```
#ifndef PIC32_SPI_HAL
#define PIC32_SPI_HAL

#include <stdbool.h>

typedef enum {
    SPI_SPI1 = 0,
    SPI_SPI2 = 1
} SPI_Module_t;

typedef enum {
        SPI_RPA0 = 0,
        SPI_RPA1,
        SPI_RPA2,
        SPI_RPA3,
        SPI_RPA4,
        SPI_RPB0,
        SPI_RPB1,
        SPI_RPB2,
        SPI_RPB3,
        SPI_RPB4,
        SPI_RPB5,
        SPI_RPB6,
        SPI_RPB7,
        SPI_RPB8,
        SPI_RPB9,
        SPI_RPB10,
        SPI_RPB11,
        SPI_RPB12,
        SPI_RPB13,
        SPI_RPB14,
        SPI_RPB15,
        SPI_NO_PIN
} SPI_PinMap_t;

typedef enum {
    SPI_CLK_HI = 0,
    SPI_CLK_LO = 1
} SPI_Clock_t;

typedef enum {
    SPI_FIRST_EDGE = 0,
    SPI_SECOND_EDGE = 1
} SPI_ActiveEdge_t;

typedef enum {
    SPI_SMP_MID = 0,
    SPI_SMP_END = 1
} SPI_SamplePhase_t;

typedef enum {
    SPI_8BIT = 0,
    SPI_16BIT = 1,
    SPI_32BIT = 2
} SPI_XferWidth_t;

/****************************************************************************
 Function
    SPISetup_BasicConfig
```

```c
 Parameters
   SPI_Module_t: Which SPI module to be configured

 Returns
   bool: true if the module represents a legal module; otherwise, false

 Description
   Should be the first function called when setting up an SPI module.
   1) Disables the selected SPI Module
   2) Configures the SPI clock to be based on PBCLK
   3) Disables the Framed mode
   4) Disables the Audio mode
   Further function calls from the SPI HAL will be necessary to complete
   the module setup.

Example
   PortSetup_BasicConfig(SPI_SPI1);
****************************************************************************/
bool SPISetup_BasicConfig(SPI_Module_t WhichModule);

/****************************************************************************
 Function
    SPISetup_SetFollower

 Parameters
   SPI_Module_t: Which SPI module to be configured as Follower

 Returns
   bool: true if the module represents a legal module; otherwise, false

 Description
   Sets the selected SPI Module to Follower mode and configures the SPI CLK
   pin as an input. NOTE:
   1) Either this function or the SPISetup_SetLeader function
   should be called immediately after the call to SPISetup_BasicConfig.
   2) the PIC32 documentation refers to this mode as slave mode.

Example
   SPISetup_SetFollower(SPI_SPI1);
****************************************************************************/
bool SPISetup_SetFollower(SPI_Module_t WhichModule);

/****************************************************************************
 Function
    SPISetup_SetLeader

 Parameters
   SPI_Module_t: Which SPI module to be configured as leader
   SPI_SamplePhase_t: Sample phase for the data input

 Returns
   bool: true if the module represents a legal module and the requested
         sampling phase is legal; otherwise, false

 Description
   Sets the selected SPI Module to leader mode, configures the SPI CLK
   pin as an output, and sets the input sample phase.
   NOTE: 1) Either this function or the SPISetup_SetFollower function should
   be called immediately after the call to SPISetup_BasicConfig.
   2) the PIC32 documentation refers to this mode as master mode.
```

```c
Example
   SPISetup_SetLeader(SPI_SPI1, SPI_SMP_MID);
****************************************************************************/
bool SPISetup_SetLeader(SPI_Module_t WhichModule,
                        SPI_SamplePhase_t WhichPhase);

/****************************************************************************
 Function
    SPISetup_SetBitTime

 Parameters
   SPI_Module_t: Which SPI module to be configured
   uint32_t: Desired SPI bit-time in ns

 Returns
   bool: true if the module represents a legal module and the time specified
   will fit into the SPIBRG register (13 bits) ; otherwise, false

 Description
   Based on a 20MHz PBCLK, calculates and programs the SPIBRG register for the
   specified SPI module to achieve the requested bit time.

Example
   SPISetup_SetBitTime(SPI_SPI1, 100);
****************************************************************************/
bool SPISetup_SetBitTime(SPI_Module_t WhichModule,
                         uint32_t SPI_ClkPeriodIn_ns);

/****************************************************************************
 Function
    SPISetup_MapSSInput

 Parameters
   SPI_Module_t: Which SPI module to be configured
   SPI_PinMap_t: WhichPin will be mapped to the SS as an input

 Returns
   bool: true if the module represents a legal module and the pin specified
   can be mapped to the SS as an input; otherwise, false

 Description
   Sets the designated pin to be the SS input if the selected SPI Module
   is configured in Follower mode.
   Legal port pins for the SS1 input are:
   SPI_RPA0, SPI_RPB3, SPI_RPB4, SPI_RPB7,SPI_RPB15.
   Legal port pins for the SS2 input are:
   SPI_RPA3, SPI_RPB0, SPI_RPB9, SPI_RPB10,SPI_RPB14.

Example
   SPISetup_MapSSInput(SPI_SPI1, SPI_RPA0);
****************************************************************************/
bool SPISetup_MapSSInput(SPI_Module_t WhichModule, SPI_PinMap_t WhichPin);

/****************************************************************************
 Function
    SPISetup_MapSSOutput

 Parameters
   SPI_Module_t: Which SPI module to be configured
   SPI_PinMap_t: WhichPin will be mapped to the SS as an output

 Returns
   bool: true if the module represents a legal module, it is configured in
   leader mode and the pin specified can be mapped to the SS as an output;
   otherwise, false

 Description
   Sets the designated pin to be the SS output if the selected SPI Module
   is configured in Leader mode. Clears TRIS and ANSEL to make pin an output.
   Also configures INT4/INT1 to monitor for rising edges on the SS output pin.
   Legal port pins for the SS1 output are:
   SPI_NO_PIN, SPI_RPA0, SPI_RPB3, SPI_RPB4, SPI_RPB7,SPI_RPB15.
   Legal port pins for the SS2 output are:
   SPI_NO_PIN, SPI_RPA3, SPI_RPB0, SPI_RPB9, SPI_RPB10,SPI_RPB14.

Example
   SPISetup_MapSSOutput(SPI_SPI1, SPI_RPA0);
****************************************************************************/
bool SPISetup_MapSSOutput(SPI_Module_t WhichModule, SPI_PinMap_t WhichPin);

/****************************************************************************
 Function
    SPISetup_MapSDInput

 Parameters
   SPI_Module_t: Which SPI module to be configured
   SPI_PinMap_t: WhichPin will be mapped to the Serial Data input

 Returns
   bool: true if the module represents a legal module and the pin specified
   can be mapped to the Serial Data input; otherwise, false

 Description
   Sets the designated pin to be the SD input.
   Legal port pins for the SDI1 input are:
   SPI_NO_PIN, SPI_RPA1, SPI_RPB1, SPI_RPB5, SPI_RPB8,SPI_RPB11.
   Legal port pins for the SDI2 input are:
   SPI_NO_PIN, SPI_RPA2, SPI_RPA4, SPI_RPB2, SPI_RPB6,SPI_RPB13.

Example
   SPISetup_MapSDInput(SPI_SPI1, SPI_RPA0);
****************************************************************************/
bool SPISetup_MapSDInput(SPI_Module_t WhichModule, SPI_PinMap_t WhichPin);

/****************************************************************************
 Function
    SPISetup_MapSDOutput

 Parameters
   SPI_Module_t: Which SPI module to be configured.
   SPI_PinMap_t: WhichPin will be mapped to the Serial Data output

 Returns
   bool: true if the module represents a legal module and the pin specified
   can be mapped to the Serial Data output; otherwise, false

 Description
   Sets the designated pin to be the SD output.
   Legal port pins for the SDO1 & SDO2 outputs are:
```

```
   SPI_NO_PIN, SPI_RPA1, SPI_RPA2, SPI_RPA4, SPI_RPB1, SPI_RPB2, SPI_RPB5,
   SPI_RPB6, SPI_RPB8, SPI_RPB11, SPI_RPB13.

Example
   SPISetup_MapSDOutput(SPI_SPI1, SPI_RPA1);
****************************************************************************/
bool SPISetup_MapSDOutput(SPI_Module_t WhichModule, SPI_PinMap_t WhichPin);

/****************************************************************************
 Function
    SPISetup_SetClockIdleState

 Parameters
   SPI_Module_t: Which SPI module to be configured.
   SPI_Clock_t: SPI_CLK_HI or SPI_CLK_LO

 Returns
   bool: true if the module represents a legal module, that module is disabled,
   and the clock level is legal; otherwise, false

 Description
   ets the idle state of the SPI clock.

Example
   SPISetup_SetClockIdleState(SPI_SPI1, SPI_CLK_HI);
****************************************************************************/
bool SPISetup_SetClockIdleState(SPI_Module_t WhichModule,
                                SPI_Clock_t WhichState);

/****************************************************************************
 Function
    SPISetup_SetActiveEdge

 Parameters
   SPI_Module_t: Which SPI module to be configured.
   SPI_ActiveEdge_t: Which edge of the clock will be the active edge

 Returns
   bool: true if the module represents a legal module, that module is disabled,
   and the clock edge is legal; otherwise, false

 Description
   Sets the active edge of the SPI clock.

Example
   SPISetup_SetActiveEdge(SPI_SPI1, SPI_SECOND_EDGE);
****************************************************************************/
bool SPISetup_SetActiveEdge(SPI_Module_t WhichModule,
                            SPI_ActiveEdge_t WhichEdge);

/****************************************************************************
 Function
    SPISetup_SetXferWidth

 Parameters
   SPI_Module_t: Which SPI module to be configured.
   SPI_XferWidth_t: width of transfer

 Returns
   bool: true if the module represents a legal module and the width is
```

```c
   legal; otherwise, false

 Description
   Sets the width of the transfers that the SPI module will perform.

Example
   SPISetup_SetXferWidth(SPI_SPI1, SPI_8BIT);
****************************************************************************/
bool SPISetup_SetXferWidth(SPI_Module_t WhichModule,
                            SPI_XferWidth_t DataWidth);

/****************************************************************************
 Function
    SPISetEnhancedBuffer

 Parameters
   SPI_Module_t: Which SPI module to be configured.
   bool: if true enable enhanced buffer, if false disables enhanced buffer

 Returns
   bool: true if the module represents a legal module; otherwise, false

 Description
   Enables/disables the enhanced buffer on a module based on the second param

Example
   SPISetEnhancedBuffer(SPI_SPI1, true);
****************************************************************************/
bool SPISetEnhancedBuffer(SPI_Module_t WhichModule, bool IsEnhanced);

/****************************************************************************
 Function
    SPISetup_DisableSPI

 Parameters
   SPI_Module_t: Which SPI module to be disabled

 Returns
   bool: true if the module represents a legal module; otherwise, false

 Description
   Disables the selected SPI Module

Example
   SPISetup_DisableSPI(SPI_SPI1);
****************************************************************************/
bool SPISetup_DisableSPI(SPI_Module_t WhichModule);

/****************************************************************************
 Function
    SPISetup_EnableSPI

 Parameters
   SPI_Module_t: Which SPI module to be enabled

 Returns
   bool: true if the module represents a legal module; otherwise, false

 Description
   Enables the selected SPI Module
```

```c
Example
   SPISetup_EnableSPI(SPI_SPI1);
****************************************************************************/
bool SPISetup_EnableSPI(SPI_Module_t WhichModule);

/****************************************************************************
 Function
    SPIOperate_SPI1_Send8

 Parameters
   uint8_t:      The Data to be written

 Returns
   bool: true if the module represents a legal module configured for 8-bit
   transfers; otherwise, false

 Description
   Writes the 8-bit data to the selected SPI Module data register
  Does not check if there is room in the buffer.
  Note: separate functions provided for SPI1 & SPI2 in order to speed operation
  and allow the SPI to be run at higher bit rates

Example
   SPIOperate_SPI1_Send8(0);
****************************************************************************/
void SPIOperate_SPI1_Send8(uint8_t TheData);

/****************************************************************************
 Function
    SPIOperate_SPI1_Send16

 Parameters
   uint16_t:     The Data to be written

 Returns
  Nothing

 Description
   Writes the 16-bit data to the SPI1 Module data register
  Does not check if there is room in the buffer.
  Note: separate functions provided for SPI1 & SPI2 in order to speed operation
  and allow the SPI to be run at higher bit rates

Example
   SPIOperate_SPI1_Send16(0);
****************************************************************************/
void SPIOperate_SPI1_Send16( uint16_t TheData);

/****************************************************************************
 Function
    SPIOperate_SPI1_Send32

 Parameters
   uint32_t:     The Data to be written

 Returns
   bool: true if the module represents a legal module configured for 32-bit
   transfers; otherwise, false
 Description
   Writes the 32-bit data to the selected SPI Module data register
  Does not check if there is room in the buffer.
  Note: separate functions provided for SPI1 & SPI2 in order to speed operation
  and allow the SPI to be run at higher bit rates

Example
   SPIOperate_SPI1_Send32(0);
****************************************************************************/
void SPIOperate_SPI1_Send32(uint32_t TheData);

/****************************************************************************
 Function
    SPIOperate_SPI1_Send8Wait

 Parameters
   uint8_t:      The Data to be written

 Returns
   bool: true if the module represents a legal module configured for 8-bit
   transfers; otherwise, false

 Description
   Writes the 8-bit data to the selected SPI Module data register and waits
   for the SS line to rise. NOTE: this is blocking code and should only be
   used when the bit-time on the SPI is sufficiently fast so as need to wait
   less than 200 micro-seconds to complete.
   Does not check if there is room in the buffer.
  Note: separate functions provided for SPI1 & SPI2 in order to speed operation
  and allow the SPI to be run at higher bit rates

Example
   SPIOperate_SPI1_Send8Wait(0);
****************************************************************************/
void SPIOperate_SPI1_Send8Wait(uint8_t TheData);

/****************************************************************************
  Function
    SPIOperate_SPI1_Send16Wait

  Parameters
    uint16_t:     The Data to be written

  Returns
    Nothing

  Description
    Writes the 16-bit data to the SPI1 Module data register and waits
    for the SS1 line to rise. NOTE: this is blocking code and should only be
    used when the bit-time on the SPI is sufficiently fast so as need to wait
    less than 200 micro-seconds to complete.
    Does not check if there is room in the buffer.
  Note: separate functions provided for SPI1 & SPI2 in order to speed operation
  and allow the SPI to be run at higher bit rates

Example
   SPIOperate_SPI1_Send16(0);
****************************************************************************/
void SPIOperate_SPI1_Send16Wait( uint16_t TheData);
```

```c
/****************************************************************************
 Function
    SPIOperate_SPI1_Send32Wait

 Parameters
   SPI_Module_t: Which SPI module to be disabled
   uint32_t:     The Data to be written

 Returns
   bool: true if the module represents a legal module configured for 32-bit
   transfers; otherwise, false

 Description
   Writes the 32-bit data to the selected SPI Module data register and waits
   for the SS line to rise. NOTE: this is blocking code and should only be
   used when the bit-time on the SPI is sufficiently fast so as need to wait
   less than 200 micro-seconds to complete.
  Does not check if there is room in the buffer.
  Note: separate functions provided for SPI1 & SPI2 in order to speed operation
  and allow the SPI to be run at higher bit rates

Example
   SPIOperate_SPI1_Send32Wait(0);
****************************************************************************/
void SPIOperate_SPI1_Send32Wait(uint32_t TheData);

/****************************************************************************
 Function
    SPIOperate_ReadData

 Parameters
   SPI_Module_t: Which SPI module to be read

 Returns
   uint32_t: the data read from the specified SPI data register

 Description
   Reads the data register for the selected SPI Module. Note: If the selected
   module is in 8-bit or 16-bit mode, then you should cast the result of this
   function to a uint8_t or uint16_t before assignment to a result variable.

Example
   NewData32 = SPIOperate_Read(SPI_SPI1);
   NewData16 = (uint16_t)SPIOperate_Read(SPI_SPI1);
   NewData8 = (uint8_t)SPIOperate_Read(SPI_SPI1);
****************************************************************************/
uint32_t SPIOperate_ReadData(SPI_Module_t WhichModule);

/****************************************************************************
 Function
    SPIOperate_HasSS1_Risen

 Parameters
 None

 Returns
   bool: true if a rising edge has been observed on SS1; otherwise, false

 Description
   Tests if the SS1 line has risen since the last time this
   function was called.
   Note: This is an event checking function, not a state test. If the SS line
   is found to have risen, then the hardware will be reset until the next time
   that data is written to the SPI module. After a call to this function
   returns true, subsequent calls will return false until new data is written
   and another rising edge on the SS line is detected.
  Note: separate functions provided for SS1 & SS2 in order to speed operation
  and allow the SPI to be run at higher bit rates

Example
   if(true == SPIOperate_HasSS1_Risen())
       ;
****************************************************************************/
bool SPIOperate_HasSS1_Risen(void);


/****************************************************************************
 Function
    SPIOperate_HasSS2_Risen

 Parameters
 None

 Returns
   bool: true if a rising edge has been observed on SS2; otherwise, false

 Description
   Tests if the SS2 line has gone low then back high since the last time this
   function was called.
   Note: This is an event checking function, not a state test. If the SS line
   is found to have risen, then the hardware will be reset until the next time
   that data is written to the SPI module. After a call to this function
   returns true, subsequent calls will return false until new data is written
   and another rising edge on the SS line is detected.
  Note: separate functions provided for SS1 & SS2 in order to speed operation
  and allow the SPI to be run at higher bit rates

Example
   if(true == SPIOperate_HasSS1_Risen())
       ;
****************************************************************************/
bool SPIOperate_HasSS2_Risen(void);


#endif //PIC32_SPI_HAL defined
```