```
/* Copyright (c) 2012 Oberon microsystems AG & CSA Engineering AG
 * (Switzerland)
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License. */

using Microsoft.SPOT.Hardware;

/// <summary>
/// A C# application program needs to provide hardware-specific
/// information when it initializes its I/O. For Mountaineer
/// boards, there are four different ways of doing this:
///
///  1) For Gadgeteer programs, with the visual designer:
///     Use the visual designer that comes with the Gadgeteer
///     SDK and plug your system together, using a mainboard plus
///     suitable add-on hardware modules.
///     The Gadgeteer tools in Visual Studio automatically generate
///     the code that represents the hardware configuration of your
///     system.
///     With that approach, you don't need this namespace here.
///
///  2) For Gadgeteer programs, without the visual designer:
///     In the configuration section of your program, instantiate
///     the drivers for the hardware modules that you use. You
///     need to pass the socket ID when you instantiate the driver,
///     thereby telling the Gadgeteer core libraries how your
///     hardware configuration looks like. A special mainboard driver
///     module, provided by the mainboard producer, tells the
///     Gadgeteer core libraries which sockets are available, and
///     how their pins are mapped to microcontroller pins.
///     With that approach, you don't need this namespace here.
///
///  3) For "plain vanilla" NETMF programs, without Gadgeteer libraries
///     but with a socket-oriented view on hardware features:
///     You want your application to clearly reflect its use of
///     Mountaineer mainboard sockets, so that the mapping
///     between the physical hardware and your application program
///     is as direct and obvious as possible.
///     This is the only approach for which you need this namespace here!
///
///  4) For "plain vanilla" NETMF programs, without Gadgeteer libraries
///     but with a microcontroller pin-oriented view on hardware features:
///     You want your application to directly depend on the
///     microcontroller's pinout only, so that you can most easily
///     migrate to a custom board with the same microcontroller
///     later on.
///     Use the Mountaineer.Stm32.Hardware namespace.
///     With that approach, you don't need this namespace here.
///
/// This namespaces is designed to adhere to the naming rules of .NET.
/// See http://msdn.microsoft.com/en-us/library/ms229002.aspx
/// In particular, the following rules are followed, even though they
/// may be violated in some other NETMF APIs:
///
/// - Do not prefix enum values with type name (no PWM_ or similar prefixes)
/// - Identifiers must not contain underscores
/// - Acronyms with more than two letters must be Pascal-cased
///     (Pwm instead of PWM), except for brand names
/// - Names in identifiers must not be abbreviated
///
/// - "NETMF" is considered a brand name, but still Pascal-cased
///
/// When using FxCop, add the words "netmf", "pwm", "spi", and "mainboard"
/// as recognized words to CustomDictionary.xml.
/// </summary>
```

```
namespace Mountaineer.Netmf.Hardware
{
    // <summary>
    // Hardware provider for Mountaineer mainboards. This class is already
    // implemented generically by NETMF, so no methods are overridden. The
    // generic implementation uses HAL features to obtain the hardware
    // configuration.
    // </summary>
    class MountaineerHardwareProvider : HardwareProvider
    {
        static MountaineerHardwareProvider();
    }
```

```
/// <summary>
/// socket 1 (type CY)
/// </summary>
public static class Socket1
{
    /// <summary>
    /// socket 1 (type CY), pin 3, PD11
    /// </summary>
    public const Cpu.Pin Pin3 = (Cpu.Pin)59;

    /// <summary>
    /// socket 1 (type CY), pin 4, PD1/CAN1_TX
    /// </summary>
    public const Cpu.Pin Pin4 = (Cpu.Pin)49;

    /// <summary>
    /// socket 1 (type CY), pin 5, PD0/CAN1_RX
    /// </summary>
    public const Cpu.Pin Pin5 = (Cpu.Pin)48;

    /// <summary>
    /// socket 1 (type CY), pin 6, PD10
    /// </summary>
    public const Cpu.Pin Pin6 = (Cpu.Pin)58;

    /// <summary>
    /// socket 1 (type CY), pin 7, PE3
    /// </summary>
    public const Cpu.Pin Pin7 = (Cpu.Pin)67;

    /// <summary>
    /// socket 1 (type CY), pin 8, PE4
    /// </summary>
    public const Cpu.Pin Pin8 = (Cpu.Pin)68;

    /// <summary>
    /// socket 1 (type CY), pin 9, PE15
    /// </summary>
    public const Cpu.Pin Pin9 = (Cpu.Pin)79;
}
```

```
/// <summary>
/// socket 2 (type SUX)
/// </summary>
public static class Socket2
{
    /// <summary>
    /// socket 2 (type SUX), pin 3, PE7
    /// </summary>
    public const Cpu.Pin Pin3 = (Cpu.Pin)71;

    /// <summary>
    /// socket 2 (type SUX), pin 4, PD8/USART3_TX
    /// </summary>
    public const Cpu.Pin Pin4 = (Cpu.Pin)56;

    /// <summary>
    /// socket 2 (type SUX), pin 5, PD9/USART3_RX
    /// </summary>
    public const Cpu.Pin Pin5 = (Cpu.Pin)57;

    /// <summary>
    /// socket 2 (type SUX), pin 6, PB9
    /// </summary>
    public const Cpu.Pin Pin6 = (Cpu.Pin)25;

    /// <summary>
    /// socket 2 (type SUX), pin 7, PB5/SPI1_MOSI
    /// </summary>
    public const Cpu.Pin Pin7 = (Cpu.Pin)21;

    /// <summary>
    /// socket 2 (type SUX), pin 8, PB4/SPI1_MISO
    /// </summary>
    public const Cpu.Pin Pin8 = (Cpu.Pin)20;

    /// <summary>
    /// socket 2 (type SUX), pin 9, PB3/SPI1_SCK
    /// </summary>
    public const Cpu.Pin Pin9 = (Cpu.Pin)19;

    /// <summary>
    /// COM3 is mapped to socket 2 (type U) and to on-chip UART3.
    /// </summary>
    public const string SerialPortName = "COM3";

    /// <summary>
    /// SPI1 is mapped to socket 2 and to on-chip SPI1.
    /// </summary>
    public const Microsoft.SPOT.Hardware.SPI.SPI_module SpiModule =
        Microsoft.SPOT.Hardware.SPI.SPI_module.SPI1;
}
```

```
/// <summary>
/// socket 3 (type KUY)
/// </summary>
public static class Socket3
{
    /// <summary>
    /// socket 3 (type KUY), pin 3, PE8
    /// </summary>
    public const Cpu.Pin Pin3 = (Cpu.Pin)72;

    /// <summary>
    /// socket 3 (type KUY), pin 4, PD5/USART2_TX
    /// </summary>
    public const Cpu.Pin Pin4 = (Cpu.Pin)53;

    /// <summary>
    /// socket 3 (type KUY), pin 5, PD6/USART2_RX
    /// </summary>
    public const Cpu.Pin Pin5 = (Cpu.Pin)54;

    /// <summary>
    /// socket 3 (type KUY), pin 6, PD4/USART2_RTS
    /// </summary>
    public const Cpu.Pin Pin6 = (Cpu.Pin)52;

    /// <summary>
    /// socket 3 (type KUY), pin 7, PD3/USART2_CTS
    /// </summary>
    public const Cpu.Pin Pin7 = (Cpu.Pin)51;

    /// <summary>
    /// socket 3 (type KUY), pin 8, PE0
    /// </summary>
    public const Cpu.Pin Pin8 = (Cpu.Pin)64;

    /// <summary>
    /// socket 3 (type KUY), pin 9, PE1
    /// </summary>
    public const Cpu.Pin Pin9 = (Cpu.Pin)65;

    /// <summary>
    /// COM2 is mapped to socket 3 (type KU) and to on-chip UART2.
    /// </summary>
    public const string SerialPortName = "COM2";
}
```

```
/// <summary>
/// socket 4 (type PY)
/// </summary>
public static class Socket4
{
    /// <summary>
    /// socket 4 (type PY), pin 3, PE9/PWM_CH4
    ///
    /// Pin 3 could also be used as PWM channel 4.
    /// This pin-assignment does not conform to
    /// the official Gadgeteer types and thus
    /// should not be used if it can be avoided.
    /// </summary>
    public const Cpu.Pin Pin3 = (Cpu.Pin)73;

    /// <summary>
    /// socket 4 (type PY), pin 4, PC9/I2C3_SDA
    /// </summary>
    public const Cpu.Pin Pin4 = (Cpu.Pin)41;

    /// <summary>
    /// socket 4 (type PY), pin 5, PA8/I2C3_SCL
    /// </summary>
    public const Cpu.Pin Pin5 = (Cpu.Pin)8;

    /// <summary>
    /// socket 4 (type PY), pin 6, PD15/PWM_CH3
    ///
    /// Pin 6 could also be used as PWM channel 3.
    /// This pin-assignment does not conform to
    /// the official Gadgeteer types and thus
    /// should not be used if it can be avoided.
    /// </summary>
    public const Cpu.Pin Pin6 = (Cpu.Pin)63;

    /// <summary>
    /// socket 4 (type PY), pin 7, PD12/TIM4_CH1
    /// </summary>
    public const Cpu.Pin Pin7 = (Cpu.Pin)60;

    /// <summary>
    /// socket 4 (type PY), pin 7, PD12/PWM_CH0
    /// </summary>
    public const Cpu.PWMChannel Pwm7 = Cpu.PWMChannel.PWM_0;

    /// <summary>
    /// socket 4 (type PY), pin 8, PD13/TIM4_CH2
    /// </summary>
    public const Cpu.Pin Pin8 = (Cpu.Pin)61;

    /// <summary>
    /// socket 4 (type PY), pin 8, PD13/PWM_CH1
    /// </summary>
    public const Cpu.PWMChannel Pwm8 = Cpu.PWMChannel.PWM_1;
    /// <summary>
    /// socket 4 (type PY), pin 9, PD14/TIM4_CH3
    /// </summary>
    public const Cpu.Pin Pin9 = (Cpu.Pin)62;

    /// <summary>
    /// socket 4 (type PY), pin 9, PD14/PWM_CH2
    /// </summary>
    public const Cpu.PWMChannel Pwm9 = Cpu.PWMChannel.PWM_2;
}
```

```
/// <summary>
/// socket 5 (type SUY)
/// </summary>
public static class Socket5
{
    /// <summary>
    /// socket 5 (type SUY), pin 3, PD2/USART5_RX
    /// </summary>
    public const Cpu.Pin Pin3 = (Cpu.Pin)50;

    /// <summary>
    /// socket 5 (type SUY), pin 4, PC6/USART6_TX
    /// </summary>
    public const Cpu.Pin Pin4 = (Cpu.Pin)38;

    /// <summary>
    /// socket 5 (type SUY), pin 5, PC7/USART6_RX
    /// </summary>
    public const Cpu.Pin Pin5 = (Cpu.Pin)39;

    /// <summary>
    /// socket 5 (type SUY), pin 6, PA15/SPI3_NSS
    /// </summary>
    public const Cpu.Pin Pin6 = (Cpu.Pin)15;

    /// <summary>
    /// socket 5 (type SUY), pin 7, PC12/SPI3_MOSI/USART5_TX
    /// </summary>
    public const Cpu.Pin Pin7 = (Cpu.Pin)44;

    /// <summary>
    /// socket 5 (type SUY), pin 8, PC11/SPI3_MISO/USART4_RX
    /// </summary>
    public const Cpu.Pin Pin8 = (Cpu.Pin)43;

    /// <summary>
    /// socket 5 (type SUY), pin 9, PC10/SPI3_SCK/USART4_TX
    /// </summary>
    public const Cpu.Pin Pin9 = (Cpu.Pin)42;

    /// <summary>
    /// COM6 is mapped to socket 5 (type U) and to on-chip UART6.
    /// </summary>
    public const string SerialPortName = "COM6";

    // COM5 is mapped to socket 5, with Rx on pin 3
    // and Tx on pin7. This pin-assignment does not
    // conform to the official Gadgeteer types and
    // thus should not be used if it can be avoided.

    // COM4 is mapped to socket 5, with Rx on pin 8
    // and Tx on pin9. This pin-assignment does not
    // conform to the official Gadgeteer types and
    // thus should not be used if it can be avoided.

    /// <summary>
    /// SPI3 is mapped to socket 5 and to on-chip SPI3.
    /// </summary>
    public const Microsoft.SPOT.Hardware.SPI.SPI_module SpiModule = Microsoft.SPOT.Hardware.SPI.SPI_module.SPI3;
}
```

```
/// <summary>
/// socket 6 (type AIOX)
/// </summary>
public static class Socket6
{
    /// <summary>
    /// socket 6 (type AIOX), pin 3, PA6/AD12_IN6
    /// </summary>
    public const Cpu.Pin Pin3 = (Cpu.Pin)6;

    /// <summary>
    /// socket 6 (type AIOX), pin 3, PA6/ADC12_IN6
    /// </summary>
    public const Cpu.AnalogChannel AnalogInput3 = Cpu.AnalogChannel.ANALOG_2;

    /// <summary>
    /// socket 6 (type AIOX), pin 4, PA5/ADC12_IN5
    /// </summary>
    public const Cpu.Pin Pin4 = (Cpu.Pin)5;

    /// <summary>
    /// socket 6 (type AIOX), pin 4, PA5/ADC12_IN5/DAC_CH1
    /// </summary>
    public const Cpu.AnalogChannel AnalogInput4 = Cpu.AnalogChannel.ANALOG_1;

    /// <summary>
    /// socket 6 (type AIOX), pin 5, PA4/ADC12_IN4/DAC1_OUT
    /// </summary>
    public const Cpu.Pin Pin5 = (Cpu.Pin)4;

    /// <summary>
    /// socket 6 (type AIOX), pin 5, PA4/ADC12_IN4/DAC1_OUT
    /// </summary>
    public const Cpu.AnalogChannel AnalogInput5 = Cpu.AnalogChannel.ANALOG_0;

    /// <summary>
    /// socket 6 (type AIOX), pin 6, PC8
    /// </summary>
    public const Cpu.Pin Pin6 = (Cpu.Pin)40;

    // pin 7 is not used

    /// <summary>
    /// socket 6 (type AIOX), pin 8, PB7/I2C1_SDA
    ///
    /// Warning: this pin has an associated pull-up resistor
    /// </summary>
    public const Cpu.Pin Pin8 = (Cpu.Pin)23;

    /// <summary>
    /// socket 6 (type AIOX), pin 9, PB6/I2C1_SCL
    ///
    /// Warning: this pin has an associated pull-up resistor
    /// </summary>
    public const Cpu.Pin Pin9 = (Cpu.Pin)22;
}
```

```
/// <summary>
/// socket 7 (type UX)
/// </summary>
public static class Socket7
{
    /// <summary>
    /// socket 7 (type UX), pin 3, PE5
    /// </summary>
    public const Cpu.Pin Pin3 = (Cpu.Pin)69;

    /// <summary>
    /// socket 7 (type UX), pin 4, PA9/USART1_TX
    /// </summary>
    public const Cpu.Pin Pin4 = (Cpu.Pin)9;

    /// <summary>
    /// socket 7 (type UX), pin 5, PA10/USART1_RX
    /// </summary>
    public const Cpu.Pin Pin5 = (Cpu.Pin)10;

    /// <summary>
    /// socket 7 (type UX), pin 6, PE6
    /// </summary>
    public const Cpu.Pin Pin6 = (Cpu.Pin)70;

    // pin 7 is MCU_RST (reset)

    // pin 8 is BOOT0

    /// <summary>
    /// socket 7 (type UX), pin 9, PB2
    ///
    /// This pin can also be used as BOOT1
    /// and is shared with pin 9 of socket 8.
    /// </summary>
    public const Cpu.Pin Pin9 = (Cpu.Pin)18;

    /// <summary>
    /// COM1 is mapped to socket 7 (type U) and to on-chip UART1.
    ///  COM1 is mainly intended for deployment and debugging.
    /// </summary>
    public const string SerialPortName = "COM1";
}
```

```
/// <summary>
/// socket 8 (type Z)
/// </summary>
public static class Socket8
{
    // pin 3 is MCU_RST (reset)

    /// <summary>
    /// socket 8 (type Z), pin 4, PA13
    ///
    /// This pin can also be used as SWDIO for debugging
    /// </summary>
    public const Cpu.Pin Pin4 = (Cpu.Pin)13;

    /// <summary>
    /// socket 8 (type Z), pin 5, PA14
    ///
    /// This pin can also be used as SWCLK for debugging
    /// </summary>
    public const Cpu.Pin Pin5 = (Cpu.Pin)14;

    // pin 6 is unused

    // pin 7 is VBAT

    // pin 8 is BOOT0

    /// <summary>
    /// socket 8 (type Z), pin 9, PB2
    ///
    /// This pin can also be used as BOOT1
    /// and is shared with pin 9 of socket 7.
    /// </summary>
    public const Cpu.Pin Pin9 = (Cpu.Pin)18;
}
```

```
/// <summary>
/// socket 9 (type DH, only on USB mainboard)
/// </summary>
public static class Socket9
{
    /// <summary>
    /// socket 9 (type DH, only on USB mainboard), pin 2, PB13/VBUS
    ///
    /// This is an exception to the rule that pins 1, 2 and 10 are
    /// not visible to the programmer. As this socket can implement
    /// the USB host role, the 5 V supply needs to be switchable.
    /// </summary>
    public const Cpu.Pin Pin2 = (Cpu.Pin)29;

    /// <summary>
    /// socket 9 (type DH, only on USB mainboard), pin 3, PE12
    /// </summary>
    public const Cpu.Pin Pin3 = (Cpu.Pin)76;

    /// <summary>
    /// socket 9 (type DH, only on USB mainboard), pin 4, PB14/OTG_HS_DM
    /// </summary>
    public const Cpu.Pin Pin4 = (Cpu.Pin)30;

    /// <summary>
    /// socket 9 (type DH, only on USB mainboard), pin 5, PB15/OTG_HS_DP
    /// </summary>
    public const Cpu.Pin Pin5 = (Cpu.Pin)31;

    /// <summary>
    /// socket 9 (type DH, only on USB mainboard), pin 6, PB12/OTG_HS_ID
    /// </summary>
    public const Cpu.Pin Pin6 = (Cpu.Pin)28;

    /// <summary>
    /// socket 9 (type DH, only on USB mainboard), pin 7, PB10
    /// </summary>
    public const Cpu.Pin Pin7 = (Cpu.Pin)26;

    // pin 8 is not used

    // pin 9 is not used
}
```

```
    /// <summary>
    /// pins used on the mainboard, not on a Gadgeteer socket
    /// </summary>
    public static class OnboardIO
    {
        /// <summary>
        /// on-board switch, PE10
        /// </summary>
        public const Cpu.Pin Button = (Cpu.Pin)74;

        /// <summary>
        /// on-board LED Green, PE11/PWM_CH5
        /// </summary>
        public const Cpu.Pin LedGreen = (Cpu.Pin)75;

        /// <summary>
        /// on-board LED Green, PE11/PWM_CH5
        /// </summary>
        public const Cpu.PWMChannel PwmGreen = Cpu.PWMChannel.PWM_5;

        /// <summary>
        /// on-board LED Red, PE13/PWM_CH6
        /// </summary>
        public const Cpu.Pin LedRed = (Cpu.Pin)77;

        /// <summary>
        /// on-board LED Red, PE13/PWM_CH6
        /// </summary>
        public const Cpu.PWMChannel PwmRed = Cpu.PWMChannel.PWM_6;

        /// <summary>
        /// on-board LED Blue, PE14/PWM_CH7
        /// </summary>
        public const Cpu.Pin LedBlue = (Cpu.Pin)78;

        /// <summary>
        /// on-board LED Blue, PE14/PWM_CH7
        /// </summary>
        public const Cpu.PWMChannel PwmBlue = Cpu.PWMChannel.PWM_7;

        /// <summary>
        /// Determine whether this is a USB mainboard or an Ethernet mainboard.
        /// </summary>
        /// <returns>True if the mainboard is USB mainboard</returns>
        public static bool IsUsbMainboard();
    }
}
```