



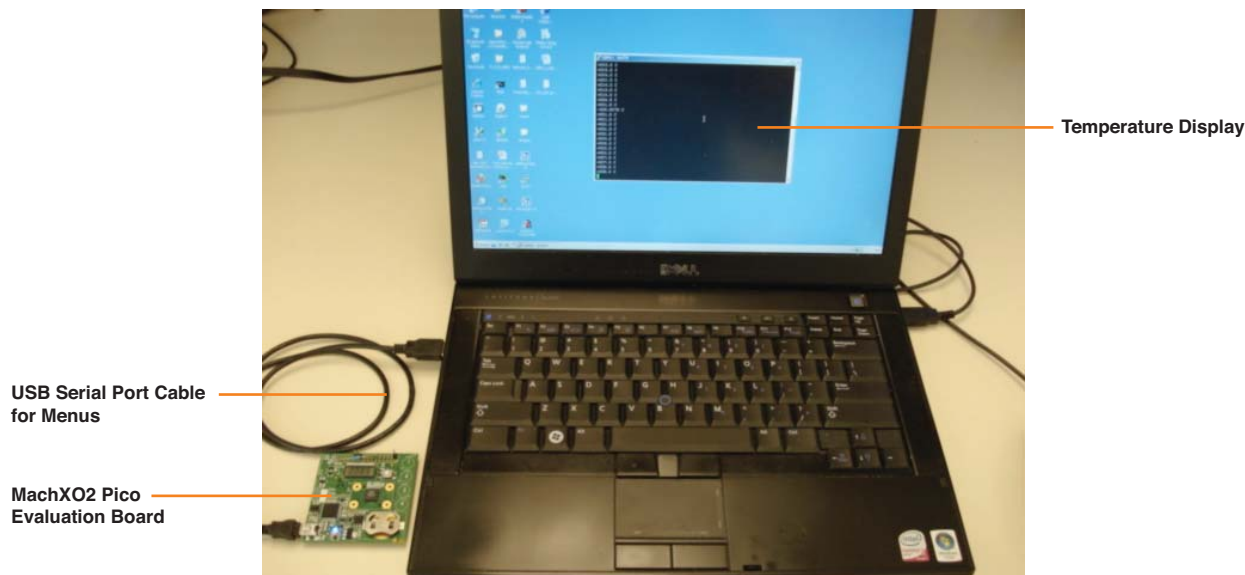
MachXO2 Master SPI/I²C Demo Using 'C' and LatticeMico8

User's Guide

Introduction

The MachXO2™ LatticeMico8™ Temperature Logging Demo uses a MachXO2 Pico Evaluation Board to read the temperature and display it on the LCD screen and also in a terminal window on a PC. The temperature is obtained from an I²C temperature sensor device on the evaluation board. The temperature readings can also be logged into SPI Flash memory for later recall.

Figure 1. Demonstration Setup



The purpose of the demo is to:

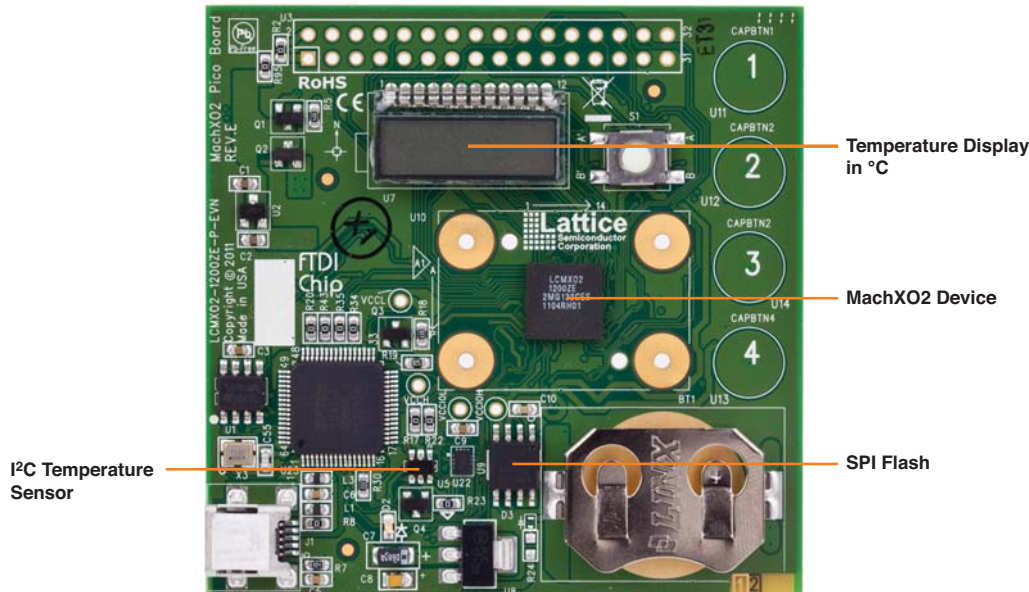
1. Demonstrate use of the MachXO2 Embedded Function Block (EFB) I²C and SPI Master controllers
2. Demonstrate I²C and SPI drivers written in 'C'
3. Demonstrate LatticeMico8 application software written in 'C'

The serial port from the MachXO2 Pico Evaluation Board provides the user interface. The RS232 serial port is implemented using a USB connection. A mini-USB cable connects the evaluation board to a PC. The PC runs a USB to virtual COM port program that allows the HyperTerminal to connect to the board over the USB cable.

The LatticeMico8 soft microcontroller in the MachXO2 device provides the menu software and temperature acquisition and display routines. To see the temperature change, place a finger on the temperature sensor chip. Note that the temperature reading measures the board temperature and not the ambient air temperature, so do not expect the reported temperature to match the room temperature.

Note: This demo assumes the reader is familiar with Lattice Mico8 projects and 'C' programming. Please refer to the Lattice Mico System documentation for information regarding importing platforms, 'C' projects, compiling, deploying, etc.

Figure 2. MachXO2 Pico Evaluation Board Components



Using menu options, the temperature data can be logged to the SPI Flash device. Up to 32,000 temperature samples can be recorded. The readings can then be recalled and displayed.

The temperature display on the LCD is in degrees C (Celsius), with ½ degree resolution for positive temperature readings. When displaying negative temperatures, a leading minus sign is added, and the fractional digit is no longer present. Temperature display on the serial port terminal is always full resolution.

Setup

This section describes how to connect and set up the demo.

General Warnings

Please observe the following important safety items:

- Follow ESD precautions.
- Do not expose to extreme temperatures in an attempt to get full range temperature +/- 128 °C readings. Operate within the commercial temperature range.

Required Demo Components

- MachXO2 Pico Evaluation Board
- Mini USB cable to power MachXO2 and provide terminal connection
- Terminal emulation software (HyperTerminal or PuTTY)

Project Setup

Unzip the demo design package.

The demo bitstream that needs to be loaded into the evaluation board is located in the Bitstreams directory.

Start Diamond Programmer to download the bitstream. Use the XO2_Pico.xcf programmer configuration file to load the bitstream into the MachXO2 device. *Note: You may need to browse to locate the JEDEC file as Diamond Programmer only stores absolute paths.*

MachXO2 Pico Evaluation Board Installation

Connect the USB cable to the PC for power, bitstream programming and serial port menu access.

Download the demo bitstream into the MachXO2 device (if first time).

After successful programming, temperature readings should be displayed on the LCD indicating the design is functioning.

FTDI Communications Port Driver Installation

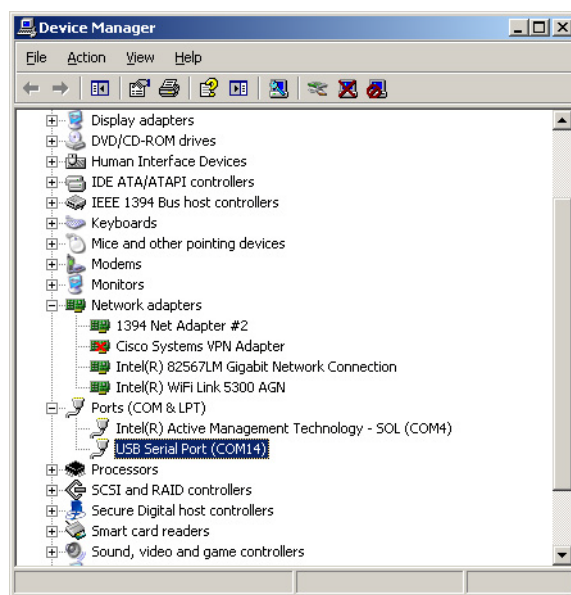
The next step is to set up the serial port that will provide the connection to the menus on the MachXO2 Pico Evaluation Board. You may need to install a driver provided by FTDI (the manufacturer of the FT232RL device used on the evaluation board that converts the RS-232 UART output to USB). The driver converts the PC USB port to a communications port so that a serial terminal program (HyperTerminal) can open the USB port and communicate over it as if it were an RS-232 port. The FTDI driver installation program can be downloaded from www.ftdichip.com/Drivers/VCP.htm.

Note: Windows Vista and Windows 7 no longer ship with HyperTerminal. These users will need to install a suitable serial port terminal emulator like PuTTY 0.60 (or newer) or the XP version of HyperTerminal.

Once the driver is installed, power up the MachXO2 Pico Evaluation Board by connecting the USB cable.

To determine which communications port the USB serial device is mapped to, open **My Computer > Properties > Hardware > Device Manager**. Select **Ports** to see available serial ports and choose the one that is the USB Serial Port. See Figure 3 as an example.

Figure 3. Locate New COM Port



Open HyperTerminal or PuTTY and select the new USB Serial Port. Set the communication settings to: 115000 baud, 8 data bits, No Parity, 1 stop bit (115000,8,N,1). Figures 5 and 6 show these settings for a PuTTY session.

Figure 4. Select COM Port in Terminal Emulator

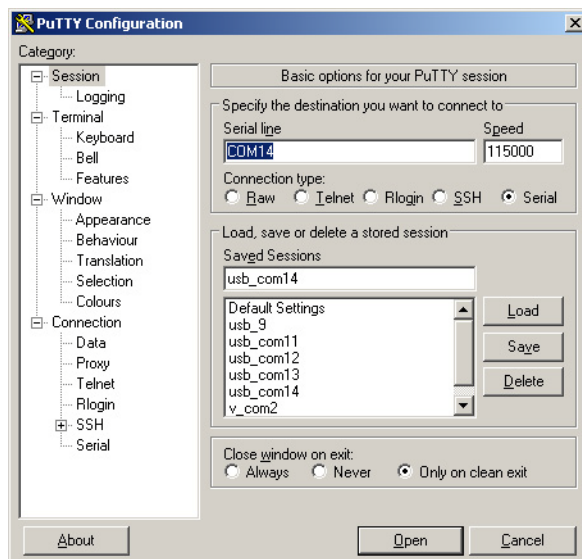
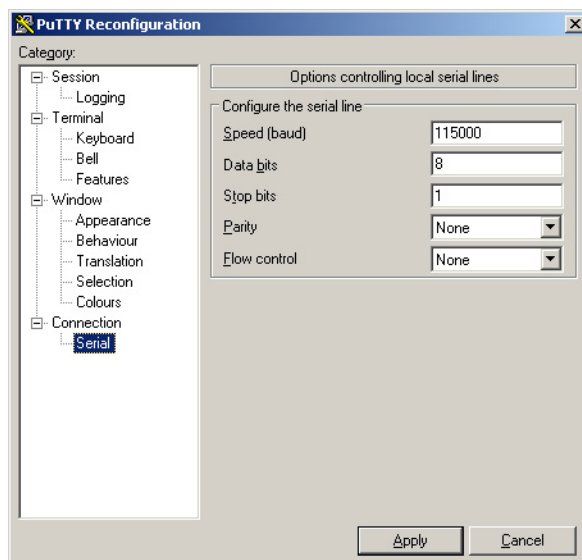


Figure 5. Serial Port Settings



After the serial port settings are configured and power is applied to the evaluation board, the current temperature in Celsius will be reported. Use any key to access the menu.

Hardware Settings

For demo operation, there is no hardware configuration necessary, other than connecting the USB cable. This section is provided strictly for reference, or in cases where the board has been used in other applications and the current configuration is in doubt.

MachXO2 Pico Evaluation Board

The LCD is used to indicate the current temperature. There are no other indicators, such as blinking LEDs, or switches or jumpers for control.

MachXO2 output pin EnI2CSPI is set to '1' in the top level so that power is applied to the I²C and SPI devices.

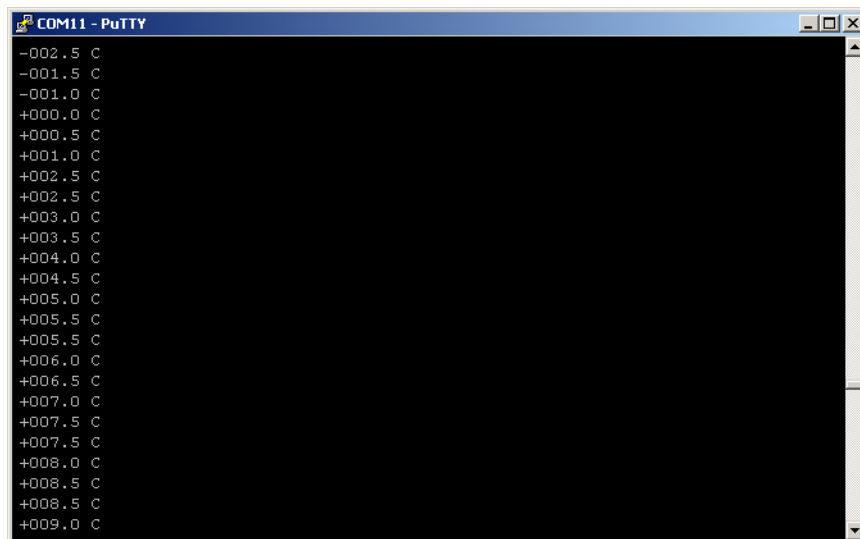
Demo Operation

All user interaction and demo control is handled through the text menus provided by the LatticeMico8 soft micro-controller. This section discusses the available menu options that demonstrate the MachXO2 EFB I²C and SPI accesses to the Temperature Sensor and SPI Flash devices.

Initial Temperature Display

At power-up the demo begins reading and displaying the current temperature twice a second. The temperature is displayed on the LCD and in the terminal window. Pressing any key will exit Temperature Display mode and enter the Main Menu.

Figure 6. Example Temperature Display

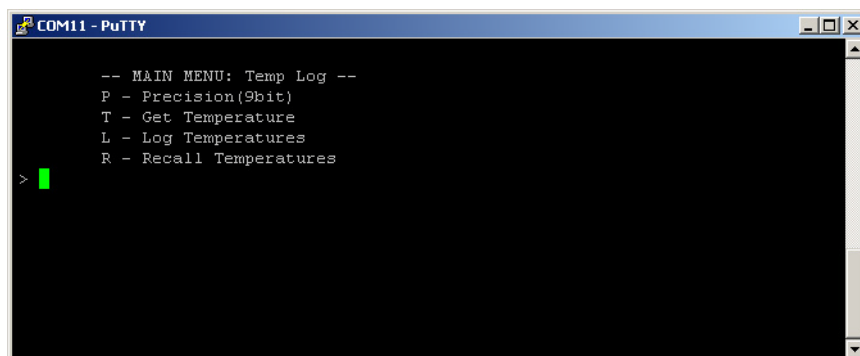


```
COM11 - PuTTY
-002.5 C
-001.5 C
-001.0 C
+000.0 C
+000.5 C
+001.0 C
+002.5 C
+002.5 C
+003.0 C
+003.5 C
+004.0 C
+004.5 C
+005.0 C
+005.5 C
+005.5 C
+006.0 C
+006.5 C
+007.0 C
+007.5 C
+007.5 C
+008.0 C
+008.5 C
+008.5 C
+009.0 C
```

Menu Screen

The Menu screen allows the user to select the temperature display resolution and save and recall temperature data to the SPI Flash. Simply enter the letter of the operation to execute.

Figure 7. Main Menu



```
COM11 - PuTTY
-- MAIN MENU: Temp Log --
P - Precision(9bit)
T - Get Temperature
L - Log Temperatures
R - Recall Temperatures
> █
```

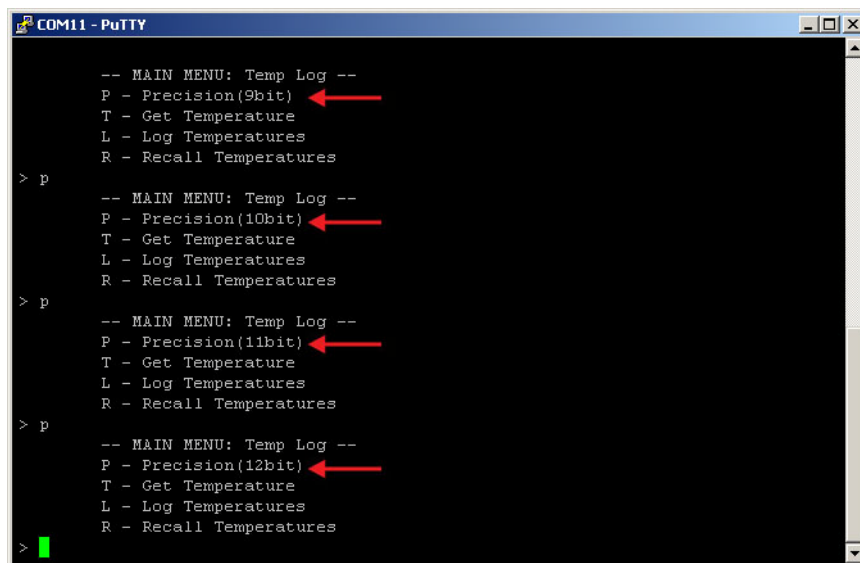
Precision

The I²C Temperature Sensor can be configured to report the temperature in 9, 10, 11 or 12 bits of accuracy. This menu option allows changing the precision of the reported temperature. This will affect the fractional part of the temperature reading. The whole number portion remains unchanged, and is still 7 bits of range plus a sign bit (+/- 127 °C).

- For 9-bit precision, a single bit is used for the fractional part to indicate 1/2 degree resolution
- For 10-bit precision, two bits are used for the fractional part to indicate 1/4 degree resolution
- For 11-bit precision, three bits are used for the fractional part to indicate 1/8 degree resolution
- For 12 bit precision, four bits are used for the fractional part to indicate 1/16 degree resolution

Pressing the **P** key toggles through the available precision settings

Figure 8. Temperature Resolution Selections



```

COM11 - PuTTY

-- MAIN MENU: Temp Log --
P - Precision(9bit)
T - Get Temperature
L - Log Temperatures
R - Recall Temperatures
> p

-- MAIN MENU: Temp Log --
P - Precision(10bit)
T - Get Temperature
L - Log Temperatures
R - Recall Temperatures
> p

-- MAIN MENU: Temp Log --
P - Precision(11bit)
T - Get Temperature
L - Log Temperatures
R - Recall Temperatures
> p

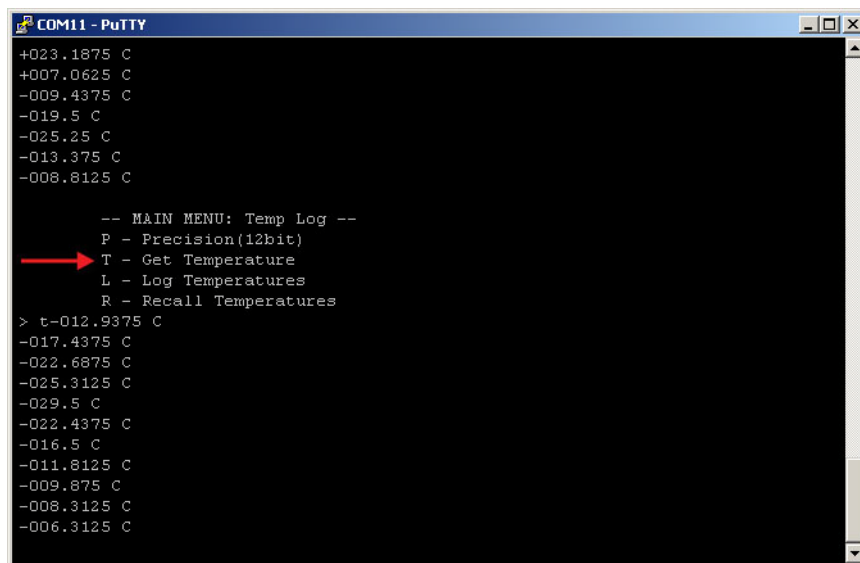
-- MAIN MENU: Temp Log --
P - Precision(12bit)
T - Get Temperature
L - Log Temperatures
R - Recall Temperatures
>
  
```

Note that while the terminal output might show temperature readings in 1/16th degree resolution, the LCD screen only shows 0.5 degree resolution to make the LCD software display routines simpler. The LCD also drops the fractional portion when the temperature becomes negative to make room for the '-' (minus sign).

Temperature Display

To return to the real-time display of the temperature data, press the **T** key. This will cause the software to read the I²C temperature sensor, convert the value for display, pause for 0.5 seconds and check for a key press. If the user does not press a key then the temperature acquisition, conversion and display will continue two times per second.

Figure 9. Return to Temperature Display



```
COM11 - PuTTY
+023.1875 C
+007.0625 C
-009.4375 C
-019.5 C
-025.25 C
-013.375 C
-008.8125 C

-- MAIN MENU: Temp Log --
P - Precision(12bit)
T - Get Temperature
L - Log Temperatures
R - Recall Temperatures
> t-012.9375 C
-017.4375 C
-022.6875 C
-025.3125 C
-029.5 C
-022.4375 C
-016.5 C
-011.8125 C
-009.875 C
-008.3125 C
-006.3125 C
```

Log Temperature Readings to SPI Flash

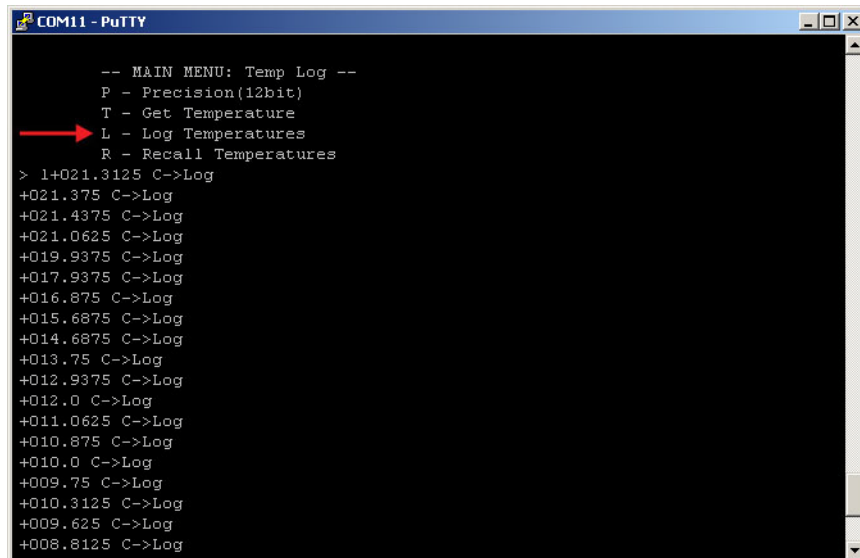
The 'L' selection enables logging the temperature data to the SPI Flash. Pressing the L key will return to the Temperature Display mode and will indicate that each reading is being written to the SPI Flash by the '->Log' indicator on the terminal screen, and the display of the ":" (colon) character on the LCD.

Press the L key again to turn off logging.

Each time logging is enabled, the Flash sector is erased in preparation for storage of the new readings. All previously stored data will be lost. This means that you cannot start logging, pause logging (press 'L' to turn it off), then resume logging at a later time.

A maximum of 64 kB of data can be logged. This is approximately 16,000 seconds (4.5 hours). Logging is only done to the first sector in the SPI Flash. Upon filling the sector, the logging will still indicate it is active but not write any more data into the SPI Flash. No indication is given that the address has reached its maximum.

Figure 10. Logging Temperature to SPI Flash



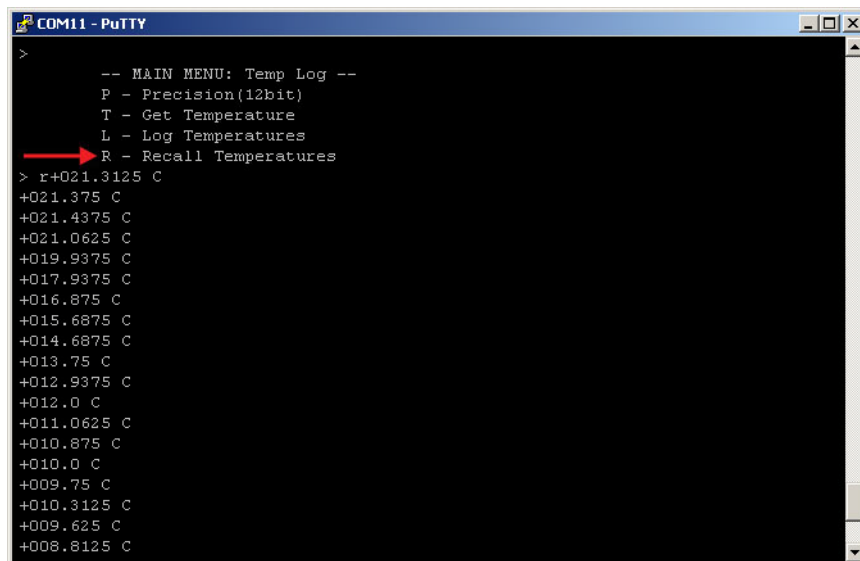
```
COM11 - PuTTY

-- MAIN MENU: Temp Log --
P - Precision(12bit)
T - Get Temperature
L - Log Temperatures
R - Recall Temperatures
> l+021.3125 C->Log
+021.375 C->Log
+021.4375 C->Log
+021.0625 C->Log
+019.9375 C->Log
+017.9375 C->Log
+016.875 C->Log
+015.6875 C->Log
+014.6875 C->Log
+013.75 C->Log
+012.9375 C->Log
+012.0 C->Log
+011.0625 C->Log
+010.875 C->Log
+010.0 C->Log
+009.75 C->Log
+010.3125 C->Log
+009.625 C->Log
+008.8125 C->Log
```


Recall Temperature Data from Log

The 'R' selection reads the temperature data stored in the SPI Flash and displays it in the precision chosen during the logging. This example shows a read-back of the temperature values that were previously stored in the above logging example.

Figure 11. Recall Temperature Data



```
>
-- MAIN MENU: Temp Log --
P - Precision(12bit)
T - Get Temperature
L - Log Temperatures
R - Recall Temperatures
> r+021.3125 C
+021.375 C
+021.4375 C
+021.0625 C
+019.9375 C
+017.9375 C
+016.875 C
+015.6875 C
+014.6875 C
+013.75 C
+012.9375 C
+012.0 C
+011.0625 C
+010.875 C
+010.0 C
+009.75 C
+010.3125 C
+009.625 C
+008.8125 C
```

Changing the precision after data values are logged will not change the recalled data display format. The reason is that the raw 2-byte temperature data is stored in the SPI Flash. If it is stored with 12-bit precision then it will be displayed with 12-bit precision because the three least significant bits have data. If it is stored with 9-bit precision, it cannot be displayed with 12-bit accuracy because the information is not there; those three bits will be zeros.

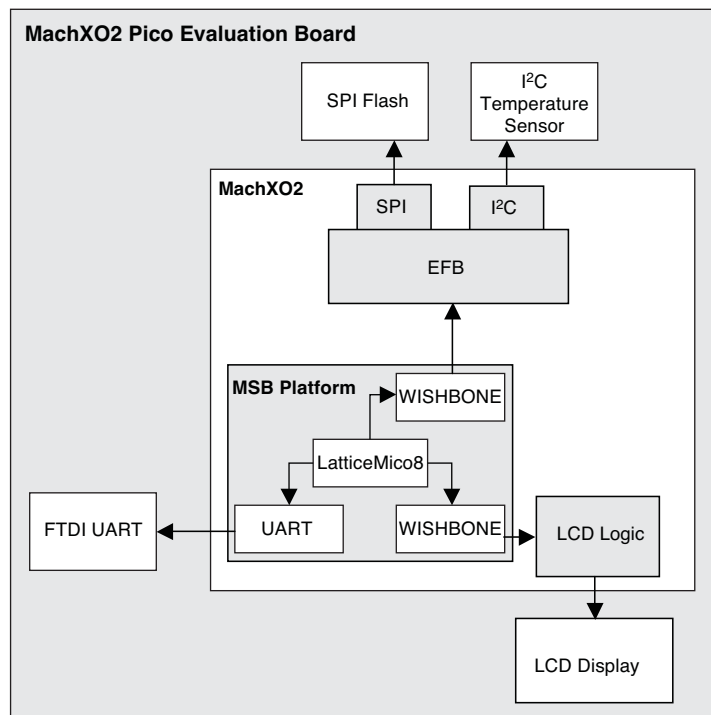
Demo Hardware Architecture

This section discusses the design and implementation of the RTL logic and operation of the external devices used in the demo.

Overview

The LatticeMico8 Mico System Builder processor system runs the demo software from on-chip program memory. The processor system also contains the UART component that provides the serial port communications to the external PC. The hardened I²C master in the EFB provides access to the I²C temperature sensor device (Burr-Brown TMP101). The hardened SPI master in the EFB provides access to the SPI Flash device (Atmel AT25DF041A).

Figure 12. Demo Hardware Diagram

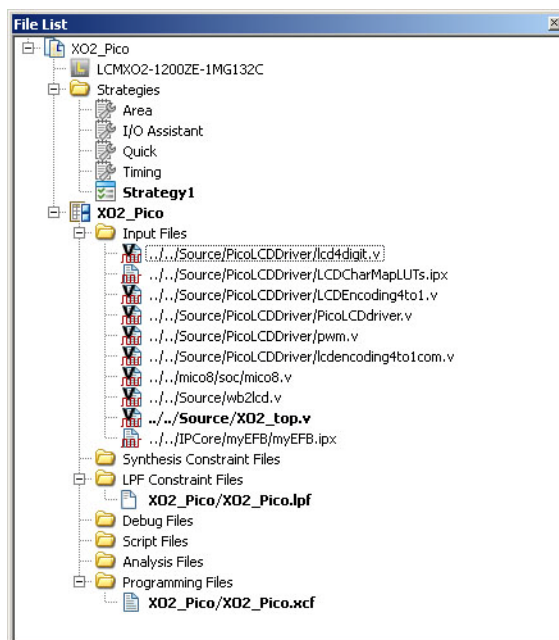


MachXO2 Diamond Project

The MachXO2 project is located in Hardware\Implementation\Diamond_1.4.

The project builds the LatticeMico8 system and wires the EFB I²C master and EFB SPI master to the respective devices on the MachXO2 Pico Evaluation Board. It also drives the LCD display and has Tx/Rx connections to the USB UART.

Figure 13. Diamond Project and Source



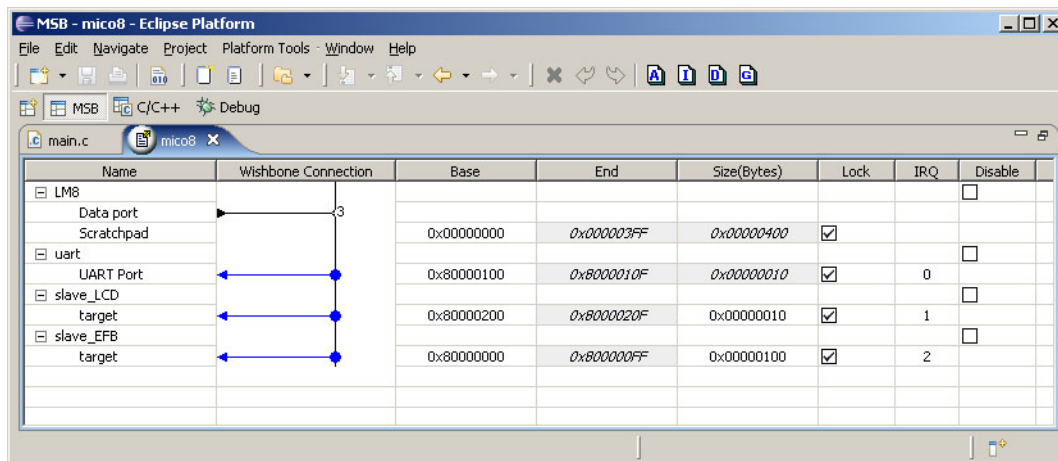
Building the project is straightforward, although the LatticeMico8 Mico System Builder platform needs to be generated first and the program memory initialized with the demo code.

Programming the MachXO2 on the evaluation board is accomplished with the included Programmer .XCF file.

MachXO2 Mico System Builder Platform

The LatticeMico8 platform defines the embedded processor architecture of the system. The Mico System Builder platform is located in Hardware\mico8\soc. Refer to the Readme.txt file in the Workspace folder for instructions on importing the Mico platform and C/C++ projects into the Lattice Mico System tool.

Figure 14. LatticeMico8 Platform



The platform frequency is selected to be 12 MHz, sourced from the clock device that also drives the FTDI device.

LatticeMico8

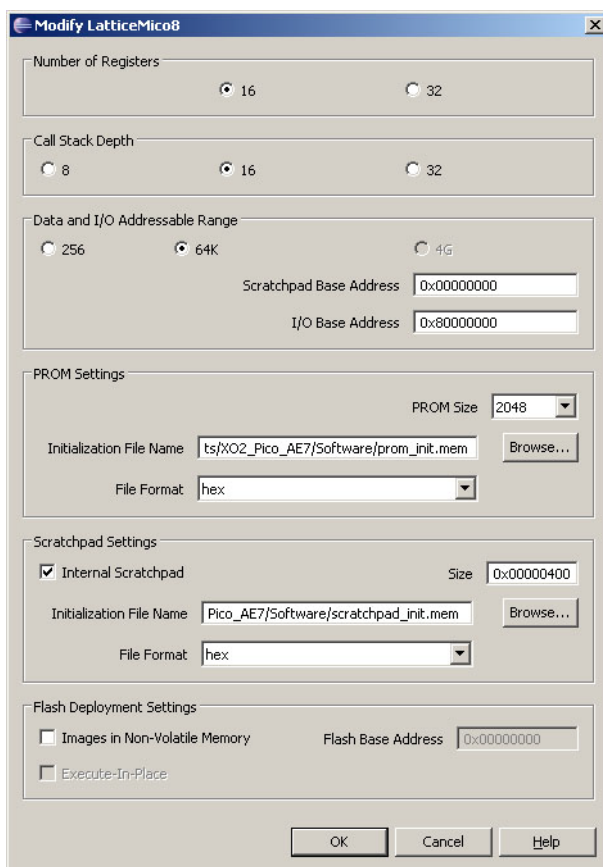
The LatticeMico8 soft microcontroller is configured to have the maximum amount of executable code space available in the MachXO2 device. The code space is available in power-of-2 sizes. A code size of 2KB instructions requires four EBRs. The next size up (4KB instructions) requires eight EBRs, which is greater than the number of EBRs in a MachXO2 device.

The scratch pad size is set at 0x400 bytes (1KB, which is 1 EBR). The size is dictated by the fact that the EFB has a 256-byte address window. In order to include the other components in separate memory mapped windows, the small memory model cannot be used (8-bit addressing), so the medium memory model is used (16-bit addressing – which can cause inefficiencies in the resulting executable code).

No interrupts are used to reduce executable code size. Interrupts require interrupt handlers, interrupt stacks and save/restore context routines. Disabling interrupts eliminates this code with no penalty for this application. Polling is used to service all the external devices (UART, I²C, SPI).

The program and scratch pad initialization files point to the prom_init.mem and scratchpad_init.mem files in the Software directory. These files are created when the C/C++ Software Deployment tool is used to create the binary executable memory image.

Figure 15. LatticeMico8 Configuration



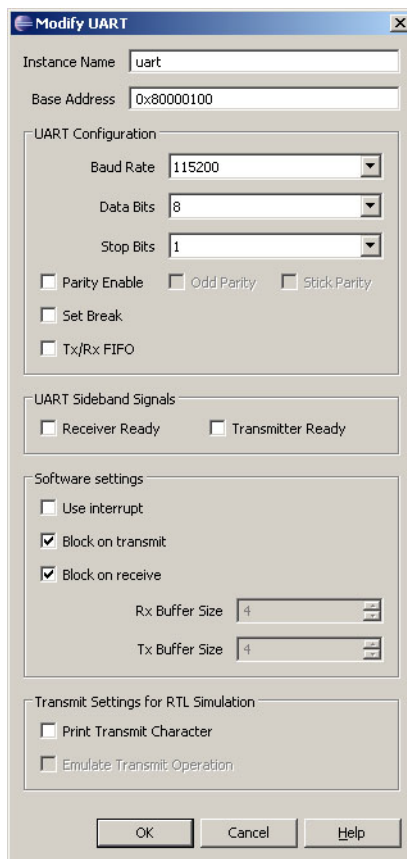
UART

The UART component provides the user menu interface to the terminal emulation program. The UART component also includes the standard LatticeMico8 character drivers, operating in polling mode. These drivers are quite small and attention was not placed on replacing/rewriting these to save more instruction space.

No interrupts are used to reduce executable code size. Interrupts require interrupt handlers, interrupt stacks and save/restore context routines. Disabling interrupts eliminates this code with no penalty for this application. Polling is used to service all the UART Tx/Rx functions.

The UART is set for 115200 baud, 8 data bits, 1 stop bit and no parity or flow control. Ensure that the terminal emulation software matches these settings.

Figure 16. UART Configuration



The 'Modify UART' dialog box contains the following configuration options:

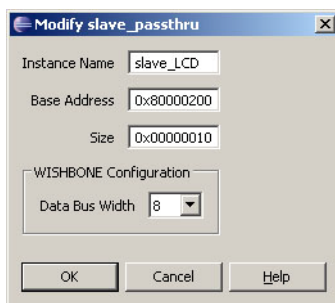
- Instance Name:** uart
- Base Address:** 0x80000100
- UART Configuration:**
 - Baud Rate: 115200
 - Data Bits: 8
 - Stop Bits: 1
 - Parity Enable: ☐ (Odd Parity: ☐ Stick Parity: ☐)
 - Set Break: ☐
 - Tx/Rx FIFO: ☐
- UART Sideband Signals:**
 - Receiver Ready: ☐
 - Transmitter Ready: ☐
- Software settings:**
 - Use interrupt: ☐
 - Block on transmit: ☒
 - Block on receive: ☒
 - Rx Buffer Size: 4
 - Tx Buffer Size: 4
- Transmit Settings for RTL Simulation:**
 - Print Transmit Character: ☐
 - Emulate Transmit Operation: ☐

Buttons: OK, Cancel, Help

LCD Driver Slave Passthru

The LCD driver module that produces the electrical signals to drive the 4-digit LCD display on the MachXO2 Pico Evaluation Board is interfaced to the LatticeMico8 by a slave passthru port. This is a simple way to attach a WISH-BONE compliant slave to a platform without the need to create a LatticeMico8 custom component.

Figure 17. LCD Access Configuration



The 'Modify slave_passthru' dialog box contains the following configuration options:

- Instance Name:** slave_LCD
- Base Address:** 0x80000200
- Size:** 0x00000010
- WISHBONE Configuration:**
 - Data Bus Width: 8

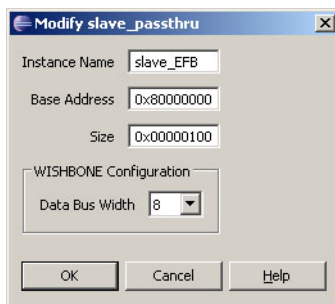
Buttons: OK, Cancel, Help

EFB Slave Passthru

The EFB in the MachXO2 device is interfaced to the LatticeMico8 by a slave passthru port. This is done to avoid importing an EFB component into the platform. The EFB component also brings along the default I²C, SPI and timer drivers that have more features than are needed and inflate the size of the executable, exceeding the 2,048 instruction limit. By creating the EFB using IPexpress™ and interfacing to it through a slave passthru, specialized, optimized I²C and SPI drivers can be written to do the bare minimum needed to accomplish this demo.

Note that the address window is 256 bytes wide because the EFB contains nearly this many registers.

Figure 18. EFB Access Configuration



LCD Driver Module

The LCD driver module top level is PicoLCDdriver.v, located in the \Hardware\Source\PicoLCDDriver\ directory.

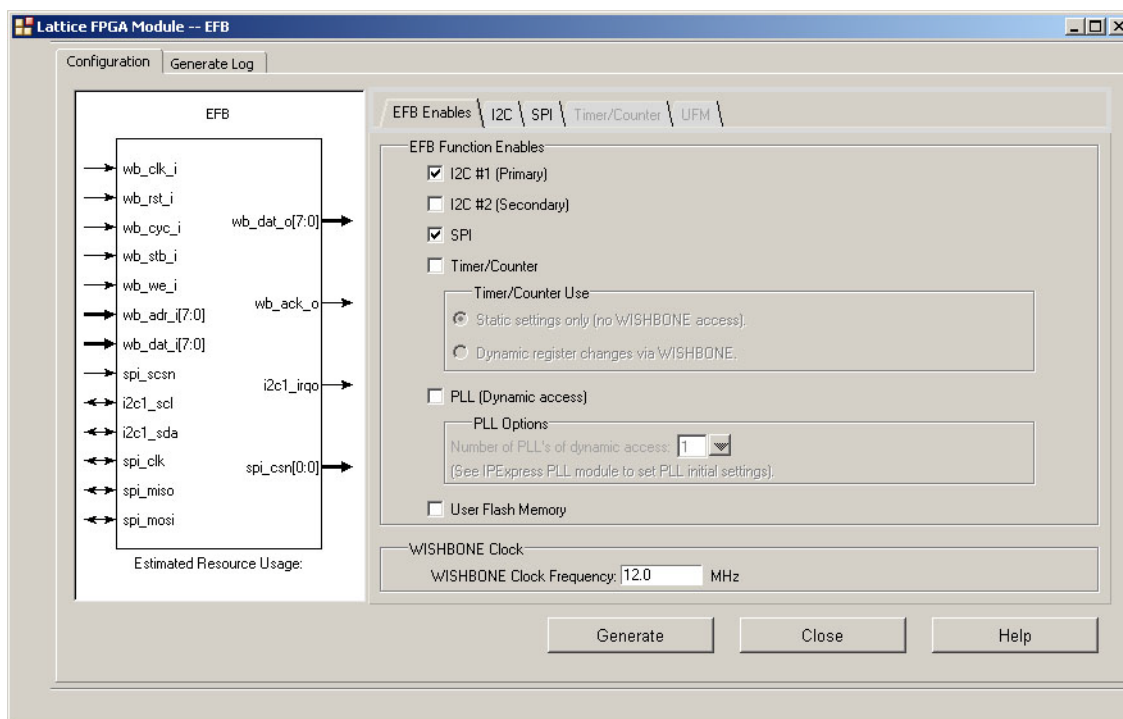
There are two implementations of the character map look-up table. The original method uses an EBR to hold the character map (LCDCharMap.v). To save that EBR for program data space, another approach makes a 64-entry, 8-bit wide memory out of distributed memory (LCDCharMapLUTs.v). This has little impact on the hardware resources used by this project, and frees up the EBR for software data storage.

- **LCDCharMapDecode.txt** – The index key to which characters are displayed for which numeric value written to the index register.
- **PicoLCDdriver.v** – Top level module
- **wb2lcd.v** – Interface between the LatticeMico8 WISHBONE bus and the PicoLCDdriver.v, using the lcd_slave_passthru.

EFB Module

The EFB module is generated with IPexpress. Double-click on the **IPCore/myEFB/myEFB.ipx** file in Diamond to launch IPexpress and examine the EFB settings. The I²C #1 Master is enabled and the SPI is enabled for Master mode.

Figure 19. EFB Configuration



I²C Temperature Sensor

The temperature sensor is a 6-pin Burr-Brown TMP101 I²C slave device. The I²C interface allows reading and writing the registers in the temperature sensor. The configuration register is written to select the resolution of the temperature reading (9,10,11 or 12-bit accuracy). The temperature register is read to obtain the current temperature. Temperature is reported in 2 bytes in the following format (see TMP101 data sheet for more details).

Table 1. Conversion to Temperature

Temperature (°C)	Digital Output (Binary)	Hexadecimal
128	0111 1111 1111	7FF
127.9375	0111 1111 1111	7FF
100	0110 0100 0000	640
80	0101 0000 0000	500
75	0100 1011 0000	4B0
50	0011 0010 0000	320
25	0001 1001 0000	190
0.25	0000 0000 0100	004
0	0000 0000 0000	000
-0.25	1111 1111 1100	FFC
-25	1110 0111 0000	E70
-55	1100 1001 0000	C90
-128	1000 0000 0000	800

The first byte contains the whole number portion of the temperature, with the most significant bit being the sign indicator. The second byte contains the fractional portion of the temperature, with either 1, 2, 3 or 4 bits of precision. The lower nibble of the second byte is always 4'b0000. A returned value of 0x09 0xc0 would translate to 9.75 °C.

The TMP101 device also has comparison registers for asserting an alarm output, but this is not used in the demo.

SPI Flash

The SPI Flash is an Atmel AT25DF041A device. The device has numerous sectors, but only the first sector is used for storage. This is to reduce the erase time and keep the addressing to 16 bits (less software overhead).

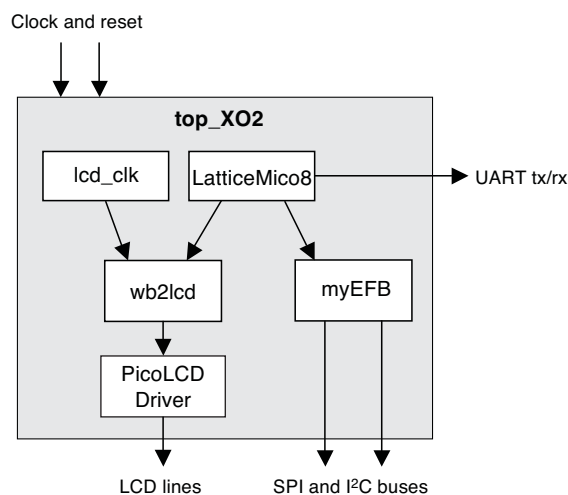
The SPI Flash has a Sector Protect register that must be disabled at power-up to allow erasing and writing to the sector.

Bytes are written to and read from the Flash two at a time, corresponding to the high and low bytes of the temperature register.

Top Level

The top level of the hardware is in XO2_top.v. A graphical representation of the modules instantiated and input/output signals is shown below.

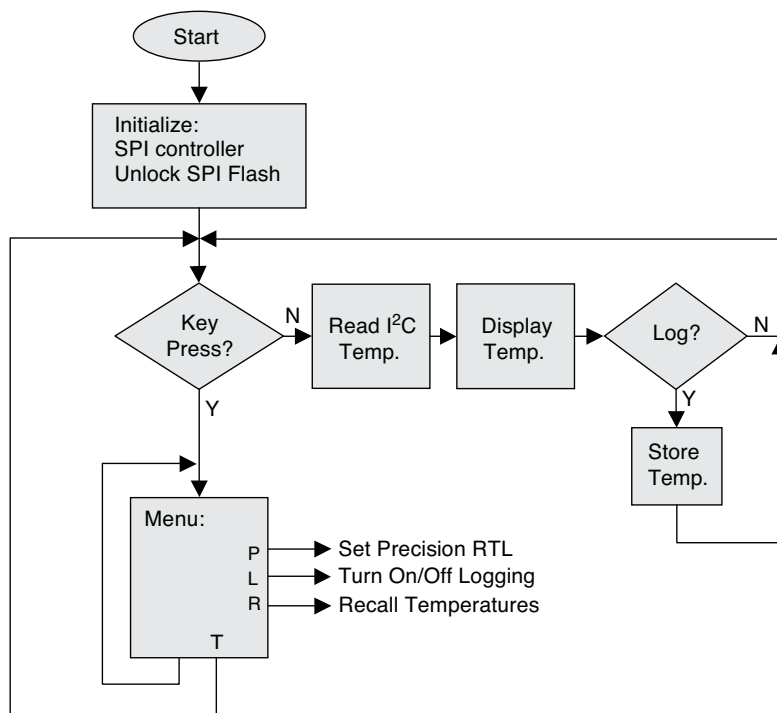
Figure 20. RTL Hierarchy



Demo Software Architecture

The LatticeMico8 platform provides a typical 8-bit embedded processor system for user I/O and data acquisition, conversion and storage. The 8-bit LatticeMico8 soft microcontroller uses on-chip memory (EBR) for program and data storage. It uses the UART for user I/O and the I²C master and SPI master EFB components to access the respective hardware devices. The software operates in a display loop, constantly displaying the currently read temperature until a key is pressed. Then the menu is displayed and the user makes a selection and execution either returns to displaying the temperature or stays in the menu.

Figure 21. Software Flow



Software Modules

The following global optimizations and coding styles have been used in most routines to generate the smallest executable size possible:

- Global parameters are used for commonly shared data, instead of passing params or local params, to keep memory references low
- Use of fixed addresses (i.e., platform #defines) for hardware devices, rather than a pointer variable
- Try to perform all computations and manipulations on a variable in localized area, rather than spread the computation over stages throughout the function.
- Computational algorithms produce a smaller footprint than look-up type algorithms

I²C Driver

The I²C driver is adapted from the standard LatticeMico8 EFB I²C driver that can be found in the Micosystem EFB component drivers directory. The source file is MicoEFB_I2C.c. Optimizations were made to reduce the code size and focus on the minimum required for operation. The optimizations include:

- Fixed, compile-time address for the EFB. A pointer to a Mico context is not used.
- Fixed access to I²C #1 only. This reduces code and eliminates a parameter that needs to be passed.
- Operation is master only. All slave mode functionality was removed. This also eliminates a parameter.
- No error checking or state checking. The only checking is to wait until it is not busy with an operation. However, no timeout is implemented so the operation could wait indefinitely.
- Read/writes can insert start/stops to eliminate further function calls from the main program.

SPI Driver

The SPI driver is adapted from the standard LatticeMico8 EFB SPI driver that can be found in the Micosystem EFB component drivers directory. The source file is EFB_SPI.c. Optimizations were made to reduce the code size and focus on the bare minimum required for operation. The optimizations include:

- Fixed, compile-time address for the EFB. A pointer to a Mico context is not used
- Operation is master only. All slave mode functionality was removed. This reduces code size and eliminates a parameter that needs to be passed.
- No error checking or state checking. The only checking is to wait until it is not busy with an operation. However, no timeout is implemented so the operation could wait indefinitely.
- Only the MicoEFB_SPITransfer() function is called.

Serial Port

The standard LatticeMico8 UART driver routines are used, but in polling mode. The Mico putc() and getc() assembly routines are quite small. Further optimization could be possible by hard-coding the UART base address per the platform #define rather than using the pointer to a context.

The Mico UART routines are used to print one character at a time to the console, read one character at a time from the console, and check if a key has been hit (character pending). This provides the low-level user I/O functions for the menus and temperature displays.

Menu Loop

After initialization, the main() function enters the temperature reading loop or the menu loop. The menu loop shows the menu and waits for the user to make a selection.

- **P - Precision** – Increments the precision mode and writes the new setting to the I²C Temperature device using the setPrecision() function that calls the I²C driver write function.
- **T - Get Temperature** – Returns to displaying the temperature reading.
- **L - Log Temperatures** – Erases SPI Flash sector 0, resets the SPI access address to 0x0000, enables logging and exits to the Temperature display loop.
- **R - Recall Temperatures** – Resets the SPI access address to 0x0000 to read from the beginning of the sector, calls SPI_RecallTemp() to read two bytes from SPI Flash current address, loads into RawTempValHi and RawTempValLo, and calls ShowTemp() as long as the two bytes aren't both 0xff (indicating reading into erased Flash area).

Temperature Reading

The temperature loop is read from the I²C Temperature device using the readI2CTemp() function. It first sends the register address to the I²C device, and then sends a read command for two bytes. The returned two bytes are the raw temperature register values. They are stored in the global variables RawTempValHi and RawTempValLo for use by other functions, such as showTemp().

Temperature Display

The temperature display routine, showTemp(), takes the 2-byte temperature value in the global variables RawTempValHi and RawTempValLo and converts the upper byte into decimal hundreds, tens and ones values for display. The fractional temperature value in the low byte indexes a look-up table for the direct text string to display on the screen. The actual conversion to a decimal number is not done since there are at most only 16 possible values (in 12-bit precision).

Displaying the digits to the LCD is done by writing the binary value for each digit to the LCD slave registers.

Negative numbers are not quite as straightforward with regard to the fractional byte. A complementary portion of the lookup table is used for "counting backwards" for negative fractions.

When in Recall mode, the showTemp() function is called after the global variables have been updated by reading the SPI Flash.

The display routine can be rebuilt to display the 2-byte raw hex value read from the I²C temperature sensor. The compile time option is:

```
#define DISPLAY_RAW_HEX_DATA
```

Temperature Storage

The first time logging is enabled (detected 'L' key pressed) the sector is erased with the SPI_EraseSector0() function.

Then, after each new temperature value is obtained and displayed, the contents of the global variables RawTempValHi and RawTempValLo are written to the SPI Flash. The function SPI_StoreTemp() first unlocks the SPI Flash with a Write Enable command, and then writes the two bytes to the current address. Then the address is incremented by two for the next write to the next location.

SPI_UnProtectSector0() is called at initialization time to unlock Sector 0 for access. The SPI Flash has a protection mechanism that powers up with all sectors locked. Software needs to unlock the sector before erasing and writing to it.

Temperature Recall

Resets address to 0x0000, and reads two bytes at a time from the SPI Flash, storing them into RawTempValHi and RawTempValLo until 0xff 0xff is read, indicating unprogrammed Flash values, and an illegal temperature value. Further reading and displaying is halted. The showTemp() routine is called to display the temperature on the screen and LCD, same as in normal Temperature Display operation.

Code Generation

This section gives some guidance on building the 'C' application code (including tips for keeping it small) and the steps to building and deploying the LatticeMico8 executable image.

Optimization Tips

- Uncalled functions will not be included in the binary executable. The linker checks to see if a function is referenced and excludes it from the build. Removing unused source will not reduce the final execution size.
- Lookup tables may not be the most code-efficient method. Experiment with different methods.
- Keep variable declarations close to where the actual operations on the code are done.
- Try moving the execution order of code around within a function. Sometimes different execution orders produce smaller code size.
- Watch out for .text size >= 6144 bytes (2048 instructions * 3 bytes per instruction)
- Macros are used to access hardware registers (pointer dereferencing does not work in LatticeMico8):
 - #define LM8_READ_MEM_BYTE(X, Y) (Y) = (__builtin_import((size_t)(X)))
 - #define LM8_WRITE_MEM_BYTE(X, Y) (__builtin_export((char)(Y), (size_t)(X)))
- The compiler optimization level is set at -O1 for the application code. Experimentation with other settings, including size optimization (-Os), did not produce as good a result as -O1.
- Use in-line assembly or call assembly routines from the main C control loop. This approach is effective, but not used in this demo because the objective was all 'C' code.

Building

The 'C' project to import (C/C++ > File > Import > Existing Projects into Workspace) is located in \Software\TempLog.

The LatticeMico8 Mico System Builder platform to reference is located in Hardware\mico8\soc\mico8.msb. Refer to the Readme.txt file in the Workspace folder for instructions on importing the Mico platform and C/C++ projects into the Lattice Mico System tool.

Enable Map File Generation: Add the following -Wl,-Map,mem.map to the linker options to produce a memory map of the linked executable image. This will help to understand which routines are consuming the most instructions and where to look for optimization.

Software Deployment: After building the 'C' source code, be sure to deploy the software to the on-chip memory initialization files: Software\prom_init.mem and Software\scratchpad_init.mem.

Use: **Tools > Software Deployment > Mico8 Memory Deployment.**

Additional MachXO2 Projects

There are two additional LatticeMico8 MachXO2 'C' projects that provide more diagnostics and information about the I²C operations to the temperature sensor and SPI operations to the SPI Flash device. Both these projects use the same hardware platform. The source code is optimized for one type of device operation or the other.

- Software\I2C_lite can be used to debug and simulate I²C accesses. See Appendix A.
- Software\spi_rdwr can be used to debug and simulate SPI accesses. See Appendix B.

Known Limitations

This section explains some of the limitations of the MachXO2 LatticeMico8 Temperature Logging Demo.

Instruction Code Size

The maximum size is 2,048 LatticeMico8 instructions, which equals 6,144 bytes. The demo design is very close to the limit. If rebuilding the demo software, pay attention to the .text size. If it grows beyond 6,144 bytes, the linker stage will fail saying the .text is too large.

This has major implications on the amount of experimenting or customizing that can be done. Even a simple change might exceed the available number of instructions. See Appendix A or Appendix B for other software projects that have a little more room left.

Error Checking

No checking for errors is done to save instruction space.

Drivers are Customized

The I²C and SPI drivers are customized to operate in only Master mode and access just the first I²C controller. The flexibility has been removed to reduce code footprint.

Limited User Display

Menu functions and display format are kept to a minimum.

Temperature conversion to Fahrenheit is not implemented.

Fixed LCD Output Format

To keep the LCD display routine software as simple as possible, a fixed 3 digit plus ".0" or ".5" is used for positive numbers. Leading zeros are always displayed, for example 002.5. The display routine only looks at the upper bit of the LSB to determine if a ".5" or ".0" should be displayed; it does not attempt to make intelligent formatting decisions, such as: 1.125 or -11.5, or do rounding to display true 1/10ths precision: 1.0, 1.1, 1.2, 1.3, 1.4, ...

The routines would be too large to fit with all the other tasks that need to be done, so LCD display formatting was sacrificed.

Rounding is Not Exact

To keep the math as simple as possible (to conserve instruction space), true rounding is not done for the LCD display. For example, if the temperature is 20.875 °C, the LCD will show 20.5 instead of rounding up to 21.0 (20.875 + 0.5).

Temperature Reading is Not Ambient Air Temperature

The temperature device senses the heat generated by the components surrounding it on the MachXO2 Pico Evaluation Board. It will not report the ambient air temperature in the room.

Logging Data

The logging feature starts at the beginning of the sector each time it is enabled. You cannot append to existing logged data.

Logging is limited to the first 64kB sector in the SPI Flash. This keeps the software simple and small. 64 kB addressing only requires two bytes (16 bits). Reading/writing/erasing more sectors would have required use of a 24-bit address (made out of three bytes) which would have increased the instruction count for this math.

References

- EB61, [MachXO2 Pico Development Kit User's Guide](#)
- Digital Temperature Sensor with I²C Interface (Texas Instruments, Inc.)
- Atmel AT25DF041A SPI Flash Data Sheet

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
April 2012	01.0	Initial release.

Appendix A. I²C Driver Test Project

Another software project is also included in this demo. This project is named I2C_lite and was used to develop, test and debug the customized I²C drivers. A separate project was used to test the I²C driver because of the limited executable code space in the LatticeMico8 in a MachXO2 device. Debug and diagnostics would not fit along with the SPI driver.

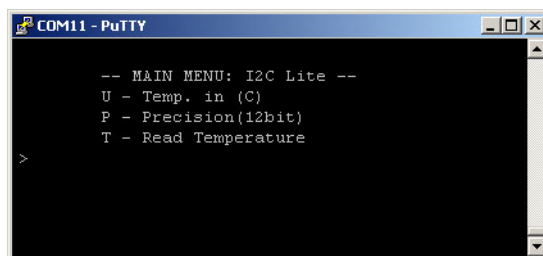
This software project can be imported from the Software/ directory into the work space that the TempLog project resides in. This project also allows simulation. The simulation is done using an I²C EEPROM model because a behavioral model for the temperature sensor could not be located.

Simulation allows “seeing” how the I²C read and writes will actually appear on the wires and to debug any issues with the driver, and verifying the driver is adhering to the I²C protocol. Building for simulation is done by defining SIMULATION in the GCC compiler. Select the SIM build configuration of the project: **Properties > C/C++Build > Configuration = SIM**. This will disable the normal menu code, and build in test code that does specific accesses to the I²C EEPROM. The EEPROM does not physically exist on the MachXO2 Pico Evaluation Board. It is used as a target I²C slave device for access operations during simulation.

For simulation, be sure to build the project configuration SIM, and then deploy the resulting code to the memory initialization files before running the simulation, so the LatticeMico8 executes the test code for simulation, and not the full menu code.

The following menu options are available in the I²C Lite project.

Figure 22. I²C Lite Menu



- **U** – Changes the units. Celsius or Raw Hex format can be selected. This affects the console display format and the LCD display format. Hex display format shows the values as read directly from the temperature sensor registers, before conversion to Celsius.
- **P** – Changes the precision: 9, 10, 11, or 12-bit
- **T** – Reads and displays the temperature

The purpose of this project is to investigate and optimize operation of the I²C drivers and temperature device, as well as formatting and displaying the temperature value.

Appendix B. SPI Driver Test Project

A third software project is also included in this demo. This project is named SPI_RdWr and was used to develop, test and debug the customized SPI drivers. A separate project was used to test the SPI driver because of the limited executable code space in the LatticeMico8 in a MachXO2 device. Debug and diagnostics would not fit along with the I²C driver.

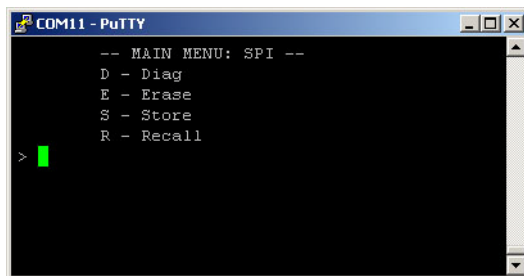
This software project can be imported from the Software/ directory into the work space that the TempLog project resides in. This project also allows simulation. The simulation is done using an Atmel SPI Flash model of the device on the evaluation board. *Note: This model, is not the same as the data sheet for the device on the MachXO2 Pico Evaluation Board, but works well enough for the basic commands.*

Simulation allows the user to see how the SPI read and writes will actually appear on the wires and to debug any issues with the driver and verify that the driver is adhering to the SPI protocol. Building for simulation is done by defining SIMULATION in the GCC compiler. Select the SIM build configuration of the project: **Properties > C/C++Build > Configuration = SIM**. This will disable the normal menu code and build in test code that does specific accesses to the SPI Flash.

For simulation, be sure to build the project configuration SIM, and then deploy the resulting code to the memory initialization files before running the simulation, so the LatticeMico8 executes the test code for simulation, and not the full menu code.

The following menu options are available in the SPI only project.

Figure 23. SPI Read Write Menu



- 'D' prints the current status register contents, for example:
 - DevID: 1F 44 01 00 00
 - S: 14

Where 'DevID' is the SPI Flash manufacturer ID in Hex, and 'S:' is the status register value in Hex.

- 'E' clears Sector 0 contents of the SPI Flash.
- 'S' stores 16 bytes into the SPI Flash. The pattern is an incrementing 8-bit value. The address also increments.
- 'R' recalls and displays what has been stored in SPI Flash.