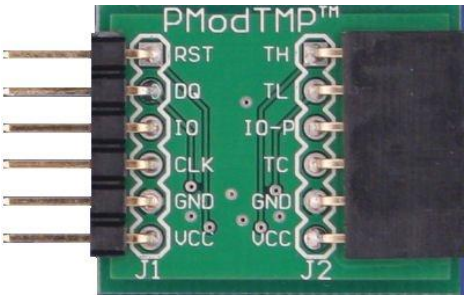
PmodTMP™ Reference Manual

Revision: November 12, 2008
Note: This document applies to REV A of the board.

DIGILENT®
www.digilentinc.com
215 E Main Suite D | Pullman, WA 99163
(509) 334 6306 Voice and Fax

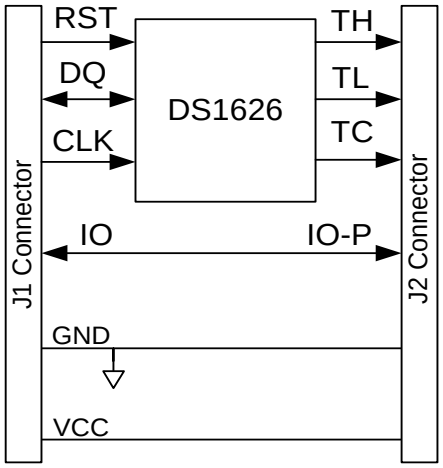
Overview



The PmodTMP is an interface board for the Dallas Semiconductor DS1626 3-wire digital thermometer and thermostat. The DS1626 can be used in projects requiring a precisely measured ambient temperature reading.

Features include:

- a Dallas Semiconductor DS1626 IC
- a 6-pin header and 6-pin connector
- programmable thermostat outputs
- low power consumption
- ±0.5ºC accuracy from 0ºC to +70ºC
- -55ºC to +125 ºC range
- a small form factor (0.80" x 0.80")
- a simple 3-wire interface



PmodTMP Block Diagram

Functional Description

The PmodTMP can be used as either a thermometer with configurable precision or as a programmable standalone thermostat with outputs TH, TL, and TC to indicate if the ambient temperature has exceeded any of the preprogrammed thresholds.

The DS1626 located on the board is equipped with a programmable EEPROM that stores the device's configuration settings. Data transmission between the PmodTMP and the host board is initiated and driven by the host using the 3-wire interface described in the DS1626 data sheet.

To successfully use the PmodTMP, it is expected that the reader first consult the DS1626 data sheet available at www.Maxim-IC.com.

In the appendix of this document, there is sample code illustrating how to set up the PmodTMP as a thermometer that reports the temperature in degrees Fahrenheit. This code can be copied into a file called PmodTMPdriver.c and used in projects using one of Digilent's embedded control boards. The code was written for the Digilent Minicon but can easily be adapted for other boards. Please see the *Remote Temperature Sensing Reference Design* for more information.

The PmodTMP is designed to work with either Digilent programmable logic system boards or embedded control system boards. Most Digilent system boards, such as the Nexys, Basys, or

Cerebot II, have 6-pin connectors that allow the PmodTMP to plug directly into the system board or to connect via a Digilent 6-wire cable.

Some older Digilent boards may need a Digilent Module Interface Board (MIB) and a 6-pin cable to connect to the PmodTMP. The MIB plugs into the system board and the cable connects the MIB to the PmodTMP.

See Table 1 for a description of the signals on interface connectors J1 and J2. For more information, refer to the PmodTMP schematic available at www.digilentinc.com.

The PmodTMP is usually powered from the Digilent system board connected to it. The power and ground connections are on pins five and six of the digital interface connector J1. Alternatively, the PmodTMP can be powered from an external power supply provided through pins five and six of the thermostat interface connector J2. In this case the power select jumper on the system board should be set to “disconnect power” from the system board to J1. Damage may result if two power supplies are connected at the same time.

The Digilent convention is to provide 3.3V to power Pmods. The PmodTMP can be operated at any power supply voltage between 2.7V and 5.5V, but use caution when using any voltage greater than 3.3V, as damage to the Digilent system board could result. For more information refer to reference manuals or schematics for the system board available at www.digilentinc.com.

For detailed information about the Dallas Semiconductor DS1626 data sheet available at www.maxim-ic.com.

Table 1 ***Interface Connector Signal Descriptions***

J1	**Digital Interface**
1	RST (Reset Input)
2	DQ (Data In/Out)
3	IO (Pass Through)
4	CLK (Clock Input)
5	GND
6	VCC
J2	**Thermostat Interface**
1	TH (T_H Output Trip)
2	TL (T_L Output Trip)
3	IO-P (Pass Through)
4	TC (T_{COM} Output Signal)
5	GND
6	VCC

Appendix

```
/************************************************************************/
/*                                                                      */
/*    PmodTMPdriver.c --    Contains functions used to drive the PmodTMP.  */
/*                                                                      */
/************************************************************************/
/*    Author:     Mark Taylor                                           */
/*    Copyright 2008, Digilent Inc.                                     */
/************************************************************************/
/*  Revision History:                                                   */
/*                                                                      */
/*    03/20/2008(MarkT): created                                        */
/*                                                                      */
/************************************************************************/

/* ------------------------------------------------------------------------
-------- */
/*                      Include File Definitions                */
/* ------------------------------------------------------------------------
-------- */

#include <avr/io.h>

#include "stdtypes.h"
#include "minicon.h"

/* ------------------------------------------------------------------------
-------- */
/*                      Local Type Definitions                */
/* ------------------------------------------------------------------------
-------- */

/*    Pin out declarations for the PmodTMP.
      These should be editted to be compatible with the DDRx, PORTx,
      and PINx registers that you are using.  The listed definitions here
      are compatible with the Digilent Minicon using port C of the Minicon.    */
#define ddrPmodTMPrst ddrJC1
#define ddrPmodTMPdq ddrJC2
#define ddrPmodTMPclk ddrJC4
#define prtPmodTMPrst prtJC1
#define prtPmodTMPdq      prtJC2
#define prtPmodTMPclk prtJC4
#define pinPmodTMPrst pinJC1
#define pinPmodTMPdq pinJC2
#define pinPmodTMPclk pinJC4
#define bnPmodTMPrst      bnJC1
#define bnPmodTMPdq       bnJC2
#define bnPmodTMPclk      bnJC4

/*    Dallas Semi. DS1626/DS1726 command set.  This
      command set is explained in detail in the Dallas data sheet. */
#define cmdStartConvertT          0x51
#define cmdStopConvertT           0x22
#define cmdReadTemperature 0xAA
#define cmdReadTH                 0xA1
```

```
#define cmdReadTL                   0xA2
#define cmdWriteTH                  0x01
#define cmdWriteTL                  0x02
#define cmdReadConfig       0xAC
#define cmdWriteConfig      0x0C
#define cmdPOR                      0x54

/*      Desired Configuration Register setup.
        See the Dallas Semi DS1626/DS1726 data sheet for
        additional information about the Configuration Register.*/
#define bConfigData                 0b00001111

/*      Bit position in the Configuration Register of the DONE bit */
#define posDONE                     7


/* ------------------------------------------------------------------------ */
/*                      Forward Declarations                                */
/* ------------------------------------------------------------------------ */


/* Wait_ms() is defined in my main.c file.  Cut the following and
place it in your main.c file:

void Wait_ms(WORD delay) {
        WORD i;

        while(delay > 0){

                for( i = 0; i < 390; i ++){  //i < 390 assumes a clock of 8MHz
                        ;;
                }
                delay -= 1;
        }
}
*/

//tells the AVR Studio compiler that this function is located elsewhere
extern void Wait_ms(WORD ms);

//These are the functions used to drive the PmodTMP
void        InitPmodTMP(void);
WORDGetTemperature(void);




/* ------------------------------------------------------------------------ */
/*                      Procedure Definitions                               */
```

```
/* ------------------------------------------------------------------------
-------- */


/* ------------------------------------------------------------------------
-------- */
/***  InitPmodTMP
**
**    Synopsis:
**          InitPmodTMP()
**
**    Parameters:
**          none
**
**    Return Values:
**          none
**
**    Errors:
**          none
**
**    Description:
**          This routine sets up the PmodTMP to simply report the ambient
**          temperature in 1Shot Mode using the DS1626 with 12-bit precision.
**
**          DS1626 can not be clocked any faster than 1.75 MHz.  Device needs
**          100ns between rising RST edge and falling CLK edge.  Device needs
**          a minimum of 40ns to read the DQ line after the rising edge of CLK.
**          Device needs minimum of 35ns before clock's rising edge to set up
**          DQ line.
**
**          We will simply put data and commands on the DQ line on the falling **
           edge of the clock and read data on the DQ line on the rising edge.
**
**          In this routine, we will set up the DS1626 as follows:
**                R1          = 1:  see R0
**                R0          = 1:  12-bit conversion mode (may take up to 750 **
                                      ms to read!)
**                CPU   = 1:  Stand-alone mode is disabled
**                1SHOT       = 1:  One-Shot Mode.  The cmdStartConvertT
**                                  command initiates a single temperature
**                                  conversion and then the device goes into a
**                                  low-power standby state.
**
*/

void InitPmodTMP() {

BYTE i;

/*    Set up pins that connect to the Digilent PmodTMP board.
      This code should be moved to the function where you set up all of
      your boards IO pins.*/
      prtPmodTMPrst    &= ~(1 << bnPmodTMPrst);  //LLV
      prtPmodTMPdq     &= ~(1 << bnPmodTMPdq);
      prtPmodTMPclk    &= ~(1 << bnPmodTMPclk);

      ddrPmodTMPrst    |= (1 << bnPmodTMPrst);  //initially all outputs
```

```
	ddrPmodTMPdq	|= (1 << bnPmodTMPdq);
	ddrPmodTMPclk	|= (1 << bnPmodTMPclk);


	/* bring RST line high to enable PmodTMP */
	prtPmodTMPrst |= (1 << bnPmodTMPrst);

/*	Start clocking out write config. command */
	for(i = 0; i < 8; i ++) {
		//lower clock
		prtPmodTMPclk &= ~(1 << bnPmodTMPclk); //falling edge of clock

		//set prtPmodTMPdq based on state of data bit in position i
		if(cmdWriteConfig & (1 << i)) {
			prtPmodTMPdq |= (1 << bnPmodTMPdq); //set dq if data bit set
		}
		else {
			prtPmodTMPdq &= ~(1 << bnPmodTMPdq); //else clear dq
		}

		prtPmodTMPclk |= (1 << bnPmodTMPclk); //rising edge of clock
	}

/*	Start writing Configuration Reg. Data */
	for(i = 0; i < 8; i ++) {
		//lower clock
		prtPmodTMPclk &= ~(1 << bnPmodTMPclk); //falling edge of clock

		//set prtPmodTMPdq based on state of data bit in position i
		if(bConfigData & (1 << i)) {
			prtPmodTMPdq |= (1 << bnPmodTMPdq); //set dq if data bit set
		}
		else {
			prtPmodTMPdq &= ~(1 << bnPmodTMPdq); //else clear dq
		}

		prtPmodTMPclk |= (1 << bnPmodTMPclk); //rising edge of clock
	}

	//make sure DQ pin is tied to ground
	prtPmodTMPdq &= ~(1 << bnPmodTMPdq); //LLV

	//disable device by returning RST line low
	prtPmodTMPrst &= ~(1 << bnPmodTMPrst);
}
```

```
/* ------------------------------------------------------------------------
-------- */
/***	GetTemperature
**
**	Synopsis:
**		WORD GetTemperature(void)
**
**	Parameters:
**		none
**
**	Return Values:
**		wTemperature
**
**	Errors:
**		none
**
**	Description:
**		This routine requests that the PmodTMP takes a temperature
**		reading.  It then waits, polling the DONE bit in the Configuration
**		register until the conversion is complete.  It then clocks out the
**		temperature data and returns it.
**
*/

WORD GetTemperature() {

	BYTE i;
	BYTE bRegister;
	WORD wTemperature;

	bRegister = 0;
	wTemperature = 0;


	/* bring RST line high to enable PmodTMP */
	prtPmodTMPrst |= (1 << bnPmodTMPrst);


/*	Start clocking out StartConvertT command */
	for(i = 0; i < 8; i ++) {
		//lower clock
		prtPmodTMPclk &= ~(1 << bnPmodTMPclk); //falling edge of clock

		//set prtPmodTMPdq based on state of data bit in position i
		if(cmdStartConvertT & (1 << i)) {
			prtPmodTMPdq |= (1 << bnPmodTMPdq); //set dq if data bit set
		}
		else {
			prtPmodTMPdq &= ~(1 << bnPmodTMPdq); //else clear dq
		}
		prtPmodTMPclk |= (1 << bnPmodTMPclk); //rising edge of clock
	}
```

```
	/*	End command and disable device by returning RST line low.  You must
		end each command/read with the device by toggling the RST line.	*/
	prtPmodTMPrst &= ~(1 << bnPmodTMPrst);


/*	Poll DONE bit in Configuration Register until the conversion is complete. */
	do {

		/* Toggle Led1 to indicate we are waiting for conversion to complete.
			This LED is located on the Minicon.  Remove this line if you do
			not want visual indicator of the conversion process or if you do
			not have a spare LED on your board. */
		prtLed1 ^= (1 << bnLed1);

		/* Do the following every 10ms.
			The following function is defined in my main.c file and is a
			simple while loop designed to take a WORD input and loop for the
			specified number of milliseconds before returning. */
		Wait_ms(10);

		/* bring RST line high to enable PmodTMP */
		prtPmodTMPrst |= (1 << bnPmodTMPrst);


	/* Clock out command to read Configuration Register */
		for(i = 0; i < 8; i ++) {
			//lower clock
			prtPmodTMPclk &= ~(1 << bnPmodTMPclk); //falling edge

			//set prtPmodTMPdq based on state of data bit in position i
			if(cmdReadConfig & (1 << i)) {
				prtPmodTMPdq |= (1 << bnPmodTMPdq);
			}
			else {
				prtPmodTMPdq &= ~(1 << bnPmodTMPdq);
		}

			prtPmodTMPclk |= (1 << bnPmodTMPclk); //rising edge
		}


		// Set DQ pin to read response of device
		prtPmodTMPdq &= ~(1 << bnPmodTMPdq); //no pull-ups
		ddrPmodTMPdq &= ~(1 << bnPmodTMPdq); //input


	/* Clock in Configuration Register data */
		for(i = 0; i < 8; i ++) {
			//lower clock
			prtPmodTMPclk &= ~(1 << bnPmodTMPclk); //falling edge

			/* Ensure device is given time to write bit before rising edge.
				Depending on the speed of your clock, these assembly "nop"
				commands  may be totally unnecessary.  Check the
				DS1626/DS1726 data sheet for the minimum rise/fall times
				for the device. */
			asm volatile("nop");
```

```
                asm volatile("nop");

                prtPmodTMPclk |= (1 << bnPmodTMPclk); //rising edge

                if( pinPmodTMPdq & (1 << bnPmodTMPdq) ) { //if data bit high
                        bRegister |= (1 << i); //set bit in variable
                }
                else {
                        bRegister &= ~(1 << i); //otherwise clear bit in variable
                }

        }

        //end command and disable device by returning RST line low
        prtPmodTMPrst &= ~(1 << bnPmodTMPrst);

        ddrPmodTMPdq |= (1 << bnPmodTMPdq); //output again

} while ( ! (bRegister & (1 << posDONE) ) ); //while DONE bit is not set


/* Turn off Led1 to indicate we are done waiting for conversion to complete.
        Remove this line if you are not using an LED as a visual indicator of
conversion time */
prtLed1 &= ~(1 << bnLed1);


 //give it a bit of time before issuing next command
Wait_ms(2);


/* bring RST line high to enable PmodTMP */
prtPmodTMPrst |= (1 << bnPmodTMPrst);

/* Clock out command to read Temperature Register */
for(i = 0; i < 8; i ++) {
        //lower clock
        prtPmodTMPclk &= ~(1 << bnPmodTMPclk); //falling edge of clock

        //set prtPmodTMPdq based on state of data bit in position i
        if(cmdReadTemperature & (1 << i)) {
                prtPmodTMPdq |= (1 << bnPmodTMPdq); //set dq if data bit set
        }
        else {
                prtPmodTMPdq &= ~(1 << bnPmodTMPdq); //else clear dq
        }

        prtPmodTMPclk |= (1 << bnPmodTMPclk); //rising edge of clock
}


// Set DQ pin to read output of device
prtPmodTMPdq &= ~(1 << bnPmodTMPdq); //no pull-ups
ddrPmodTMPdq &= ~(1 << bnPmodTMPdq); //input
```

```
/* Clock in Temperature Register data */
	for(i = 0; i < 12; i ++) { //Temperature data is 12-bits long
		//lower clock
		prtPmodTMPclk &= ~(1 << bnPmodTMPclk); //falling edge of clock

		//Ensure device is given time to write bit before rising edge
		asm volatile("nop");
		asm volatile("nop");

		prtPmodTMPclk |= (1 << bnPmodTMPclk); //rising edge of clock

		if( pinPmodTMPdq & (1 << bnPmodTMPdq) ) { //if data bit high
			wTemperature |= (1 << i); //set bit in variable
		}
	}

	//end command and disable device by returning RST line low
	prtPmodTMPrst &= ~(1 << bnPmodTMPrst);

	ddrPmodTMPdq |= (1 << bnPmodTMPdq); //output again

/*	wTemperature currently reflects the contents of the device's Temperature
	Register.  This notation includes binary decimal units of type Celsius.
	In my application, I do not need the additional bits beyond the
	decimal point, so we will shift them out.  Then convert to Fahrenheit. */

	//shift out bits 0 -> 3. Bit 4 becomes LSB.
	wTemperature = (wTemperature >> 4);
	//convert to Fahrenheit
	wTemperature = ( ( (wTemperature * 9) / 5) + 32 ); //classic equation

	return wTemperature;
}
```