App Note:

SwiftX Application Development

FORTH, Inc.

Software products and services since 1973
www.forth.com

Current revision: July 2012

FORTH, Inc.
Los Angeles, California USA
www.forth.com

SECTION 1: OVERVIEW

This paper demonstrates the use of SwiftX in the development of a very simple embedded application, a Morse code beacon that flashes out the universal distress signal “S.O.S.” on an LED.

The application is ported to four off-the-shelf evaluation boards, each with a different CPU type.

1.1 About SwiftX

SwiftX is FORTH, Inc.’s interactive cross-compiler, a fast and powerful tool for the development of software for embedded microprocessors and microcontrollers. SwiftX is based on the Forth programming language and is itself written in Forth.

SwiftX has been ported to the following microprocessor and microcontroller families:

- Freescale 6801 / Renesas 6303
- Freescale 6809
- Freescale 68HC11
- Freescale 68HC12 (S12, S12X, etc.)
- Freescale 68HCS08
- Freescale 68K
- Aeroflex UTMC 69R000
- Intel (NXP, SiLabs, others) 8051
- Atmel, Cirrus, NXP, ST Microelectronics (and other) ARM core
- Atmel AVR
- Freescale ColdFire
- Renesas H8H (H8/300H, H8S)
- Intel (AMD, other) i386
- Texas Instruments MSP430
- Patriot PSC1000
- Harris RTX2010

The suite of SwiftX cross-compilers are all applications that run in the SwiftForth for Windows programming environment. As such, they inherit all the features of SwiftForth and extend its interactive development environment to manage multiple program and data spaces as well as generate the code and data that fill them.

SwiftX provides the user with the most intimate, interactive relationship possible with the target system, speeding the software development process and helping to ensure thoroughly tested, bug-free code. It also provides a fast, multitasking kernel and libraries to give you a big head start in developing your target application.

1.2 Evaluation Boards

SwiftX is available ready to run on a wide variety of off-the-shelf evaluation boards. Three of them will be used in this sample application.

The initial application programming and development was done on the Axiom Manufacturing CMS-8GB60 evaluation board, which features the Freescale[1] MC9S08GB60, a member of the HCS08 family of 8-bit microcontrollers. This board is the OEM version of the Freescale M68DEMO908GB60 board.

The application was then quickly ported to the Freescale M52235EVB, Olimex LPC-P2103, and Texas Instruments MSP-ESP430G2 LaunchPad boards. The M52235EVB features the 32-bit MCF52235 ColdFire V2 core. The LPC-P2103 features the NXP[2] LPC2103, a low-cost microcontroller built around the 32-bit ARM7TDMI core. The TI LaunchPad is a development tool for the MSP430 Value Line microsontrollers.

Details about these boards are presented in Appendix A.

1.Formerly Motorola Semiconductor Products Sector
2.Formerly Philips Semiconductors

SECTION 2: MORSE CODE APPLICATION

Keeping portability in mind, the simple Morse code "SOS" application will be structured in two layers. The lower layer supplies the hardware application programming interface (API). The upper layer is the application itself.

2.1 About Morse Code

The material in this section is excerpted from the Morse code topic on Wikipedia. The complete topic with all its technical details, illustrations, and references can be found here:

Reference `http://en.wikipedia.org/wiki/Morse_code`

Morse code is a method for transmitting telegraphic information, using standardized sequences of short and long elements to represent the letters, numerals, punctuation and special characters of a message. The short and long elements can be formed by sounds, marks, or pulses and are commonly known as "dots" and "dashes" or "dits" and "dahs".

International Morse code is composed of six elements:

1. short mark, dot or "dit" (·)
2. longer mark, dash or "dah" (-)
3. intra-character gap (between the dots and dashes within a character)
4. short gap (between letters)
5. medium gap (between words)
6. long gap (between sentences — about seven units of time)

These six elements serve as the basis for International Morse code and therefore can be applied to the use of Morse code world-wide.

Morse code can be transmitted in a number of ways: originally as electrical pulses along a telegraph wire, but also as an audio tone, a radio signal with short and long tones, or as a mechanical or visual signal (e.g. a flashing light) using devices like an Aldis lamp or a heliograph. Morse code is transmitted using just two states (on and off) so it was an early form of a digital code. However, it is technically not binary, as the pause lengths are required to decode the information.

The length of the dit determines the speed at which the message is sent, and is used as the timing reference.

The speed of Morse code is typically specified in words per minute (WPM). A dah is conventionally three times as long as a dit. The spacing between dits and dahs within a character is the length of one dit; between letters in a word it is the length of a dah (three dits); and between words it is seven dits. The "Paris" Morse code standard defines the speed of transmission as the dot and dash timing needed to send the word "Paris" a given number of times per minute. The word "Paris" is used

because it is precisely 50 “dits” based on the textbook timing.

Under this standard, the time for one “dit” can be computed by the formula:

T = 1200 / W

Where: W is the desired speed in words-per-minute, and T is the duration of one dit in milliseconds.

2.2 Driver Layer Development

Morse code can be transmitted in bursts of audio tone, continuous wave radio frequency (CW RF), direct current over a wire, or light. In this application, we will use one of the LEDs on the evaluation board to provide an “SOS” beacon.

The API needs to supply three basic functions:

- Turn the LED on
- Turn the LED off
- Hardware initialization

We will define these Forth words to supply those functions:

Glossary

+LED (--)

Turn the LED on.

-LED (--)

Turn the LED off.

/LED (--)

Perform any necessary hardware initialization for LED output.

2.2.1 CMS-8GB60 LED Interface

As shown in the schematic section in Figure 1, the CMS-8GB60 board has 5 LEDs available on I/O ports PTF and PTD. We’ll use LED1 (PTF0) for this application. PTF is a multi-function 4-pin I/O port. The default function for each pin is general-purpose I/O and the default mode for each GPIO is input. Therefore, our initialization routine needs to set PF0 as an output with an initial state of logic 1:

```
CODE /LED ( -- )   PTFDD 0 BSET   PTFD 0 BSET   RTS   END-CODE
```

Turning the LED on and off is accomplished by setting PTF0 low and high, respectively:

```
CODE +LED ( -- )   PTFD 0 BCLR   RTS   END-CODE
CODE -LED ( -- )   PTFD 0 BSET   RTS   END-CODE
```

Note the use of **CODE** definitions to take advantage of the nice bit manipulation operators in the HCS08 instruction set.

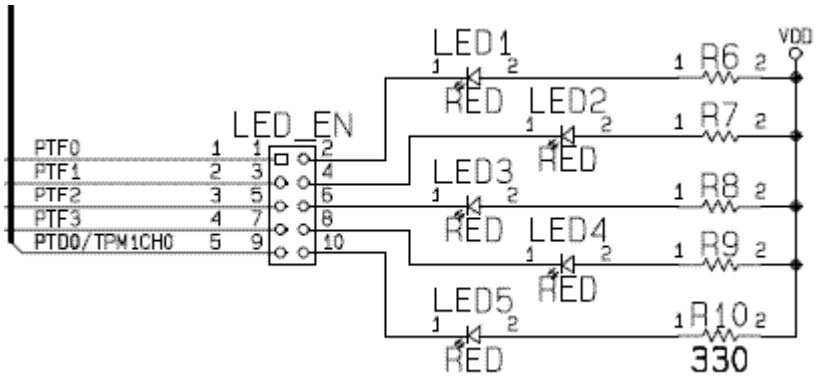

Figure 1. CMS-8GB60 LED port connections

2.2.2 M52235EVB LED Interface

As shown in the schematic block in Figure 2, the M52235EVB has 4 LEDs available on I/O pins TIN[3:0]. We'll use LED1 (TIN0) for this application.

These pins are connected to the MCF52235's PORT TC, which is a multi-function 4-pin I/O port. The primary and alternate pin functions are shown in Figure 3.

Note that the default function (which is not the primary function -- just to keep us alert!) for each pin is general-purpose I/O and the default mode for each GPIO is input. Therefore, our initialization routine needs to set PTC0 as an output with an initial state of logic 0.

Turning the LED on and off is a simple matter of writing to the `SETTC` and `CLRTC` registers that set and clear GPIO output pins:

```
: +LED ( -- )   1 SETTC C! ;
: -LED ( -- )   $FE CLRTC C! ;
: /LED ( -- )   DDRTC C@  1 OR  DDRTC C!  -LED ;
```

Note that I/O registers in the ColdFire family are memory mapped and can be accessed with conventional memory operators as shown above.

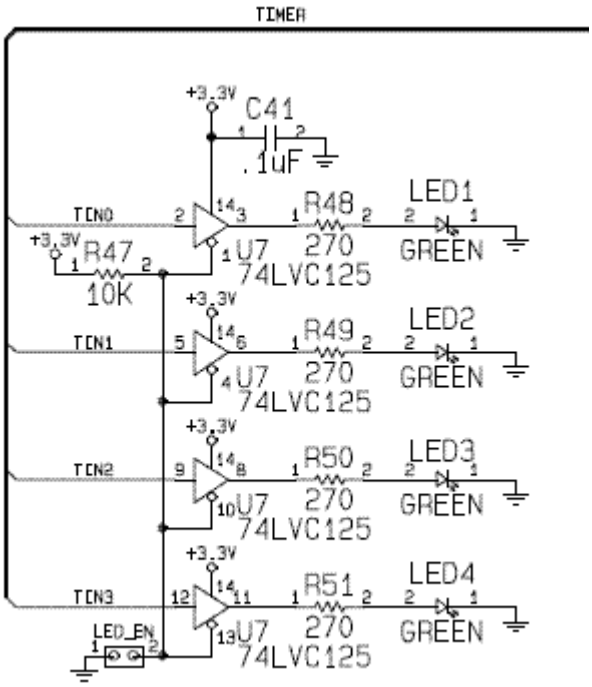

Figure 2. M52235EVB LED port connections

Figure 3. MCF52235 Port TC pin functions

2.2.3 LPC-P2103 LED Interface

The LPC-P2013 board has a single LED available, connected to port pin P0.26 as shown in Figure 4. This is a multi-function I/O pin whose default function is GPIO (input). The LED output is active low, so our output initialization routine needs to set P0.26 as an output with an initial state of logic 1.

Turning the LED on and off is a simple matter of clearing and setting P0.26.

```
1 26 LSHIFT EQU %LED

: +LED ( -- )   %LED IOCLR ! ;
: -LED ( -- )   %LED IOSET ! ;
: /LED ( -- )   -LED  IODIR @  %LED OR  IODIR ! ;
```

Note the use of the constant **%LED** which is the bit mask for bit 26. We use the `IOCLR` and `IOSET` registers to clear (turn LED off) and set (turn LED on) the I/O bit.

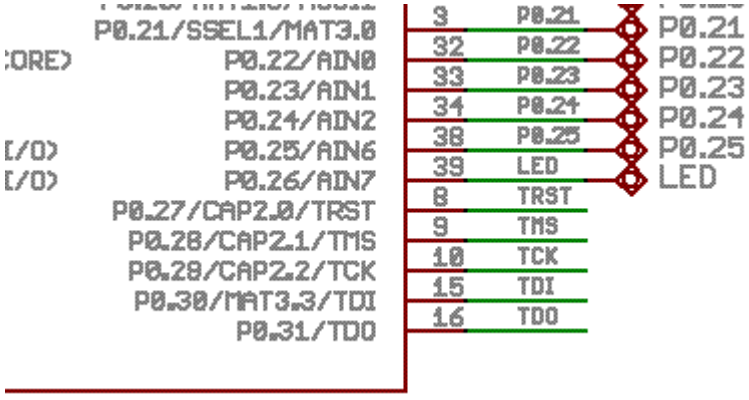

Figure 4. LPC-P2103 LED pin assignment

2.2.4 LaunchPad LED Interface

The LaunchPad board has two user LEDs available, both connected to pins on GPIO port P1 as shown in Figure 5.

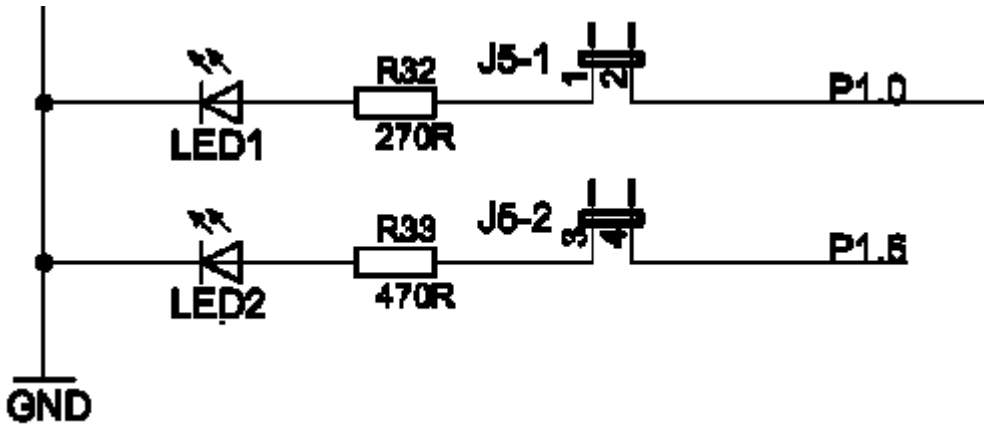

Figure 5. LaunchPad LED pin assignments

For this application, we'll use LED1 on port pin P1.0. The LED output is active high, so our output initialization routine needs to set P1.0 as an output with an initial state of logic 0.

Turning the LED on and off is a simple matter of setting and clearing P1.0.

```
CODE +LED ( -- )   1 # P1OUT & BIS   RET   END-CODE
CODE -LED ( -- )   1 # P1OUT & BIC   RET   END-CODE
CODE /LED ( -- )   1 # P1DIR & BIS   1 # P1OUT & BIC   RET   END-CODE
```

2.3 Application Layer Development

The Morse code "SOS" application can be divided into sections:

- Timing functions
- Primary code elements ("dit" and "dah")
- Character codes
- Message

2.3.1 Timing Functions

Recall from the discussion of Morse code in Section 2.1 that the speed of Morse code transmission is typically specified in words per minute (WPM) and that the timing for a dah is conventionally 3 times as long as a dit. The spacing between dits and dahs within a character is the length of one dit; between letters in a word it is the length of a dah (3 dits); and between words it is 7 dits.

Under the "Paris" standard, the time for one "dit" can be computed by the formula:

```
Tu = 1200 / W
```

Where: W is the desired speed in words-per-minute, and Tu is one dit-time in milliseconds.

So our basic timing and WPM setting functions might look like this:

```
CREATE Tu 120 ,

: WPM ( n -- )   1200 SWAP /  Tu ! ;
: DELAY ( n -- )   Tu @ * MS ;
```

Tu holds the current value of one "dit" time. The default value of 120 sets the initial rate at 10 WPM as defined above.

WPM sets **Tu** based on the formula above.

DELAY pauses for *n* dit times using the standard SwiftX **MS** (millisecond delay) function.

2.3.2 Primary Code Elements

Using our LED API calls (**+LED** and **-LED**) along with **DELAY** timing defined above, we can build the **DIT** and **DAH** code elements:

```
: DIT ( -- )   +LED  1 DELAY  -LED  1 DELAY ;
: DAH ( -- )   +LED  3 DELAY  -LED  1 DELAY ;
```

Note that the one-unit intra-character delay time trails each code element.

2.3.3 Character Codes

Only the codes for letters 'S' (dit-dit-dit) and 'O' (dah-dah-dah) are required here.

```
: S ( -- )   DIT DIT DIT  2 DELAY ;
: O ( -- )   DAH DAH DAH  2 DELAY ;
```

Note the additional **2 DELAY** at the end of each character to supply the total three-unit inter-character delay time.

2.3.4 The Distress Signal

```
: SOS ( -- )  S O S  4 DELAY ;
```

Again, note the **4 DELAY** at the end which results in gap that is seven units in duration.

2.4 Background Task Assignment

We finish by assigning the infinite loop "SOS" output to a background task using the SwiftOS multitasker, available in all SwiftX implementations.

First, we define the task:

```
|U| |S| |R| BACKGROUND BEACON
```

This defines a background task named **BEACON** with initial user area size **|U|**, data stack size **|S|**, and return stack size **|R|**. These contants ("size of U", "size of S", and "size of R") are defined for each SwiftX implementation. They define "full-size" user and stack spaces. In applications tight for RAM, smaller values may be used as needed.

Next, we define an initialization procedure that performs the hardware setup and assigns the task's behavior:

```
: /BEACON ( -- )   /LED
   BEACON ACTIVATE  BEGIN  SOS  AGAIN ;
```

Finally, the instantiation of the task and assignment of its behavior need to be

added to the main system start-up code:

```
BEACON BUILD
/BEACON
```

We now have a portable application that can sit on top of the LED API for different target processors.

This page intentionally left blank.

Appendix A: Evaluation Board Details

This Appendix describes the Axiom CMS-8GB60, Freescale M52235EVB and Olimex LPC-P2103 evaluation boards.

A.1 Axiom CMS-8GB60 Board

The Axiom CMS8-GB60, pictured in Figure 6, is a low cost demonstration system for the Freescale M9S08GB60 microcontroller. The BDM (Background Debug Mode) port is provided for development tool application. SwiftX uses the BDM for its debug interface and for flash programming. A P&E Microsystems USB-ML-12 BDM cable is supplied with the SwiftX development kit.

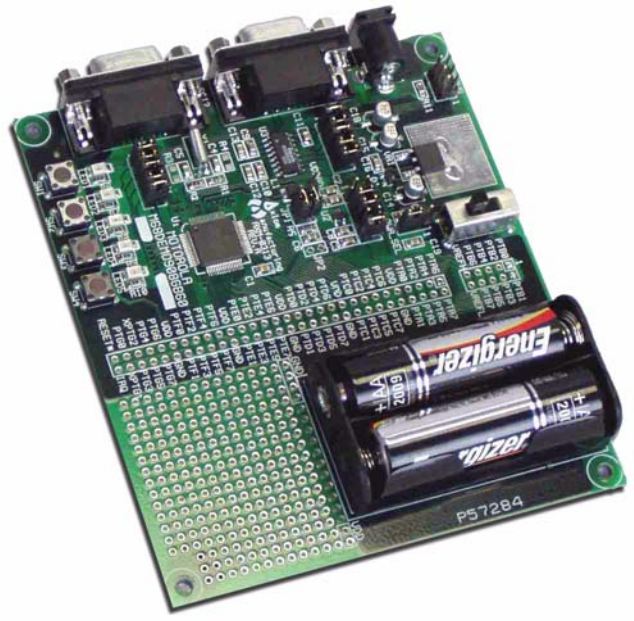

Figure 6. Axiom CMS-8GB60

Board features include:

- Freescale M9S08GB60 microcontroller
- 32kHz crystal
- Regulated +3.3V power supply
- SCI1&2 serial ports, RS-232 (DB9S)
- Power ON/OFF switch
- 5 LED indicators (PTF0-3, PTD0)
- 4 push switches (PTA4-7)

- Digital to analog (PTD2, PTB1)
- 1.8V analog reference
- Prototype area

The MC9S08GB60 is a member of the low-cost, high-performance HCS08 Family of 8-bit microcontroller units (MCUs). All MCUs in the family use the enhanced HCS08 core and are available with a variety of modules, memory sizes, memory types and package types.

The HCS08 Family is an extension of the HC08 Family, offering extended battery life with maximum performance down to 1.8 V, industry-leading flash memory, and built-in development support.

M9S08GB60 microcontroller features:

- 60K Byte Flash
- 4K Bytes internal SRAM
- 56 I/O lines
- 5 channel TPM 2 timer
- 3 channel TPM 1 timer
- 8-channel 10-bit A/D
- SPI and IIC ports
- 2 SCI serial ports
- Keyboard wake-up ports
- BDM debug port
- Clock generator, FLL up to 40Mhz

A.2 FREESCALE M52235EVB

The M52235EVB evaluation board, pictured in Figure 7, provides a development platform for a high-performance embedded design using the MCF5223x Family of ColdFire microcontrollers. The BDM (Background Debug Mode) port is provided for development tool application. SwiftX uses the BDM for its debug interface and for flash programming. A P&E Microsystems USB-ML-CF BDM cable is supplied with the SwiftX development kit.

Board features include:

- Ethernet port 10/100Mb/s with on-chip PHY
- Abort/IRQ7 logic switch (debounced)
- PLL Clocking options - Oscillator, Crystal or SMA for external clocking signals
- Three UART interfaces and standard DB9 connections
- Supports Zigbee-ready RF daughter card
- Breakout connector for serial interfaces (including I2C, QSPI, GPIO, and ADC)
- LEDs for power-up indication, general purpose I/O, and timer output signals

- Power over Ethernet
- Potentiometer and light sensor dedicated to the ADC
- CAN transceiver interface
- BDM interface

Figure 7. Freescale M52235EVB

The MCF5223x family of 68K/ColdFire devices are single-chip solutions that provide 32-bit control with an Ethernet interface. The MCF5223x Family integrates standard 68K/ColdFire peripherals, including three universal asynchronous receiver/transmitters (UARTs) for medium and long distance connections, an inter-integrated circuit (I2C) and queued serial peripheral interface (QSPI) for in-system communications to connected peripherals.

MCF52235 microcontroller features:

- V2 ColdFire core delivering 57 (Dhrystone 2.1) MIPS at 60 MHz
- eMAC Module and HW Divide
- 32k bytes SRAM
- 256k bytes Flash
- 10/100 Ethernet MAC with PHY
- Cryptographic Acceleration Unit with Random Number Generator
- CAN 2.0B Controller (FlexCAN)
- 3 UARTs
- QSPI
- I²C bus interface
- 12-bit ADC

A.3 OLIMEX LPC-P2103

The LPC-P2103 Evaluation Board, pictured in Figure 8, enables real-time code development and evaluation for the NXP LPC2103. The JTAG port is provided for development tool application. SwiftX uses the JTAG for its debug interface and for flash programming. A Rowley Associates Cross-Connect Lite JTAG USB cable is supplied with the SwiftX development kit.

Figure 8. Olimex LPC-P2103

Board features include:

- NXP LPC2103 MCU
- ARM JTAG 2x10 pin connector
- Extension header for uC pins
- 14.7456 Mhz crystal supports common baud rates (4x PLL = 58.9824 Mhz CPU clock)
- 32768 Hz crystal
- RS232 interface circuit
- User button
- Status LED
- Power supply LED
- RTC backup Li-ion battery
- Reset circuit
- Reset button
- DEBUG jumper for JTAG enable/disable
- BSL jumper for bootloader enable/disable
- RTCK pullup resistor
- Push BUTTON with pullup

- Power plug-in jack
- Single power supply: 6VAC or +9VDC
- Two on board voltage regulators 1.8V and 3.3V with up to 800mA current
- Prototype PCB area with +3.3V and GND bus

LPC2103 microcontroller features:

- ARM7TDMI CPU core
- 32k Bytes Program Flash
- 8k Bytes SRAM
- RTC
- 2 UARTs
- 2 I2C
- SPI
- 5 32-bit timers
- 8-channel, 10-bit ADC
- CCR
- PWM
- WDT
- 5V tolerant I/O
- Up to 70MHz operation

A.4 Texas Instruments MSP-EXP430G2 "LaunchPad"

The MSP-EXP430G2 LaunchPad, pictured in Figure 9, is an easy-to-use flash programmer and debugging tool that provides everything you need to start developing on MSP430 Value Line devices. It includes a 14-/20-pin DIP socketed target board with integrated emulation to quickly program and debug MSP430 Value Line devices in-system through the Spy Bi-Wire (2-wire JTAG) protocol. The flash memory can be erased and programmed in seconds with no external power supply required due to the MSP430's ultra-low power flash. SwiftX uses the JTAG interface for flash programming and as its debug interface.

Board features include:

- One 14-/20-pin DIP (N) socket
- Built-in flash emulation for debugging and programming
- Two programmable LEDs
- One power LED
- One programmable button
- One reset button

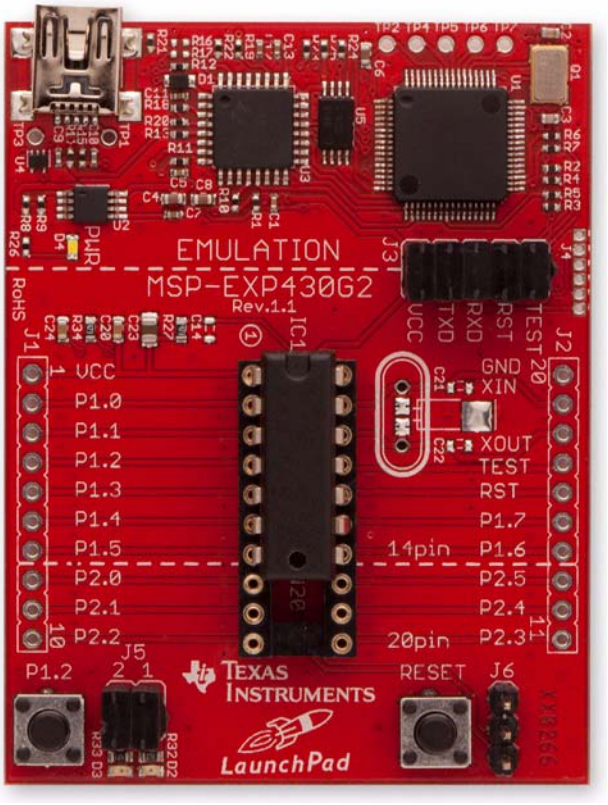

Figure 9. LaunchPad Board

MSP430G microcontroller features:

- Low supply-voltage range: 1.8V to 3.6V
- Ultra-low power consumption
- Five power-saving modes
- Ultra-fast wake-up from standby mode in less than 1 µs
- 16-Bit RISC architecture, 62.5-ns instruction cycle time
- Internal frequencies up to 16 MHz with four calibrated frequencies
- Internal very-low-power low-frequency oscillator
- 32-kHz crystal
- External digital clock source
- Two 16-Bit timers with three capture/compare registers
- Up to 24 touch-sense-enabled I/O pins
- Universal serial communication interface (USCI) with UART, IrDA, SPI, and I2C modes
- On-chip comparator for analog signal compare function
- 10-Bit 200-ksps analog-to-digital (A/D) converter with internal reference
- Brownout detector

- Serial onboard programming, no external programming voltage needed
- Programmable code protection by security fuse
- On-chip emulation logic with Spy-Bi-Wire interface