Since the by12 family has no provision for setting individual TRIS bits, this parameters defines the remaining bits in the register.

Remarks

Return Value None

Limitations None

Error Conditions
PWM_PORT must evaluate in the range of 5-7 (PortA, PortB or PortC)
PWM_PIN must be in the range 0-7.
Generates an assembly warning if either of these are not true.

Size

| EEPROM | Clock cycles | File Registers | Stack Levels | IO Pins |
|---|---|---|---|---|
| 2 | 2 | 0 | 0 | 1 |

See Also AN538

Example:

```
include <SWPWM12.inc>
…
…
OpenSWPWM
…
…
```

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## SWPWM12

Function Advances the state of the PWM state machine.

Include File SWPWM.INC

Options PWM_PORT, PWM_PIN. Library must be reassembled to change.

Syntax call SWPWM12

Remarks

- The calling program must call PWMby12 at regular intervals. The PWM period consists of PERIOD calls to SWPWM12, which is 256 times the calling frequency. For example, if SWPWM12 is called every 10uS, the PWM period = 256 * 10uS = 2560 uS = 2.56mS. PWM Frequency is the inverse of the period. In this case, F= 390 Hz
- Due to the read-modify-write nature of the BSF and BCF instructions, the remaining bits on the port may have changed during the course of the transmission. For more information, see paragraph 5.6.1 “Bi-Directional Data Port” of the 16c5x and enhanced 16c5x datasheets. (Paragraph 5.4.1 of the 12c50x datasheet).

Return Value None

Limitations Maximum PWM frequency (1/Period) = 3Khz (20 Mhz clock, no delay between calls, 13 clocks per cycle).

Error Conditions None

Size

| | EEPROM | Clock cycles | File Registers | Stack levels | IO Pins |
|---|---|---|---|---|---|
| **Normal** | 9 | 9 | 2 | 1 | 1 |

See Also AN538

Example:

```
        include <SWPWM12.inc>
        …
        OpenPWMPort   PORTB, 1, 0xFF          ;  PortB.1 output pin, rest of port B are inputs
        movlw   0x40                          ; set up 25% duty cycle
        SetPWMDutyCycle
pwmloop
        call    PWMby12Normal
        movlw   0xfe                  ; set up 10uS delay
        movwf   temp
        incfsz  temp,f
        goto    $-1
        goto    pwmloop
        …
```

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## SetPeriodSWPWM

Function Saves the contents of the W register to the PWM Period register.

Include File SWPWM12.inc

Options none

Syntax
```
movlw   0x80
SetPeriodSWPWM
```

Remarks The period determines the number of calls that make up one PWM waveform. Total time for the period is the number of calls multiplied by the time interval between the calls. Fewer calls allow a higher frequency, but at a lower resolution. More calls increase the resolution but decrease the maximum frequency.

Return Value None

Limitations Maximum Period value is 0xFF (255). If Duty Cycle > Period, output will be 100% high.

Error Conditions None

Size

| EEPROM | Clock cycles | File Registers | Stack Levels | IO Pins |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |

See Also AN538

Example:
```
include <SWPWM12.inc>
…
movlw   0x80            ; need 7 bit resolution
SetPeriodSWPWM          ;
…
…
```

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## SetDutyCycleSWPWM

Function Macro to save the contents of the W register to the PWM Duty Cycle.

Include File SWPWM12.inc

Options none

Syntax
```
movlw   0x80
SetPeriodSWPWM              ; set up a 7 bit resolution
movlw   0x40            ; with a 50% duty cycle.
SetDutyCycleSWPWM
```

Remarks Duty cycle defines how many of the PWM intervals defined by Period will be high.

The value programmed should be the DutyCycle divided by the Period.

Return Value None

Limitations Maximum Duty Cycle value is 0xFF (255). If Duty Cycle > Period, output will be 100% high.

Error Conditions None

Size

| EEPROM | Clock cycles | File Registers | Stack levels | IO Pins |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |

See Also AN538

Example:
```
include <SWPWM12.inc>
…
movlw   0x40                ; need at 25% duty cycle
SetDutyCycleSWPWM           ;
…
…
```

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## Software I2C

The by12 software I2C functions provide single master control of an I2C bus. While directly aimed to support communication between the 12 bit devices and the 24xx series I2C EEPROMS, these functions should be easily useable for communicating with other I2C devices.

These routines should not be used for communicating with 12CE518/519 devices. The hardware requires different firmware. See 12CE51x datasheet Appendix A.

Using the I2C Library.

1. Edit file SWI2C12.ASM to choose the port and pins for SDA, SCL.
2. Specify the remaining TRIS bits for the port. This is very important because the I2C routines need to TRIS the port as a normal part of their operation. The TRIS bits are set at assemble time and there is no provision to change them at run time.
3. Assemble the library, using the /o (object file) option.
4. Include SWI2C12.INC in your program This defines the names of the interface variables.
5. EXTERN the functions names that you need to use. Any function names that are EXTERNed will be linked into the final hex file.
6. Assemble your program as normal and link together with the library generated in step 3.

| | **EEPROM** | **Clock cycles** | **File Registers** | **Stack Levels** | **IO Pins** |
|---|---|---|---|---|---|
| **$I^2C$ Library** | $116_{10}$ | | 9* | | 2 |
| **OpenSWI2C** | | 3 | | 1 | |
| **putcSWI2C** | | 508 | | 2 | |
| **getcSWI2C** | | 627 | | 2 | |
| **getnextcSWI2C** | | 284 | | 2 | |
| **SetI2CSlave** | 1 | 1 | | 0 | |
| **SetI2CAddress** | 1 | 1 | | 0 | |

* 5 of the 9 registers are overlayable

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## OpenSWI2C

Function Sets the TRIS register and initial conditions for the SDA and SCL lines.

Include File SWI2C12.INC

Options none

Syntax call OpenSWI2C

Remarks The Software I2C library must be reassembled to change Port, Pins or TRIS mask. These are defines at the top of SWI2C.ASM.

Return Value None

Limitations

- SDA and SCL must be on the same port.

Error Conditions None

See Also AN567

Example:

```
include <SWI2C12.INC>
…
…
call    OpenSWI2C
…
…
```

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## putcSWI2C

Function Write a byte to an I2C device.

Include File SWI2C12.INC

Options None

Syntax Load the byte to write into the W register,
call putcSWI2C.

Remarks

- The I2C port must be opened prior to calling putcSWI2C, and the Slave and Address must be defined (see SetSlave and SetAddress).
- Due to the read-modify-write nature of the BSF and BCF instructions, the remaining bits on the port may have changed during the course of the transmission. For more information, see paragraph 5.6.1 “Bi-Directional Data Port” of the 16c5x and enhanced 16c5x datasheets. (Paragraph 5.4.1 of the 12c50x datasheet).

Return Value None

Limitations 2 byte address parts only; 3 byte addresses not supported.

Error Conditions None

See Also AN567, OpenSWI2C, SetSlaveSWI2C, SetAddressSWI2C

Example:

```
include <SWI2C12.INC>
…
…
call    OpenSWI2C
movlw   b'10100000'
movwf   I2CSlave
movlw   0x05
movwf   I2CAddress
movlw   ‘A’
call    I2CWriteByte
```

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## getcSWI2C

Function Reads a byte from an I2C port. Sends the I2C address first, then requests a read from that address.

Include File SWI2C12.INC

Options None

Syntax call Open I2C port,
SetI2CSlave
SetI2CAddress
call getcSWI2C.

Remarks Byte is read in single byte read mode. This sends a stop bit after the byte is read, leaving the bus free for the next transaction.

Return Value byte read returned in I2CRxbuf

Limitations

Error Conditions None

See Also AN567, I2COpenPort, I2CSetSlave, I2CSetAddress

Example:

```
#include <SWI2C12.INC>
…
…
…
movlw   b'10100000'
SetI2CSlave
movlw   0x07
SetI2CAddress    ; load address
call    getcSWI2C
```

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## getnextcSWI2C

Function Reads the next byte from the I2C device. GetnextcSWI2C differs from getcSWI2C in that the slave and address bytes are not transmitted. Rather it depends on the address counter internal to the device being read. Since two fewer bytes are transmitted, throughput is increased.

Include File SWI2C12.INC

Options none

Syntax call getnextSWI2C

Remarks

- Due to the read-modify-write nature of the BSF and BCF instructions, the remaining bits on the port may have changed during the course of the transmission. For more information, see paragraph 5.6.1 “Bi-Directional Data Port” of the 16c5x and enhanced 16c5x datasheets. (Paragraph 5.4.1 of the 12c50x datasheet).
- If writing to a memory part, delay at least 10 mS between writes to allow part to complete previous write.

Return Value Byte returned in I2CRxbuf

Limitations

Error Conditions None

See Also AN567

Example:

```
include <SWI2C12.inc>
…
…
movlw   0xA0
SetI2CSlave
clrf    I2CAddress
call    getcSWI2C       ; read first byte
movf    I2CRxbuf,w
…
call    getnextcSWI2C   ; read next byte
 movf   I2CRxbuf,w
…
…
```

# Software Parts & Peripherals

**Callable Software functions extracted from the ECH**

## SetI2CSlave

Function Macro to set the I2C Slave address (movwf I2CSlave).

Include File SWI2C12.INC

Options none

Syntax Load slave address in Wreg,
SetI2CSlave

Remarks none

Return Value none

Limitations none

Error Conditions None

Size

| EEPROM | Clock cycles | File Registers | Stack Levels | IO Pins |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |

See Also AN567

Example:

```
include <SWI2C12.INC>
…
…
movlw   0xA0
SetI2CSlave
clrf    I2CAddress

call    I2CReadByte     ; read first byte
movf    I2CRxbuf,w
…
call    I2CReadNext     ; read next byte
 movf   I2CRxbuf,w
…
```

# Software Parts & Peripherals

**Callable Software functions extracted from the ECH**

## SetI2CAddress

Function Macro to set the I2Caddress.

Include File SWI2C12.INC

Options none

Syntax SetI2CAddress

Remarks None

Return Value None

Limitations None

Error Conditions None

Size

| | EEPROM | Clock cycles | File Registers | Stack Levels | IO Pins |
|---|---|---|---|---|---|
| | 1 | 1 | 1 | 0 | 2 |

See Also AN567

Example:

```
include <SWI2C12.inc>
…
…
movlw   0xA0
SetI2CSlave
movlw   0x05
SetI2CAddress
call    I2CReadByte     ; read first byte
movf    I2CRxbuf,w
…
call    I2CReadNext     ; read next byte
 movf   I2CRxbuf,w
…
```

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## OpenSWSPI

Function Sets the pins to appropriate initial values and TRISs the port to set the direction bits.

Include File SWSPI12.INC

Options MODE

MODE should be set to one of the following at assembly time:
MODE0: Idle Low, data tx'd on falling edge
MODE1: Idle High, data tx'd on rising edge
MODE2: Idle Low, data tx'd on rising edge
MODE3: Idle High, Data tx'd on falling ednge

Syntax call OpenSWSPI

The Software SPI library must be recompiled to change MODE, PORT and PIN assignments at the top of the SWSPI12.ASM file.

Remarks none

Return Value None

Limitations CLK, DI and DO must be on the same port.

- Due to the read-modify-write nature of the BSF and BCF instructions, the remaining bits on the port may have changed during the course of the transmission. For more information, see paragraph 5.6.1 “Bi-Directional Data Port” of the 16c5x and enhanced 16c5x datasheets. (Paragraph 5.4.1 of the 12c50x datasheet).

Error Conditions None

Size

| | EEPROM | Clock cycles | File Registers | Stack Levels | IO Pins |
|---|---|---|---|---|---|
| | 6 | 6 | 0 | 1 | 3 |

See Also

Example:

```
include <SWSPI12.INC>
…
…
…
call    OpenSWSPI
```

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## putcSWSPI

Function Reads and Writes a byte on the SPI port. On each cycle of the clock line (CLK) a bit is both read on DI and written on DO.

Include File SWSPI12.INC

Options none

Syntax load the byte to be transmitted in Wreg.
Call putcSWSPI
returns incoming byte read in datain

Remarks

- different name for the same function as WriteSWSPI
- Due to the read-modify-write nature of the BSF and BCF instructions, the remaining bits on the port may have changed during the course of the transmission. For more information, see paragraph 5.6.1 "Bi-Directional Data Port" of the 16c5x and enhanced 16c5x datasheets. (Paragraph 5.4.1 of the 12c50x datasheet).

Return Value Byte read in returned in datain.

Limitations None

Error Conditions None

Size

| | EEPROM | Clock cycles | File Registers | Stack Levels | IO Pins |
|---|---|---|---|---|---|
| | | | 3 | 1 | 4 |

See Also AN

Example:

```
include <SWSPI12.inc>
…
…
…
call    OpenSWSPI       ; set initial values on pins and tris port.
…
…
movlw   'A'             ; byte to write.
call    putcSWSPI       ;  write and read a byte
movlw   datain          ; get byte read in
…
```

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## WriteSWSPI

Function Reads and Writes a byte on the SPI port. On each cycle of the clock line (CLK) a bit is both read on DI and written on DO.

Include File SWSPI12.INC

Options MODE (set at module assembly time. See OpenSWSPI for details)

Syntax load byte to transmit in Wreg
call WriteSWSPI
returns incoming byte read in datain

Remarks

- Due to the read-modify-write nature of the BSF and BCF instructions, the remaining bits on the port may have changed during the course of the transmission. For more information, see paragraph 5.6.1 "Bi-Directional Data Port" of the 16c5x and enhanced 16c5x datasheets. (Paragraph 5.4.1 of the 12c50x datasheet).

Return Value Byte read in returned in datain.

Limitations None

Error Conditions None

Size

| | EEPROM | Clock cycles | File Registers | Stack Levels | IO Pins |
|---|---|---|---|---|---|
| | | | | 1 | 4 |

See Also AN

Example:

```
include <SWSPI12.INC>
…
…
…
call    OpenSWSPI       ; set initial values on pins and tris port.
…
…
movlw   'A'             ; byte to write.
call    putcSWSPI       ;  write and read a byte
movlw   datain          ; get byte read in
…
```

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## SetCSSWSPI

Function Macro to set the Chip Select line.

Include File SWSPI12.INC

Options SW_SPI_PORT and SW_CS_PIN must be edited at the top of the SWSPI.INC file.

Syntax SetCSSWSPI

Remarks

- Due to the read-modify-write nature of the BSF and BCF instructions, the remaining bits on the port may have changed during the course of the transmission. For more information, see paragraph 5.6.1 "Bi-Directional Data Port" of the 16c5x and enhanced 16c5x datasheets. (Paragraph 5.4.1 of the 12c50x datasheet).

Return Value none

Limitations None

Error Conditions None

Size

| | EEPROM | Clock cycles | File Registers | Stack Levels | IO Pins |
|---|---|---|---|---|---|
| | 1 | 1 | 0 | 0 | 1 |

See Also AN

Example:

```
include <SWSPI12.INC>
…
…
…
call    OpenSWSPI       ; set initial values on pins and tris port.
…
…
SetCSSWSPI
movlw   'A'             ; byte to write.
call    putcSWSPI       ;  write and read a byte
movlw   datain          ; get byte read in
ClearCSSWSPI
…
```

# Software Parts & Peripherals

**Callable Software functions extracted from the ECH**

## ClearCSSWSPI

Function Macro to clear the Chip Select line.

Include File SWSPI12.INC

Options SW_SPI_PORT and SW_CS_PIN must be edited at the top of the SWSPI.INC file.

Syntax ClearSWSPI

Remarks

- Due to the read-modify-write nature of the BSF and BCF instructions, the remaining bits on the port may have changed during the course of the transmission. For more information, see paragraph 5.6.1 “Bi-Directional Data Port” of the 16c5x and enhanced 16c5x datasheets. (Paragraph 5.4.1 of the 12c50x datasheet).

Return Value None

Limitations None

Error Conditions None

Size

| | EEPROM | Clock cycles | File Registers | Stack Levels | IO Pins |
|---|---|---|---|---|---|
| | 1 | 1 | 0 | 0 | 1 |

See Also AN

Example:

```
include <SWSPI12.INC>
…
…
…
call    OpenSWSPI       ; set initial values on pins and tris port.
…
…
movlw   ‘A’             ; byte to write.
call    putcSWSPI       ;  write and read a byte
movlw   datain          ; get byte read in
…
```

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## Wait on Pin Change Macros

While deceptively simple, these macros form the basis of other functions, such as zero crossing detect.

The wait on pin change macros watch a specific pin and wait for 3 transition types:
Low to High
High to Low
Any change (Low to High or High to Low)

With external hardware to condition the signal, these can be used to detect zero crossings. See AN 521 which shows how to interface 115VAC power lines to IO pins using only a current limiting resistor.

The port and pin must be setup as an input (high impedance) either as the result of a TRIS instruction or by using the OpenPinChangePort macro prior to using the WaitOnPinChange macros.

# Software Parts & Peripherals

**Callable Software functions extracted from the ECH**

## OpenPinChange (Macro)

Function Opens a port for pin change

Include File <PinChang.inc>

Options none

Syntax OpenPinChangePort <Port>,<Pin>,<TRIS value>

Remarks Performs TRIS instruction to set up Port and pin for input. Remainder of TRIS register must be specified.

Return Value None

Limitations Only one pin at a time may be monitored

Error Conditions None

Size

| | EEPROM | Clock cycles | File Registers | Stack Levels | IO Pins |
|---|---|---|---|---|---|
| **OpenPinChangePort** | 2 | 2 | 0 | 0 | 1 |
| **WaitOnPinChange** | 7 | * | 0 | 0 | 1 |
| **WaitOnPinChangeHL** | 4 | * | 0 | 0 | 1 |
| **WaitOnPinChangeLH** | 4 | * | 0 | 0 | 1 |

* Clock cycles depend on an external event.

See Also AN521 (interfacing IO pins to AC power lines) for

Example:

```
include <pinchang.inc>
…
…
OpenPinChangePort PORTA, 3, 0x00
```

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## WaitOnPinChange (Macro)

Function Waits in a tight loop until the specified input pin changes state (High to Low or Low to High).

Include File <PinChang.inc>

Options none

Syntax WaitOnPinChange <Port>,<Pin>

Remarks Port and pin must be set for high impedance either as the result of a TRIS instruction or by using the OpenPinChangePort macro.

Return Value None

Limitations Only one pin at a time may be monitored

Error Conditions None

Size

| | EEPROM | Clock cycles | File Registers | Stack Levels | IO Pins |
|---|---|---|---|---|---|
| **OpenPinChangePort** | 2 | 2 | 0 | 0 | 1 |
| **WaitOnPinChange** | 7 | * | 0 | 0 | 1 |
| **WaitOnPinChangeHL** | 4 | * | 0 | 0 | 1 |
| **WaitOnPinChangeLH** | 4 | * | 0 | 0 | 1 |

* Clock cycles depend on an external event.

See Also AN521 (interfacing IO pins to AC power lines) for

Example:

```
include <pinchang.inc>
…
…
WaitOnPinChange PORTA, 3
```

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## WaitOnPinChangeLH (Macro)

Function Waits in a tight loop until a rising edge is detected on the specified input pin (pin changes state from Low to High).

Include File <PinChang.inc>

Options none

Syntax WaitOnPinChangeLH <Port>,<Pin>

Remarks None

Return Value Port and pin must be set for high impedance either as the result of a TRIS instruction or by using the OpenPinChangePort macro.

Limitations Only one pin at a time may be monitored

Error Conditions None

Size

| | EEPROM | Clock cycles | File Registers | Stack Levels | IO Pins |
|---|---|---|---|---|---|
| **OpenPinChangePort** | 2 | 2 | 0 | 0 | 1 |
| **WaitOnPinChange** | 7 | * | 0 | 0 | 1 |
| **WaitOnPinChangeHL** | 4 | * | 0 | 0 | 1 |
| **WaitOnPinChangeLH** | 4 | * | 0 | 0 | 1 |

* Clock cycles depend on an external event.

See Also AN521 (interfacing IO pins to AC power lines) for

Example:

```
include <pinchang.inc>
…
…
WaitOnPinChangeLH PORTA, 3
```

# Software Parts & Peripherals

### Callable Software functions extracted from the ECH

## WaitOnPinChangeHL (Macro)

Function Waits in a tight loop until a falling edge is detected on the specified input pin (pin changes state from High to Low).

Include File <PinChang.inc>

Options none

Syntax WaitOnPinChangeHL <Port>,<Pin>

Remarks Port and pin must be set for high impedance either as the result of a TRIS instruction or by using the OpenPinChangePort macro.

Return Value None

Limitations Only one pin at a time may be monitored

Error Conditions None

Size

| | EEPROM | Clock cycles | File Registers | Stack Levels | IO Pins |
|---|---|---|---|---|---|
| **OpenPinChangePort** | 2 | 2 | 0 | 0 | 1 |
| **WaitOnPinChange** | 7 | * | 0 | 0 | 1 |
| **WaitOnPinChangeHL** | 4 | * | 0 | 0 | 1 |
| **WaitOnPinChangeLH** | 4 | * | 0 | 0 | 1 |

* Clock cycles depend on an external event and therefore is non-deterministic

See Also AN521 (interfacing IO pins to AC power lines) for

Example:

```
include <pinchang.inc>
…
…
WaitOnPinChangeHL PORTA, 3
```