```
/*****************************************************************************
 Module
     PIC32_SPI_HAL.c
 Description
     Source file for the PIC32 SPI Hardware Abstraction Layer used in ME218
 Notes
     This is the prototype. Students will re-create this functionality
 History
 When           Who     What/Why
 -------------- ---     --------
  10/03/21 12:32 jec    started coding
*****************************************************************************/
/*---------------------------- Include Files ----------------------------*/
#include <xc.h>
#include <stdbool.h>
#include "PIC32_SPI_HAL.h"

/*------------------------- External Variables -------------------------*/

/*---------------------------- Module Defines ----------------------------*/
// this is based on a 13 bit (max=8191) BRG register and 20MHz (50ns) PBCLK
#define MAX_SPI_PERIOD  ((8191+1)*2*50)
#define MAP_SS1 0b0011
#define MAP_SS2 0b0100
#define MAP_SDO1 0b0011
#define MAP_SDO2 0b0100

/*---------------------------- Module Types ----------------------------*/

/*---------------------------- Module Functions ----------------------------*/
static void selectModuleRegisters(SPI_Module_t WhichModule);
static bool isSPI_ModuleLegal( SPI_Module_t WhichModule);
static bool isSS_OutputPinLegal(SPI_Module_t WhichModule,
                                SPI_PinMap_t WhichPin);
static bool isSDOPinLegal( SPI_PinMap_t WhichPin);
static bool isSDIPinLegal( SPI_PinMap_t WhichPin);

/*---------------------------- Module Variables ----------------------------*/
  // these will allow us to reference both SPI1 & SPI2 through these pointers
static volatile __SPI1CONbits_t * pSPICON;
static volatile __SPI1CON2bits_t * pSPICON2;
static volatile uint32_t * pSPIBRG;
static volatile uint32_t * pSPIBUF;

// these are the output mapping registers indexed by the SPI_PinMap_t value
static volatile uint32_t * const outputMapRegisters[] = { &RPA0R, &RPA1R,
                    &RPA2R, &RPA3R, &RPA4R,
                    &RPB0R, &RPB1R, &RPB2R, &RPB3R, &RPB4R, &RPB5R,
                    &RPB6R, &RPB7R, &RPB8R, &RPB9R, &RPB10R, &RPB11R, &RPB12R,
                    &RPB13R, &RPB14R, &RPB15R
};

// these are the TRISxSET registers indexed by the SPI_PinMap_t value
static volatile uint32_t * const setTRISRegisters[] = { &TRISASET, &TRISASET,
            &TRISASET, &TRISASET, &TRISASET,
            &TRISBSET, &TRISBSET, &TRISBSET, &TRISBSET, &TRISBSET, &TRISBSET,
            &TRISBSET, &TRISBSET, &TRISBSET, &TRISBSET, &TRISBSET, &TRISBSET,
            &TRISBSET, &TRISBSET, &TRISBSET, &TRISBSET
};
```

```
// these are the TRISxCLR registers indexed by the SPI_PinMap_t value
static volatile uint32_t * const clrTRISRegisters[] = { &TRISACLR, &TRISACLR,
            &TRISACLR, &TRISACLR, &TRISACLR,
            &TRISBCLR, &TRISBCLR, &TRISBCLR, &TRISBCLR, &TRISBCLR, &TRISBCLR,
            &TRISBCLR, &TRISBCLR, &TRISBCLR, &TRISBCLR, &TRISBCLR, &TRISBCLR,
            &TRISBCLR, &TRISBCLR, &TRISBCLR, &TRISBCLR
};

// these are the ANSELxCLR registers indexed by the SPI_PinMap_t value
static volatile uint32_t * const clrANSELRegisters[] = { &ANSELACLR, &ANSELACLR,
         &ANSELACLR, &ANSELACLR, &ANSELACLR,
         &ANSELBCLR, &ANSELBCLR, &ANSELBCLR, &ANSELBCLR, &ANSELBCLR, &ANSELBCLR,
         &ANSELBCLR, &ANSELBCLR, &ANSELBCLR, &ANSELBCLR, &ANSELBCLR, &ANSELBCLR,
         &ANSELBCLR, &ANSELBCLR, &ANSELBCLR, &ANSELBCLR
};

// these are the bit positions indexed by the SPI_PinMap_t value
static uint32_t const mapPinMap2BitPosn[] = { 1<<0, 1<<1,
         1<<2, 1<<3, 1<<4,
         1<<0, 1<<1, 1<<2, 1<<3, 1<<4, 1<<5,
         1<<6, 1<<7, 1<<8, 1<<9, 1<<10, 1<<11,
         1<<12, 1<<13, 1<<14, 1<<15
};

// these are the INT pin mapping constants indexed by the SPI_PinMap_t value
static uint32_t const mapPinMap2INTConst[] = { 0b0000/*RA0*/, 0b0000/*RA1*/,
         0b0000/*RA2*/, 0b0000/*RA3*/, 0b0010/*RA4*/,
         0b0010/*RB0*/,0b0010/*RB1*/, 0b0100/*RB2*/, 0b0001/*RB3*/,
         0b0010/*RB4*/, 0b0001/*RB5*/, 0b0001/*RB6*/, 0b0100/*RB7*/,
         0b0100/*RB8*/, 0b0100/*RB9*/, 0b0011/*RB10*/, 0b0011/*RB11*/,
         0/*RB12*/, 0b0011/*RB13*/, 0b0001/*RB14*/, 0b0011/*RB15*/
};

static SPI_PinMap_t const LegalSSOutPins[][5] = {{ SPI_RPA0, SPI_RPB3, SPI_RPB4,
                                                   SPI_RPB7,SPI_RPB15 },
                                                   { SPI_RPA3, SPI_RPB0, SPI_RPB9,
                                                   SPI_RPB10,SPI_RPB14 }
};

static SPI_PinMap_t const LegalSDOxPins[] = { SPI_NO_PIN, SPI_RPA1, SPI_RPA2,
                                               SPI_RPA4, SPI_RPB1, SPI_RPB2,
                                               SPI_RPB5, SPI_RPB6, SPI_RPB8,
                                               SPI_RPB11, SPI_RPB13
};

static SPI_PinMap_t const LegalSDIxPins[] = { SPI_NO_PIN, SPI_RPA1, SPI_RPB5,
                                               SPI_RPB1, SPI_RPB11, SPI_RPB8,SPI_RPA2,
                                               SPI_RPB6,SPI_RPA4,SPI_RPB13,SPI_RPB2



};

/*------------------------------ Module Code ------------------------------*/


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

```
 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.
*****************************************************************************/
bool SPISetup_BasicConfig(SPI_Module_t WhichModule)
{
  bool ReturnVal = true;

  // Make sure that we have a legal module specified
  if ( false == isSPI_ModuleLegal(WhichModule))
  {
      ReturnVal = false;
  }else // Legal module so set it up
  {
     selectModuleRegisters(WhichModule);

    pSPICON->ON = 0;         // Disable the selected SPI Module
    pSPICON->MCLKSEL = 0;    // Configure the SPI clock to be based on PBCLK
    pSPICON->FRMEN = 0;      // Disable the Framed mode
    pSPICON2->AUDEN = 0;     // Disable the Audio mode
  }
  return ReturnVal;
}

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

 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.
*****************************************************************************/
bool SPISetup_SetFollower(SPI_Module_t WhichModule)
{
  bool ReturnVal = true;
  // Your Code goes here :-)

  if (isSPI_ModuleLegal(WhichModule) ) {//If WhichModule is legal
      selectModuleRegisters(WhichModule);//select correct registers

      if (pSPICON->ON == 0){ //if SPI is off
        pSPICON->MSTEN = 0;//set MSTEN bit to follower mode (0)
        SPI_PinMap_t clkPin;
        if (WhichModule == SPI_SPI1){
          //CLK pin is B14 for SPI1
          clkPin = SPI_RPB14;
        } else{
          //CLK pin is B15 for SPI2
          clkPin = SPI_RPB15;
        }
        // set the TRIS bit to make it an input
        *setTRISRegisters[clkPin] = mapPinMap2BitPosn[clkPin];
```

```
        // clear the ANSEL bit to disable analog on the pin
        *clrANSELRegisters[clkPin] = mapPinMap2BitPosn[clkPin];
    }
  } else{
      ReturnVal = false;
  }


  return ReturnVal;
}

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

 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.
*****************************************************************************/
bool SPISetup_SetLeader(SPI_Module_t WhichModule, SPI_SamplePhase_t WhichPhase)
{
  bool ReturnVal = true;
  // Your Code goes here :-)

  if (isSPI_ModuleLegal(WhichModule)) {//If WhichModule is legal
      selectModuleRegisters(WhichModule); //select correct registers

      if (pSPICON->ON == 0){ //if SPI is off
        pSPICON->MSTEN = 1;//set MSTEN bit to leader mode (1)

        SPI_PinMap_t clkPin;
        if (WhichModule == SPI_SPI1){//SPI1
          //CLK pin is B14 for SPI1
          clkPin = SPI_RPB14;
        } else{//SPI2
          //CLK pin is B15 for SPI2
          clkPin = SPI_RPB15;
        }
        // clear the TRIS bit to make it an output
        *clrTRISRegisters[clkPin] = mapPinMap2BitPosn[clkPin];
        // clear the ANSEL bit to disable analog on the pin
        *clrANSELRegisters[clkPin] = mapPinMap2BitPosn[clkPin];

        //set the SMP to WhichPhase
        pSPICON->SMP = WhichPhase;
      }
  } else{//illegal module
      ReturnVal = false;
  }

  return ReturnVal;
}

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

```
 Description
   Based on a 20MHz PBCLK, calculates and programs the SPIBRG register for the
   specified SPI module to achieve the requested bit time.
*****************************************************************************/
bool SPISetup_SetBitTime(SPI_Module_t WhichModule, uint32_t SPI_ClkPeriodIn_ns)
{
  bool ReturnVal = true;

   if (isSPI_ModuleLegal(WhichModule) && SPI_ClkPeriodIn_ns <= MAX_SPI_PERIOD) {//If
WhichModule is legal and given period is valid
      selectModuleRegisters(WhichModule); //select correct registers

      if (pSPICON->ON == 0){ //if SPI is off and
        *pSPIBRG = 0.01* SPI_ClkPeriodIn_ns - 1;
      }

   }else{
        ReturnVal = false;
   }

  return ReturnVal;
}
/*****************************************************************************
 Function
    SPISetup_MapSSInput

 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.
*****************************************************************************/
bool SPISetup_MapSSInput(SPI_Module_t WhichModule, SPI_PinMap_t WhichPin)
{
  bool ReturnVal = true;

  // Make sure that we have a legal module specified & legal pin
  if ( (false == isSPI_ModuleLegal(WhichModule)) ||
       (false == isSS_OutputPinLegal(WhichModule, WhichPin)) )//legal input pins are
same as output pins
  {
    ReturnVal = false;
  }else
  { // Legal module  & pin so set try setting it up
    selectModuleRegisters(WhichModule);
    if (0 == pSPICON->MSTEN)  // configured in Follower mode?
    {
      if (SPI_NO_PIN == WhichPin)
      {
        pSPICON->SSEN = 0; // disable SS
      }else //there is an SS pin so map it
      {
        pSPICON->SSEN = 1; // enable SS
        // set the TRIS bit to make it an input
        *setTRISRegisters[WhichPin] = mapPinMap2BitPosn[WhichPin];
        // clear the ANSEL bit to disable analog on the pin
        *clrANSELRegisters[WhichPin] = mapPinMap2BitPosn[WhichPin];
```

```
        if (SPI_SPI1 == WhichModule)
        {
          SS1R = mapPinMap2INTConst[WhichPin];// Map SS1 to chosen pin
        }else
        {
          SS2R = mapPinMap2INTConst[WhichPin];// Map SS2 to chosen pin
        }
      }
    }else // not in follower mode
    {
      ReturnVal = false; // then we can't config an SS input
    }
  }
  return ReturnVal;
}

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

 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.
*****************************************************************************/
bool SPISetup_MapSSOutput(SPI_Module_t WhichModule, SPI_PinMap_t WhichPin)
{
  bool ReturnVal = true;

  // Make sure that we have a legal module specified & legal pin
  if ( (false == isSPI_ModuleLegal(WhichModule)) ||
       (false == isSS_OutputPinLegal(WhichModule, WhichPin)) )
  {
    ReturnVal = false;
  }else
  { // Legal module  & pin so set try setting it up
    selectModuleRegisters(WhichModule);
    if (1 == pSPICON->MSTEN)  // configured in Leader mode?
    {
      if (SPI_NO_PIN == WhichPin)
      {
        pSPICON->MSSEN = 0; // disable SS
      }else //there is an SS pin so map it
      {
        pSPICON->MSSEN = 1; // enable SS
        // clear the TRIS bit to make it an output
        *clrTRISRegisters[WhichPin] = mapPinMap2BitPosn[WhichPin];
        // clear the ANSEL bit to disable analog on the pin
        *clrANSELRegisters[WhichPin] = mapPinMap2BitPosn[WhichPin];

        if (SPI_SPI1 == WhichModule)
        {
          *outputMapRegisters[WhichPin] = MAP_SS1; // Map SS to chosen pin
          // set up to use INT4 to capture the rising edge of SS
          INTCONbits.INT4EP = 1;            // set for rising edge sensitivity
          IFS0CLR = _IFS0_INT4IF_MASK;      // clear any pending flag
```

```
          INT4R = mapPinMap2INTConst[WhichPin];  // map INT4 to SS as well
        }else
        {   // must be SPI2 so set up INT1
          *outputMapRegisters[WhichPin] = MAP_SS2; // Map SS to chosen pin
          // set up to use INT1 to capture the rising edge of SS
          INTCONbits.INT1EP = 1;            // set for rising edge sensitivity
          IFS0CLR = _IFS0_INT1IF_MASK;      // clear any pending flag
          INT1R = mapPinMap2INTConst[WhichPin];  // map INT1 to SS as well
        }
      }
    }else // not in Leader mode
    {
      ReturnVal = false; // then we can't config an SS output
    }
  }
  return ReturnVal;
}
/*****************************************************************************
 Function
    SPISetup_MapSDInput

 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.
*****************************************************************************/
bool SPISetup_MapSDInput(SPI_Module_t WhichModule, SPI_PinMap_t WhichPin)
{
   bool ReturnVal = true;
  // Your Code goes here :-)

  //If WhichModule is legal and provided pin is legal:
   if (isSPI_ModuleLegal(WhichModule) && isSDIPinLegal(WhichPin)) {
       selectModuleRegisters(WhichModule); //select correct registers

      if (pSPICON->ON == 0){ //if SPI is off
        if (WhichModule == SPI_SPI1){//SPI1
            SDI1R = mapPinMap2INTConst[WhichPin];// Map SDI1 to chosen pin
        } else{//SPI2
            SDI2R = mapPinMap2INTConst[WhichPin];// Map SDI1 to chosen pin

        }

        // set the TRIS bit to make it an input
          *setTRISRegisters[WhichPin] = mapPinMap2BitPosn[WhichPin];
          // clear the ANSEL bit to disable analog on the pin
          *clrANSELRegisters[WhichPin] = mapPinMap2BitPosn[WhichPin];
      }
   }else{
        ReturnVal = false;
   }

  return ReturnVal;
}

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

```
 Description
   Sets the designated pin to be the SD output.
*****************************************************************************/
bool SPISetup_MapSDOutput(SPI_Module_t WhichModule, SPI_PinMap_t WhichPin)
{
  bool ReturnVal = true;
  // Your Code goes here :-)

  //If WhichModule is legal and provided pin is legal:
   if (isSPI_ModuleLegal(WhichModule) && isSDOPinLegal(WhichPin)) {
       selectModuleRegisters(WhichModule); //select correct registers

      if (pSPICON->ON == 0){ //if SPI is off
        if (WhichModule == SPI_SPI1){//SPI1
          *outputMapRegisters[WhichPin] = MAP_SDO1; // Map SDO1 to chosen pin
        } else{//SPI2
          *outputMapRegisters[WhichPin] = MAP_SDO2; // Map SDO2 to chosen pin
        }

        // clear the TRIS bit to make it an output
          *clrTRISRegisters[WhichPin] = mapPinMap2BitPosn[WhichPin];
          // clear the ANSEL bit to disable analog on the pin
          *clrANSELRegisters[WhichPin] = mapPinMap2BitPosn[WhichPin];
      }
   }else{
        ReturnVal = false;
   }

  return ReturnVal;
}

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

 Description
   Sets the idle state of the SPI clock.
*****************************************************************************/
bool SPISetup_SetClockIdleState(SPI_Module_t WhichModule,
                                SPI_Clock_t WhichState)
{
  bool ReturnVal = true;
  // Your Code goes here :-)


  //If WhichModule is legal and  and provided pin is legal:
  if (isSPI_ModuleLegal(WhichModule) ) {
    selectModuleRegisters(WhichModule); //select correct registers

    if (pSPICON->ON == 0){ //if SPI is off
      if (WhichState == SPI_CLK_HI){
        pSPICON->CKP = 1; //set CKP bit
      }else if (WhichState == SPI_CLK_LO){
        pSPICON->CKP = 0; //clear CKP bit
      }else{ //WhichState was not legal
        ReturnVal = false;
      }
    }
   } else{
```

```
    ReturnVal = false;
   }


  return ReturnVal;

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

 Description
   Sets the active edge of the SPI clock.
*******************************************************************************/
bool SPISetup_SetActiveEdge(SPI_Module_t WhichModule,
                            SPI_ActiveEdge_t WhichEdge)
{
  bool ReturnVal = true;
  // Your Code goes here :-)

  //if WhichModule is legal
  if (isSPI_ModuleLegal(WhichModule) ) {
    selectModuleRegisters(WhichModule); //select correct registers

    if (pSPICON->ON == 0){ //if SPI is off
      if (WhichEdge == SPI_FIRST_EDGE){
        pSPICON->CKE = 1;//set CKE
      }else if (WhichEdge == SPI_SECOND_EDGE){
        pSPICON->CKE = 0; //clear CKE
      }else{ //WhichEdge was not legal
        ReturnVal = false;
      }
    }
  }else{
    ReturnVal = false;
  }

  return ReturnVal;

}

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

 Description
   Sets the width of the transfers that the SPI module will perform.
*******************************************************************************/
bool SPISetup_SetXferWidth(SPI_Module_t WhichModule,
                            SPI_XferWidth_t DataWidth)
{
  bool ReturnVal = true;
  // Your Code goes here :-)
  //if WhichModule is legal
  if (isSPI_ModuleLegal(WhichModule) ) {
    selectModuleRegisters(WhichModule); //select correct registers

    if (pSPICON->ON == 0){ //if SPI is off
      if (DataWidth == SPI_8BIT){
       pSPICON->MODE32 = 0; //clear MODE32
```

```
      pSPICON->MODE16 = 0; //clear MODE16
     }else if (DataWidth == SPI_16BIT){
       pSPICON->MODE32 = 0; //clear MODE32
       pSPICON->MODE16 = 1; //set MODE16
     }else if (DataWidth == SPI_32BIT){
       pSPICON->MODE32 = 1;//set MODE32
     }else{ //DataWidth was not legal
       ReturnVal = false;
     }
   }
  }else{
    ReturnVal = false;
  }

  return ReturnVal;
}

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

 Description
   Enables/disables the enhanced buffer on a module based on the second param
*****************************************************************************/
bool SPISetEnhancedBuffer(SPI_Module_t WhichModule, bool IsEnhanced)
{
  bool ReturnVal = true;
  // Your Code goes here :-)

  //if SPI module is legal
  if (isSPI_ModuleLegal(WhichModule) ) {
    selectModuleRegisters(WhichModule); //select correct registers
    if (pSPICON->ON == 0){ //if SPI is off
      if(IsEnhanced){
        pSPICON->ENHBUF = 1;//set ENHBUF
      } else{
        pSPICON->ENHBUF = 0;//clear ENHBUF
      }
    }
  }else{
    ReturnVal = false;
  }

  return ReturnVal;
}

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

 Description
   Disables the selected SPI Module
*****************************************************************************/
bool SPISetup_DisableSPI(SPI_Module_t WhichModule)
{
  bool ReturnVal = true;
  // Your Code goes here :-)

  //if SPI module is legal
  if (isSPI_ModuleLegal(WhichModule) ) {
```

```
    selectModuleRegisters(WhichModule); //select correct registers
    pSPICON->ON = 0;//clear ON bit

  }else{
    ReturnVal = false;
  }

  return ReturnVal;

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

 Description
   Enables the selected SPI Module
*********************************************************************/
bool SPISetup_EnableSPI(SPI_Module_t WhichModule)
{
  bool ReturnVal = true;
  // Your Code goes here :-)

 //if SPI module is legal
  if (isSPI_ModuleLegal(WhichModule) ) {
    selectModuleRegisters(WhichModule); //select correct registers
    pSPICON->ON = 1;//set ON bit

  }else{
    ReturnVal = false;
  }
  return ReturnVal;

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

 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
*********************************************************************/
void SPIOperate_SPI1_Send8(uint8_t TheData)
{
  // not needed for ME218a Labs
}

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

 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
*********************************************************************/
void SPIOperate_SPI1_Send16( uint16_t TheData)
{
```

```
    SPI1BUF = TheData;
}
/*****************************************************************************
 Function
    SPIOperate_SPI1_Send32

 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
*****************************************************************************/
void SPIOperate_SPI1_Send32(uint32_t TheData)
{
  // not needed for ME218a Labs
}

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

 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
*****************************************************************************/
void SPIOperate_SPI1_Send8Wait(uint8_t TheData)
{
    SPI1BUF = TheData;
    while (!SPIOperate_HasSS1_Risen()){
    ;
    }
}

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

  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
*****************************************************************************/
void SPIOperate_SPI1_Send16Wait( uint16_t TheData)
{
  // Your Code goes here :-)

  SPI1BUF = TheData;
  while (!SPIOperate_HasSS1_Risen()){
  ;
  }
```

```
}
/*****************************************************************************
 Function
    SPIOperate_SPI1_Send32Wait

 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
*****************************************************************************/
void SPIOperate_SPI1_Send32Wait(uint32_t TheData)
{
  // not needed for ME218a Labs
}
/*****************************************************************************
 Function
    SPIOperate_ReadData

 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.
*****************************************************************************/
uint32_t SPIOperate_ReadData(SPI_Module_t WhichModule)
{
  // not needed for ME218a Labs
}
/*****************************************************************************
 Function
    SPIOperate_HasSS1_Risen
 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
*****************************************************************************/
bool SPIOperate_HasSS1_Risen(void)
{
  bool ReturnVal = false;
  // Your Code goes here :-)

  //if INT4IF flag is set:
  if(IFS0bits.INT4IF == 1){
    IFS0CLR = _IFS0_INT4IF_MASK;// clear any pending flag
    ReturnVal = true;
  }else{
    ReturnVal = false;
  }

  return ReturnVal;
```

```
}

/*****************************************************************************
 Function
    SPIOperate_HasSS2_Risen
 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
*****************************************************************************/
bool SPIOperate_HasSS2_Risen(void)
{
  // not needed for ME218a
}


//*********************************
// private functions
//*********************************
/*****************************************************************************
 Function
    selectModuleRegisters

 Description
   based in the requested module, initializes the pointers to the various
   SPI module registers.
*****************************************************************************/
static void selectModuleRegisters(SPI_Module_t WhichModule)
{
  if (SPI_SPI1 == WhichModule)
  {
    pSPICON = (__SPI1CONbits_t *)&SPI1CON;
    pSPICON2 = (__SPI1CON2bits_t *)&SPI1CON2;
    pSPIBRG = &SPI1BRG;
    pSPIBUF = &SPI1BUF;
  }else if (SPI_SPI2 == WhichModule)
  {
    pSPICON = (__SPI1CONbits_t *)&SPI2CON;
    pSPICON2 = (__SPI1CON2bits_t *)&SPI2CON2;
    pSPIBRG = &SPI2BRG;
    pSPIBUF = &SPI2BUF;
  }
}

/*****************************************************************************
 Function
    isSPI_ModuleLegal

 Description
   Compares the requested module to the legal modules.

*****************************************************************************/
static bool isSPI_ModuleLegal( SPI_Module_t WhichModule)
{
```

```
  // Your Code goes here :-)
  bool ReturnVal = true;

  if (WhichModule == SPI_SPI1 || WhichModule == SPI_SPI2){//check if module is 1 or 2
    ;
  } else{
    ReturnVal = false;
  }

 return ReturnVal;

}

/*****************************************************************************
 Function
    isSS_OutputPinLegal

 Description
   Loops through the LegalSSOutPins array comparing the requested pin to each
   of the entries in the array. If a match is found, sets RetrnVal to true and
   breaks out of the loop.
*****************************************************************************/
static bool isSS_OutputPinLegal(SPI_Module_t WhichModule, SPI_PinMap_t WhichPin)
{
  bool ReturnVal = false;
  uint8_t index;

  for( index = 0;
       index <= ((sizeof(LegalSSOutPins[0])/sizeof(LegalSSOutPins[0][0]))-1);
       index++)
  {
    if (LegalSSOutPins[WhichModule][index] == WhichPin)
    {
      ReturnVal = true;
      break;
    }
  }
  return ReturnVal;
}

/*****************************************************************************
 Function
    isSDOPinLegal

 Description
   Loops through the LegalSDOxPins array comparing the requested pin to each
   of the entries in the array. If a match is found, sets RetrnVal to true and
   breaks out of the loop.
*****************************************************************************/
static bool isSDOPinLegal( SPI_PinMap_t WhichPin)
{
  bool ReturnVal = false;
  // Your Code goes here :-)

  uint8_t index;
   for( index = 0;
       index < (sizeof(LegalSDOxPins));
       index++)
  {
    if (LegalSDOxPins[index] == WhichPin){
```

```
      ReturnVal = true;
      break;
    }
  }

  return ReturnVal;
}

static bool isSDIPinLegal( SPI_PinMap_t WhichPin)
{
  bool ReturnVal = false;
  // Your Code goes here :-)

  uint8_t index;
   for( index = 0;
       index < (sizeof(LegalSDIxPins));
       index++)
  {
    if (LegalSDIxPins[index] == WhichPin){
      ReturnVal = true;
      break;
    }
  }

  return ReturnVal;
}
```