```
/*****************************************************************************
 Module
     PIC32PortHAL.c
 Description
     Source file for the PIC32 Port Hardware Abstraction Layer used in ME218
 Notes
     This is the prototype as an example of a Hardware Abstraction Layer.
 History
 When           Who     What/Why
 -------------- ---     --------
  10/11/22 15:41 jec    initial pass complete
  10/10/22 16:43 jec    started coding
*****************************************************************************/
/*---------------------------- Include Files ----------------------------*/
#include <xc.h>
#include <stdbool.h>
#include "PIC32PortHAL.h"
#include "bitdefs.h"

/*---------------------------- Module Defines ----------------------------*/
#define PORTA_BITS ((1<<0)|(1<<1)|(1<<2)|(1<<3)|(1<<4))
#define PORTB_BITS (0xFFFF)
#define ANA_A_BITS ((1<<0)|(1<<1))
#define ANA_B_BITS
((1<<0)|(1<<1)|(1<<2)|(1<<3)|(1<<11)|(1<<12)|(1<<13)|(1<<14)|(1<<15))


/*---------------------------- Module Types ----------------------------*/
// these are pointers to volatile registers, since the contents could change
// at anytime
typedef struct PortRegs {
  volatile uint32_t * ANSEL_;
  volatile uint32_t * ANSEL_CLR;
  volatile uint32_t * ANSEL_SET;
  volatile uint32_t * ANSEL_INV;
  volatile uint32_t * TRIS_;
  volatile uint32_t * TRIS_CLR;
  volatile uint32_t * TRIS_SET;
  volatile uint32_t * TRIS_INV;
  volatile uint32_t * PORT_;
  volatile uint32_t * PORT_CLR;
  volatile uint32_t * PORT_SET;
  volatile uint32_t * PORT_INV;
  volatile uint32_t * LAT_;
  volatile uint32_t * LAT_CLR;
  volatile uint32_t * LAT_SET;
  volatile uint32_t * LATAINV;
  volatile uint32_t * ODC_;
  volatile uint32_t * ODC_CLR;
  volatile uint32_t * ODC_SET;
  volatile uint32_t * ODC_INV;
  volatile uint32_t * CNPU_;
  volatile uint32_t * CNPU_CLR;
  volatile uint32_t * CNPU_SET;
  volatile uint32_t * CNPU_INV;
  volatile uint32_t * CNPD_;
  volatile uint32_t * CNPD_CLR;
  volatile uint32_t * CNPD_SET;
  volatile uint32_t * CNPDAINV;
```

```
  volatile uint32_t * CNCON_;
  volatile uint32_t * CNCON_CLR;
  volatile uint32_t * CNCON_SET;
  volatile uint32_t * CNCON_INV;
  volatile uint32_t * CNEN_;
  volatile uint32_t * CNEN_CLR;
  volatile uint32_t * CNEN_SET;
  volatile uint32_t * CNEN_INV;
  volatile uint32_t * CNSTAT_;
  volatile uint32_t * CNSTAT_CLR;
  volatile uint32_t * CNSTAT_SET;
  volatile uint32_t * CNSTAT_INV;
}PortRegs_t;

/*---------------------------- Module Functions ----------------------------*/
static bool IsLegalDigitalPin( PortSetup_Port_t WhichPort, PortSetup_Pin_t WhichPin );
static bool IsLegalAnalogPin( PortSetup_Port_t WhichPort, PortSetup_Pin_t WhichPin );

/*---------------------------- Module Variables ----------------------------*/
// make this const to put the array into flash, since none of these addresses
// will ever change
static const PortRegs_t PICMX170_Ports[2] =
{{  &ANSELA ,
  &ANSELACLR ,
  &ANSELASET ,
  &ANSELAINV ,
  &TRISA     ,
  &TRISACLR  ,
  &TRISASET  ,
  &TRISAINV  ,
  &PORTA     ,
  &PORTACLR  ,
  &PORTASET  ,
  &PORTAINV  ,
  &LATA      ,
  &LATACLR   ,
  &LATASET   ,
  &LATAINV   ,
  &ODCA      ,
  &ODCACLR   ,
  &ODCASET   ,
  &ODCAINV   ,
  &CNPUA     ,
  &CNPUACLR  ,
  &CNPUASET  ,
  &CNPUAINV  ,
  &CNPDA     ,
  &CNPDACLR  ,
  &CNPDASET  ,
  &CNPDAINV  ,
  &CNCONA    ,
  &CNCONACLR ,
  &CNCONASET ,
  &CNCONAINV ,
  &CNENA     ,
  &CNENACLR  ,
  &CNENASET  ,
  &CNENAINV  ,
  &CNSTATA   ,
  &CNSTATACLR,
```

```
  &CNSTATASET,
  &CNSTATAINV
 },
 {   &ANSELB    ,
  &ANSELBCLR ,
  &ANSELBSET ,
  &ANSELBINV ,
  &TRISB     ,
  &TRISBCLR  ,
  &TRISBSET  ,
  &TRISBINV  ,
  &PORTB     ,
  &PORTBCLR  ,
  &PORTBSET  ,
  &PORTBINV  ,
  &LATB      ,
  &LATBCLR   ,
  &LATBSET   ,
  &LATBINV   ,
  &ODCB      ,
  &ODCBCLR   ,
  &ODCBSET   ,
  &ODCBINV   ,
  &CNPUB     ,
  &CNPUBCLR  ,
  &CNPUBSET  ,
  &CNPUBINV  ,
  &CNPDB     ,
  &CNPDBCLR  ,
  &CNPDBSET  ,
  &CNPDBINV  ,
  &CNCONB    ,
  &CNCONBCLR ,
  &CNCONBSET ,
  &CNCONBINV ,
  &CNENB     ,
  &CNENBCLR  ,
  &CNENBSET  ,
  &CNENBINV  ,
  &CNSTATB   ,
  &CNSTATBCLR,
  &CNSTATBSET,
  &CNSTATBINV
 }
};

/*******************************************************************************
 Function
    PortSetup_ConfigureDigitalInputs

 Parameters
   PortSetup_Port_t: the port to be configured
   PortSetup_Pin_t: the pin to be configured as digital inputs

 Returns
   bool: true if port and pins represent legal ports and pins; otherwise, false

 Description
   Configures the specified pin(s) on the specified port as digital inputs, disabling
analog input(s).
```

```
Example
   PortSetup_ConfigureDigitalInputs(_Port_A, _Pin_0 | _Pin_1);
*****************************************************************************/
bool PortSetup_ConfigureDigitalInputs( PortSetup_Port_t WhichPort,
                                       PortSetup_Pin_t WhichPin){
    bool ReturnVal = false;
    if ( IsLegalDigitalPin(WhichPort, WhichPin))
    {
      ReturnVal = true;
      // start by disabling the analog inputs
      *PICMX170_Ports[WhichPort].ANSEL_CLR = WhichPin;

      // then configure the TRIS register to 1's for inputs
      *PICMX170_Ports[WhichPort].TRIS_SET = WhichPin;
    }

    return ReturnVal;
};

/*****************************************************************************
 Function
    PortSetup_ConfigureDigitalOutputs

 Parameters
   PortSetup_Port_t: the port to be configured
   PortSetup_Pin_t: the pin(s) to be configured as digital outputs

 Returns
   bool: true if port and pins represent legal ports and pins; otherwise, false

 Description
   Configures the specified pin(s) on the specified port as digital outputs
Example
   PortSetup_ConfigureDigitalOutputs(_Port_A, _Pin_0 | _Pin_1);
*****************************************************************************/
bool PortSetup_ConfigureDigitalOutputs( PortSetup_Port_t WhichPort,
                                        PortSetup_Pin_t WhichPin)
{
    bool ReturnVal = false;
    if ( IsLegalDigitalPin(WhichPort, WhichPin))
    {
      ReturnVal = true;
      // start by disabling the analog inputs
      *PICMX170_Ports[WhichPort].ANSEL_CLR = WhichPin;

      // then configure the TRIS register to 0's for outputs
      *PICMX170_Ports[WhichPort].TRIS_CLR = WhichPin;
    }

    return ReturnVal;
};

/*****************************************************************************
 Function
    PortSetup_ConfigureAnalogInputs

 Parameters
   PortSetup_Port_t: the port to be configured
   PortSetup_Pin_t: the pin(s) to be configured as analog inputs
```

```
 Returns
   bool: true if port and pins represent legal ports and pins; otherwise, false

 Description
   Configures the specified pin(s) on the specified port as analog inputs
Example
   PortSetup_ConfigureAnalogInputs(_Port_A, _Pin_0 | _Pin_1);
*****************************************************************************/
bool PortSetup_ConfigureAnalogInputs( PortSetup_Port_t WhichPort,
                                      PortSetup_Pin_t WhichPin)
{
    bool ReturnVal = false;
    if ( IsLegalAnalogPin(WhichPort, WhichPin))
    {
      ReturnVal = true;
      // start by disabling the digital outputs
      *PICMX170_Ports[WhichPort].TRIS_SET = WhichPin;

      // Then configure the analog inputs
      *PICMX170_Ports[WhichPort].ANSEL_SET = WhichPin;
    }

    return ReturnVal;
};


/*****************************************************************************
 Function
    PortSetup_ConfigurePullUps

 Parameters
   PortSetup_Port_t: the port to be configured
   PortSetup_Pin_t: the pin(s) to be configured with weak pull-ups

 Returns
   bool: true if port and pins represent legal ports and pins; otherwise, false

 Description
   Configures the specified pin(s) on the specified port with weak pull-ups
Example
   PortSetup_ConfigurePullUps(_Port_A, _Pin_0 | _Pin_1);
*****************************************************************************/
bool PortSetup_ConfigurePullUps( PortSetup_Port_t WhichPort,
                                 PortSetup_Pin_t WhichPin)
{
    bool ReturnVal = false;
    if ( IsLegalDigitalPin(WhichPort, WhichPin))
    {
      ReturnVal = true;
      // start by disabling the digital outputs
      *PICMX170_Ports[WhichPort].TRIS_SET = WhichPin;

      // then disable the pull-downs
      *PICMX170_Ports[WhichPort].CNPD_CLR = WhichPin;

      // Then configure the pull-ups
      *PICMX170_Ports[WhichPort].CNPU_SET = WhichPin;
    }

    return ReturnVal;
```

```
};

/*****************************************************************************
 Function
    PortSetup_ConfigurePullDowns

 Parameters
   PortSetup_Port_t: the port to be configured
   PortSetup_Pin_t: the pin(s) to be configured with weak pull-downs

 Returns
   bool: true if port and pins represent legal ports and pins; otherwise, false

 Description
   Configures the specified pin(s) on the specified port with weak pull-downs.
Example
   PortSetup_ConfigurePullDowns(_Port_A, _Pin_0 | _Pin_1);
*****************************************************************************/
bool PortSetup_ConfigurePullDowns( PortSetup_Port_t WhichPort,
                                   PortSetup_Pin_t WhichPin)
{
    bool ReturnVal = false;
    if ( IsLegalDigitalPin(WhichPort, WhichPin))
    {
      ReturnVal = true;
      // start by disabling the digital outputs
      *PICMX170_Ports[WhichPort].TRIS_SET = WhichPin;

      // then disable the pull-ups
      *PICMX170_Ports[WhichPort].CNPU_CLR = WhichPin;

      // Then configure the pull-downs
      *PICMX170_Ports[WhichPort].CNPD_SET = WhichPin;
    }

    return ReturnVal;
};

/*****************************************************************************
 Function
    PortSetup_ConfigureOpenDrain

 Parameters
   PortSetup_Port_t: the port to be configured
   PortSetup_Pin_t: the pin(s) to be configured as open drain outputs

 Returns
   bool: true if port and pins represent legal ports and pins; otherwise, false

 Description
   Configures the specified pin(s) on the specified port as open drain outputs
Example
   PortSetup_ConfigureOpenDrain(_Port_A, _Pin_0 | _Pin_1);
*****************************************************************************/
bool PortSetup_ConfigureOpenDrain( PortSetup_Port_t WhichPort,
                                   PortSetup_Pin_t WhichPin)
{
    bool ReturnVal = false;
    if ( IsLegalDigitalPin(WhichPort, WhichPin))
    {
```

```
      ReturnVal = true;
      // start by configuring the open drain outputs
      *PICMX170_Ports[WhichPort].ODC_SET = WhichPin;

      // Then, enable the digital outputs
      *PICMX170_Ports[WhichPort].TRIS_CLR = WhichPin;
    }

    return ReturnVal;
};


/*****************************************************************************
 Function
    PortSetup_ConfigureChangeNotification

 Parameters
   PortSetup_Port_t: the port to be configured
   PortSetup_Pin_t: the pin(s) to be enabled for change notification

 Returns
   bool: true if port and pins represent legal ports and pins; otherwise, false

 Description
   Configures the specified pin(s) on the specified port to enable change
   notifications. If any bits are set in the PortSetup_Pin_t parameter, then
   change notifications are enabled. If that parameter is 0, then change
   notifications are disabled globally.
Example
   PortSetup_ConfigureChangeNotification(_Port_A, _Pin_0 | _Pin_1);
*****************************************************************************/
bool PortSetup_ConfigureChangeNotification( PortSetup_Port_t WhichPort,
                                            PortSetup_Pin_t WhichPin)
{
    bool ReturnVal = false;
    if ( IsLegalDigitalPin(WhichPort, WhichPin))
    {
      ReturnVal = true;
      // decide if we should enable change notification globally
      if (0 != WhichPin) // at least 1 bit set, so enable
      {
        // the ON bit in the CNCONx register is in the bit 15 position
        *PICMX170_Ports[WhichPort].CNCON_SET = BIT15HI;
        // start by disabling the digital outputs, configuring as inputs
        *PICMX170_Ports[WhichPort].TRIS_SET = WhichPin;

        // Then enable change notification on the selected pins
        *PICMX170_Ports[WhichPort].CNEN_SET = WhichPin;
        // Then read the port to set the base state for changes
        *PICMX170_Ports[WhichPort].PORT_;
      }else
      { // if WhichPin is zero, then disable change notification globally
        *PICMX170_Ports[WhichPort].CNCON_CLR = BIT15HI;
      }
    }

    return ReturnVal;
};

/*****************************************************************************
```

```
 Function
    IsLegalDigitalPin

 Parameters
   PortSetup_Port_t: the port to be configured
   PortSetup_Pin_t: the pin(s) to be enabled for change notification

 Returns
   bool: true if port and pins represent legal ports and pins; otherwise, false

 Description
   Tests if the selected bits on the selected port are legal for digital I/O
   The result of the XOR will leave a 0 in any position that was legal and
   requested, with 1's in any position that was not legal (or not requested).
   By then ANDing with the requested bits we should only see 0's in the
   positions requested.
Example
   IsLegalDigitalPin(_Port_A, _Pin_0 | _Pin_1);
*****************************************************************************/
static bool IsLegalDigitalPin( PortSetup_Port_t WhichPort,
                               PortSetup_Pin_t WhichPin )
{
    bool ReturnVal = false;
    if(_Port_A == WhichPort)
    {
        if (0 == ((PORTA_BITS ^ WhichPin) & WhichPin))
        {
            ReturnVal = true;
        }
    } else {
        if(_Port_B == WhichPort)
        {
            if (0 == ((PORTB_BITS ^ WhichPin) & WhichPin))
            {
                ReturnVal = true;
            }
        }
    }
    return ReturnVal;
};

/*****************************************************************************
 Function
    IsLegalAnalogPin

 Parameters
   PortSetup_Port_t: the port to be configured
   PortSetup_Pin_t: the pin(s) to be enabled for change notification

 Returns
   bool: true if port and pins represent legal ports and pins; otherwise, false

 Description
   Tests if the selected bits on the selected port are legal for analog I/O
   The result of the XOR will leave a 0 in any position that was legal and
   requested, with 1's in any position that was not legal (or not requested).
   By then ANDing with the requested bits we should only see 0's in the
   positions requested.
Example
   IsLegalAnalogPin(_Port_A, _Pin_0 | _Pin_1);
```

```
*****************************************************************************/
static bool IsLegalAnalogPin( PortSetup_Port_t WhichPort,
                              PortSetup_Pin_t WhichPin )
{
    bool ReturnVal = false;
    if(_Port_A == WhichPort)
    {
        if (0 == ((ANA_A_BITS ^ WhichPin) & WhichPin))
        {
            ReturnVal = true;
        }
    } else {
        if(_Port_B == WhichPort)
        {
            if (0 == ((ANA_B_BITS ^ WhichPin) & WhichPin))
            {
                ReturnVal = true;
            }
        }
    }
    return ReturnVal;
};

#ifdef TEST_HARNESS
void main(void){
    uint8_t i;

    // Let's test the legal pins functions first
    for ( i=0; i<=4; i++) // these should all be legal, PORTA 0-4
    {
        if (!IsLegalDigitalPin(_Port_A, (1<<i)))
        {
            IsLegalDigitalPin(_Port_A, (1<<i)); // just a place to set a breakpoint
        }
    }

    for ( i=5; i<=31; i++) // these should all be illegal, any other PORTA
    {
        if (IsLegalDigitalPin(_Port_A, (1<<i)))
        {
            IsLegalDigitalPin(_Port_A, (1<<i)); // just a place to set a breakpoint
        }
    }

    for ( i=0; i<=15; i++) // these should all be legal, PORTB 0-15
    {
        if (!IsLegalDigitalPin(_Port_B, (1<<i)))
        {
            IsLegalDigitalPin(_Port_A, (1<<i)); // just a place to set a breakpoint
        }
    }

    for ( i=16; i<=31; i++) // these should all be illegal, any other PORTB
    {
        if (IsLegalDigitalPin(_Port_B, (1<<i)))
        {
            IsLegalDigitalPin(_Port_A, (1<<i)); // just a place to set a breakpoint
        }
    }
```

```
    // Next, let's test the legal analog pins functions
    for ( i=0; i<=1; i++) // these should all be legal, PORTA 0 & 1
    {
        if (!IsLegalAnalogPin(_Port_A, (1<<i)))
        {
            IsLegalDigitalPin(_Port_A, (1<<i)); // just a place to set a breakpoint
        }
    }

    for ( i=2; i<=31; i++) // these should all be illegal, any other PORTA
    {
        if (IsLegalAnalogPin(_Port_A, (1<<i)))
        {
            IsLegalDigitalPin(_Port_A, (1<<i)); // just a place to set a breakpoint
        }
    }

    for ( i=0; i<=3; i++) // these should all be legal, PORTB 0-3
    {
        if (!IsLegalAnalogPin(_Port_B, (1<<i)))
        {
            IsLegalDigitalPin(_Port_A, (1<<i)); // just a place to set a breakpoint
        }
    }

    for ( i=4; i<=10; i++) // these should all be illegal, PORTB 4-10
    {
        if (IsLegalAnalogPin(_Port_B, (1<<i)))
        {
            IsLegalDigitalPin(_Port_A, (1<<i)); // just a place to set a breakpoint
        }
    }

    for ( i=11; i<=15; i++) // these should all be legal, PORTB 11-15
    {
        if (!IsLegalAnalogPin(_Port_B, (1<<i)))
        {
            IsLegalDigitalPin(_Port_A, (1<<i)); // just a place to set a breakpoint
        }
    }

    for ( i=16; i<=31; i++) // these should all be illegal, any other PORTB
    {
        if (IsLegalAnalogPin(_Port_B, (1<<i)))
        {
            IsLegalDigitalPin(_Port_A, (1<<i)); // just a place to set a breakpoint
        }
    }


    PortSetup_ConfigureDigitalInputs(_Port_A, _Pin_0 | _Pin_1);
    PortSetup_ConfigureDigitalOutputs(_Port_A, _Pin_0 | _Pin_1);
    PortSetup_ConfigureAnalogInputs(_Port_A, _Pin_0 | _Pin_1);
    PortSetup_ConfigurePullUps(_Port_A, _Pin_0 | _Pin_1);
    PortSetup_ConfigurePullDowns(_Port_A, _Pin_0 | _Pin_1);
    PortSetup_ConfigureOpenDrain(_Port_A, _Pin_0 | _Pin_1);

}
#endif
```