PIC32 Pulse Width Modulation (PWM) Interface Module

Documentation Rev. 0.4 11/12/23
Code Rev. 1.0.1 01/19/22 Commit

Purpose:

This module provides a set of interface functions to perform a simplified initialization of the PWM system on the `PIC32MX170F256B` microcontroller. The initialization function must be called before any use is made of the PWM outputs.

How to obtain:

The library is obtained forking from the `PWMLib4PIC32` repository. This yields a runnable project that acts as a test harness that you can go back to at any time to check your PWM hardware on the PIC32.

Usage:

To use the PWM library in your project, copy the `PWM_PIC32.c` file to the `ProjectSource` folder in your project and the `PWM_PIC32.h` file into your `ProjectHeaders` folder, then add these two files into the Project window of your MPLAB X project. The `PWM_PIC32.h` file is added to the ***`Header Files`*** folder of the project and the `PWM_PIC32.c` file is added to the ***`Source Files`*** folder.
The file `PWM_PIC32.c` must be included in any project wishing to use the functions provided by this module.
The header file `PWM_PIC32.h` should be included in any module wishing to use the functions provided by this module.

Revision History:

October 25, 2021 Revisions for how to obtain & install library.

October 24, 2021 First draft of this document

Initialization: Basic

Function:

```
bool PWMSetup_BasicConfig(uint8_t HowMany)
```

Parameters:

`uint8_t HowMany`
: A number between 1 and 5 indicating the number of PWM outputs that you would like to configure.

Returns:

`bool`
: `True = Success; False = HowMany outside the legal range.`

Description:

Does the basic initialization of the PWM hardware on the PIC32 and the associated timers for use in generating PWM outputs. This call must be the first call in the initialization sequence but alone is not sufficient to generate PWM output. See later calls to assign a timer to the PWM channels, maps the outputs to pins, set frequency/period and duty-cycle/pulse width. If you request 1 timer, it will be Timer 1, if you request 2 timers, they will be Timer1 & Timer2, etc.

Usage:

```
PWMSetup_BasicConfig(4); // please excuse the magic number
```

This call would do the basic initialization of the PWM system to output 4 channels of PWM.

Initialization: Assign Channel to Timer

Function:

```
bool PWMSetup_AssignChannelToTimer( uint8_t whichChannel, WhichTimer_t whichTimer );
```

Parameters:

`uint8_t whichChannel`
: A number between 1 and 5 indicating the PWM channel; that you would like to assign to the requested timer.

`WhichTimer_t whichTimer`
: Either `_Timer2_` or `_Timer3_` to indicate which time-base the specified channel should use..

Returns:

`bool`
: `True = Success; False = Either whichChannel or whichTimer outside the legal range.`

Description:

Sets up the internal data structures and OCx configuration bits to use either Timer2 or Timer3 for the selected OCx channel.

Note:

If used on a running channel whose output is enabled, it will produce a glitch output. Should be used before outputs are enabled

Usage:

```
PWMSetup_AssignChannelToTimer(1, _Timer2_); // please excuse the magic number
```

This call would assign Timer2 as the time-base for OC1.

Initialization: Set Period of a Timer

Function:

```
bool PWMSetup_SetPeriodOnTimer( uint16_t reqPeriod, WhichTimer_t WhichTimer );
```

Parameters:

`uin16_t reqPeriod`
: The requested period of the timer, specified in a number of 0.4µs ticks. Minimum value: 100, Maximum value: 65535

`WhichTimer_t whichTimer`
: Either `_Timer2_` or `_Timer3_` to indicate which time-base the specified channel should use..

Returns:

`bool`
: `True = Success; False = Either reqPeriod or whichTimer outside the legal range.`

Description:

Programs the period register of the specified timer to generate a repeating waveform with the requested period.

Usage:

```
PWMSetup_SetPeriodOnTimer(2500, _Timer2_); // please excuse the magic number
```

This call would set the period of Timer2 to 2500 ticks, corresponding to 1ms in the standard SPDL configuration.

Initialization: Set Frequency of a Timer

Function:

```
bool PWMSetup_SetFreqOnTimer( uint16_t reqFreq, WhichTimer_t WhichTimer );
```

Parameters:

`uin16_t reqFreq`
: The requested frequency for the timer, specified in Hz. Minimum value: 38, Maximum value: 25,000

`WhichTimer_t whichTimer`
: Either `_Timer2_` or `_Timer3_` to indicate which time-base the specified channel should use.

Returns:

`bool`
: `True = Success; False = Either reqFreq or whichTimer outside the legal range.`

Description:

Programs the period register of the specified timer to generate a repeating waveform with the requested frequency.

Usage:

```
PWMSetup_SetFreqOnTimer(50, _Timer2_); // please excuse the magic number
```

This call would set the frequency of Timer2 to 50Hz.

Initialization: Mapping a PWM Channel to an Output Pin

Function:

```
bool PWMSetup_MapChannelToOutputPin( uint8_t channel, PWM_PinMap_t whichPin);
```

Parameters:

`Uin8_t channel`
: The requested channel to map (legal values: 1-5)

`WhichPin_t whichPin`
: One of the remapable pin values defined in the `WhichPin_t` enum.

Returns:

`bool`
: `True = Success; False = Either the channel or the pin outside the legal range.`

Description:

Programs the output mapping register of the specified pin to connect the PWM channel to the requested pin.
Legal values for Channel 1: PWM_RPA0, PWM_RPB3, PWM_RPB4, PWM_RPB7, PWM_RPB15.
Legal values for Channel 2: PWM_RPA1, PWM_RPB1, PWM_RPB5, PWM_RPB8, PWM_RPB11.
Legal values for Channel 3: PWM_RPA3, PWM_RPB0, PWM_RPB9, PWM_RPB10, PWM_RPB14.
Legal values for Channel 4: PWM_RPA2, PWM_RPA4, PWM_RPB2, PWM_RPB6, PWM_RPB13.
Legal values for Channel 5: PWM_RPA2, PWM_RPA4, PWM_RPB2, PWM_RPB6, PWM_RPB13.

Note:

If you want to use a pin that is associated only with a particular channel, then you need to activate at least that many channels when calling `PWMSetup_BasicConfig()`. Generally, the call to `PWMSetup_MapChannelToOutputPin()` should be the last step in the setup process as nothing comes out of the PWM system until a pin is assigned. If a PWM channel is not being used, simply skip the call to `PWMSetup_MapChannelToOutputPin() for that pin and the pin can be used as a normal output.`

Usage:

```
PWMSetup_MapChannelToOutputPin(1, PWM_RPA0); // please excuse the magic number
```

This call would set the output of PWM Channel 1 to pin RA0.

Set Duty Cycle

Function:

```
bool PWMOperate_SetDutyOnChannel( uint8_t dutyCycle, uint8_t channel);
```

Parameters:

`uint8_t dutyCycle` a number between 0 & 100 representing the percent duty cycle. 0% & 100% are handled properly.

`uint8_t channel` a number between 1 and 5 to indicate to which OCx channel the duty cycle should be applied.

Returns:

`bool`
`True = Success; False = Channel outside the initialized range or Duty Cycle greater than 100.`

Description:

Sets the duty cycle on the requested channel to the requested value.

Notes:

None

Usage:

```
// magic numbers for channel used solely for the documentation. Do not do this in your code!
PWMOperate_SetDutyOnChannel(50, 1);
```

Sets the duty cycle on Channel 1 to 50%

Set Pulse Width

Function:

```
bool PWMOperate_SetPulseWidthOnChannel( uint16_t NewPW, uint8_t channel);
```

Parameters:

`uint16_t NewPW` a number between 0 & the currently programmed period for this channel specified as a number of 0.4μS ticks.

`uint8_t channel` a number between 1 and 5 to indicate to which channel the pulse width should be applied.

Returns:

`bool`
`True = Success; False = Channel number outside the initialized range or pulse width greater than or equal to the period configured for that channel.`

Description:

Sets the pulse width (high time) on the requested channel to the requested value.

Usage:

```
// magic numbers for channel used solely for the documentation. Do not do this in your code!
PWMOperate_SetPulseWidthOnChannel(2500, 1);
```

Sets the pulse width on Channel 1 to 2500 *.0.4μS = 1mS