.NET Gadgeteer Mainboard Builder’s Guide

Version 1.10 - April 19, 2013

Abstract

The *.NET Gadgeteer Mainboard Builder’s Guide* is an introduction to manufacturing requirements for building Gadgeteer mainboards. The guide explains printed circuit board (PCB) layouts, silkscreen guidelines, socket types, and mechanical design requirements for these mainboards.

Microsoft® .NET Gadgeteer extends the Microsoft commitment to .NET developers who are meeting the demand for specialized, embedded hardware components. Modules built on this platform use the .NET Micro Framework to provide direct access to hardware, without an underlying operating system, from a scaled-down version of the .NET Common Language Runtime (CLR).

Beyond embedded applications, Gadgeteer opens a large educational market for the platform. This unique solution to building small devices will be useful to researchers, secondary and tertiary educators, and hobbyists who want to build experimental devices that use Gadgeteer modules.

Open standards for hardware and software interfaces, including open source software for core libraries, enable manufacturers to build compatible mainboards and peripheral modules. The aim is to promote a thriving ecosystem of hardware that is useful to a large and diverse set of users. Early adopters who are building Gadgeteer modules can expect support from Microsoft researchers.

Document History

Date	Change
April 25, 2011	Limited partner release
August 23, 2011	Public Release 1.5
October 3, 2011	Public Release 1.6
October 31, 2011	Public Release 1.7 – updated screenshots
April 11, 2012	Public Beta release 1.8
May 15, 2012	Public Release 1.9
April 19, 2013	Public Release 1.10

Contents

Introduction

The *.NET Gadgeteer Mainboard Builder's Guide* introduces procedures for building Microsoft® .NET Gadgeteer ("Gadgeteer") mainboards. These mainboards include printed circuit board (PCB) layouts, silkscreens, socket types, and mechanical designs. Gadgeteer mainboards that comply with these guidelines are compatible with Gadgeteer modules that comply with the equivalent Module Builder's Guide guidelines, even if the mainboard and modules come from different designers and manufacturers.

The purpose of these guidelines is to help mainboard designers ensure that their mainboards are compatible with the widest range of .NET Gadgeteer modules. We recommend that designers follow these guidelines to ensure compatibility but the guidelines cannot cover every possible situation. Wherever possible, we have tried to explain the rationale behind each guideline to make it clear what issues might arise if it is not followed. We also ask module designers to treat this as an extensible document and contribute feedback to improve and clarify the process for others.

Although manufacturers can build Gadgeteer mainboards without consulting us, we welcome inquiries from hardware manufacturers who want to build Gadgeteer-compatible hardware and will provide support if we can. For further information, send an email to gadgeteer@microsoft.com.

Throughout this document, all dimensions are in millimeters (mm). Unless specified, drawings and images are not to scale.

System Requirements

To use the.NET Gadgeteer platform you need the .NET Micro Framework Software Development Kit (SDK) and the.NET Gadgeteer Core libraries. The .NET Micro Framework and Gadgeteer core libraries can be installed from Microsoft at the .NET Micro Framework website and the .NET Gadgeteer website, respectively.

The .NET Gadgeteer core libraries and the .NET Micro Framework SDK require Microsoft Visual Studio® in either of the following versions:

- Visual Studio 2012, the full-featured Visual Studio application development suite, with support for multiple programming languages.
- Visual Studio Express 2012 for Windows Desktop, a free alternative that provides lightweight, easy-to-learn and easy-to-use tools for creating Windows® applications.

Connecting Mainboards and Modules

A Gadgeteer application combines device hardware and software that provides a simple API to the functionality of the module from .NET managed code.

When you build a mainboard, the first thing you should consider is how the board will electrically interface with Gadgeteer modules.

Socket Types

Gadgeteer mainboards expose their I/O capabilities through *sockets*. Each socket is a 10-way connector, with pins labeled 1 through 10. For detailed information about connector specifications, see **Connectors** later in this document.

Mainboard sockets support one or more different *socket types*. Each socket type is represented by a letter. When a mainboard socket is labeled with a socket type letter, it guarantees a particular set of electrical connections and interfaces on the socket's pins. The following table shows the .NET Gadgeteer Socket Types Definitions. In consultation with mainboard and module manufacturers, new socket types may be added in the future.

Socket Types Table (Version 17)

TYPE	LETTER	PIN 1	PIN 2	PIN 3	PIN 4	PIN 5	PIN 6	PIN 7	PIN 8	PIN 9	PIN 10
3 GPIO	X	+3.3V	+5V	GPIO!	GPIO	GPIO	[UN]	[UN]	[UN]	[UN]	GND
7 GPIO	Y	+3.3V	+5V	GPIO!	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GND
Analog In	A	+3.3V	+5V	AIN (G!)	AIN (G)	AIN	GPIO	[UN]	[UN]	[UN]	GND
CAN	C	+3.3V	+5V	GPIO!	TD (G)	RD (G)	GPIO	[UN]	[UN]	[UN]	GND
USB Device	D	+3.3V	+5V	GPIO!	D-	D+	GPIO	GPIO	[UN]	[UN]	GND
Ethernet	E	+3.3V	+5V	[UN]	LED1 (OPT)	LED2 (OPT)	TX D-	TX D+	RX D-	RX D+	GND
SD Card	F	+3.3V	+5V	GPIO!	DAT0	DAT1	CMD	DAT2	DAT3	CLK	GND
USB Host	H	+3.3V	+5V	GPIO!	D-	D+	[UN]	[UN]	[UN]	[UN]	GND
I²C	I	+3.3V	+5V	GPIO!	[UN]	[UN]	GPIO	[UN]	SDA	SCL	GND
UART+ Handshaking	K	+3.3V	+5V	GPIO!	TX (G)	RX (G)	RTS	CTS	[UN]	[UN]	GND
Analog Out	O	+3.3V	+5V	GPIO!	GPIO	AOUT	[UN]	[UN]	[UN]	[UN]	GND
PWM	P	+3.3V	+5V	GPIO!	[UN]	[UN]	GPIO	PWM (G)	PWM (G)	PWM	GND
SPI	S	+3.3V	+5V	GPIO!	GPIO	GPIO	GPIO	MOSI	MISO	SCK	GND
Touch	T	+3.3V	+5V	[UN]	YU	XL	YD	XR	[UN]	[UN]	GND
UART	U	+3.3V	+5V	GPIO!	TX (G)	RX (G)	GPIO	[UN]	[UN]	[UN]	GND
LCD 1	R	+3.3V	+5V	LCD R0	LCD R1	LCD R2	LCD R3	LCD R4	LCD VSYNC	LCD HSYNC	GND
LCD 2	G	+3.3V	+5V	LCD G0	LCD G1	LCD G2	LCD G3	LCD G4	LCD G5	BACK-LIGHT	GND
LCD 3	B	+3.3V	+5V	LCD B0	LCD B1	LCD B2	LCD B3	LCD B4	LCD EN	LCD CLK	GND
Manufacturer Specific	Z	+3.3V	+5V	[MS]	[MS]	[MS]	[MS]	[MS]	[MS]	[MS]	GND
DaisyLink Downstream*	*	+3.3V	+5V	GPIO!	GPIO	GPIO	[MS]	[MS]	[MS]	[MS]	GND

LEGEND:

GPIO	A general-purpose digital input/output pin, operating at 3.3 Volts.
(G)	In addition to another functionality, a pin that is also usable as a GPIO.

(OPT)	A socket type that is optionally supported by a mainboard or a module.
[UN]	Modules must not connect to this pin if using this socket type. Mainboards can support multiple socket types on one socket, as long as individual pin functionalities overlap in a compatible manner. A pin from one socket type can overlap with a [UN] pin of another.
[MS]	A manufacturer-specific pin. See the documentation from the manufacturer of the board.
!	Interrupt-capable and software pull-up capable GPIO (the pull-up is switchable and in the range of 10,000 to 100,000 Ω).
*****	Socket type * should not appear on a mainboard, only on DaisyLink modules. The [MS] pins on this socket type can optionally support reflashing the firmware on the module.

NOTES:

- All pins must be at least 3.6V tolerant.
- The low logic input is 0V minimum, 0.4V maximum. Logic input high is 0.7*Vdd min, Vdd+0.2 maximum.
- A module must not assume that the mainboard can pull up or pull down GPIOs using built in pull-ups/pull-downs. If a pull-up or pull-down resistor is required on a GPIO, the module should incorporate it into the PCB.
- A module's screen printing must list both X and Y if it is compatible with both (the mainboard will say only “X” or ”Y,” but not both). Socket-compatibility labels are case sensitive.
- If pins are shared across multiple sockets, users of shared bus socket types (such as I or S) must not impede use by other devices. For example, type S modules must respect the chip select signal.
- For socket type I, mainboards should include I^2C bus pull-ups of 2,200 Ω on both SDA and SCL pins. Modules must *not* include pull-ups on these lines.
- For the DaisyLink protocol, which uses socket type X, pull ups should not be on the mainboard. For details of module pull-up requirements, see the DaisyLink specification.
- For socket types C, K, and U, the rationale for having GPIO-capable function pins is to allow module makers to implement transmit-only or receive-only modules with the flexibility of an additional GPIO pin.
- For socket type A and P, the rationale for having two of the three function pins with (G) and one without (G) is to allow modules to use 1, 2, or 3 Analog Input / PWM pins, and have the flexibility of GPIOs for other pins.
- For DaisyLink modules (using type X), pin 3 is for the DaisyLink neighbor bus, pin 4 is used for I^2C SDA, pin 5 is used for I^2C SCL. See the DaisyLink spec and Appendix 1, both in this document, for more details.
- Note for UART ‘U’ and UART+Handshaking ‘K’ socket types: TX (Pin 3) is data from the Mainboard to the Module, and RX (Pin 4) is data from the Module to the Mainboard. These are idle high (3.3V).
  RTS (Pin 6) is an output from the Mainboard to the Module indicating that the module may send data, and CTS (Pin 7) is an output from the Module to the Mainboard indicating that the Mainboard may send data.
  RTS/CTS lines are “not ready” if high (3.3V) and “ready” if low (0V).

Mapping Mainboard Sockets to Socket Types

Each mainboard socket can support multiple socket types. The mainboard manufacturer chooses how many sockets to provide and with which socket types each socket is compatible, based on the underlying capabilities of the CPU and other hardware that are present on the mainboard.

These choices affect the following:

- Which modules are compatible with the mainboard.
  If a module requires a socket type that the mainboard does not support, the module cannot be used with that mainboard.
- Which *combinations* of modules can be simultaneously plugged into a mainboard. This affects the types of devices that users can build. For example, if a mainboard has only one socket that supports Type S (SPI), then a user cannot simultaneously plug in multiple Type S modules to build a Gadgeteer-based device.

Each module typically supports a single function type because of the underlying interface that the electronics of that module support, such as universal asynchronous receiver-transmitter—UART—(U), SPI (S), or Analog Input (A). If the module simply uses a few digital I/Os, socket Type X should be used. Modules can also use DaisyLink Protocol, which is a custom Gadgeteer interface that also uses socket Type X. This protocol enables chaining many devices from a single socket. From a mainboard manufacturer's point of view, DaisyLink is implemented by the Gadgeteer core libraries, so supporting DaisyLink can be as simple as making one or more sockets compatible with socket Type X. Socket-compatibility labels are case sensitive.

If you have questions about how best to group socket types onto sockets, send an email to gadgeteer@microsoft.com.

Pull-up Specifications

Pull-ups for the Hardware I^{2}C bus (Type I) should be on the mainboard. Pull-ups for DaisyLink modules should be on the modules but must be switchable. The pins should be floating by default. Only if the module detects that it is the last module in the DaisyLink chain—which occurs during the DaisyLink protocol—should the module turn on these pull-ups. The module does this either by using discrete field-effect transistors FETs to switch the pull-ups or by tying them to a processor pin and driving that pin high (if the processor can sink the pull-up current).

Firmware Requirements

The Gadgeteer core libraries and third party module assemblies include functionality that builds on.NET Micro Framework assemblies. Mainboards must include the following assemblies in their porting kit:

Microsoft.SPOT.Graphics
Microsoft.SPOT.Hardware
Microsoft.SPOT.Hardware.SerialPort
Microsoft.SPOT.IO
Microsoft.SPOT.Native
Microsoft.SPOT.Net
Microsoft.SPOT.Net.Security
Microsoft.SPOT.TinyCore
mscorlib
System
System.Http
System.IO

Software Provided to Users

Mainboard manufacturers should provide users with the following software:

- USB device drivers for all .NET Micro Framework boards.
- A class library (DLL) that provides software interface abstraction for the mainboard hardware.
- One or more C# project templates for projects that use the mainboard.

This software should be provided by using an installer that gives users a simple installation experience. Ideally, the installer should also install the following prerequisites automatically if they are missing: the .NET Micro Framework SDK and the Gadgeteer core libraries to be published on the .NET Gadgeteer website.

Please refer to the following section **Visual Studio Mainboard Template Project** for instructions on how to implement this software easily using templates provided by Microsoft.

Mainboard Class Library

To enable the Gadgeteer core libraries to interface with a mainboard, a manufacturer must provide a library (DLL) that implements the Gadgeteer.Mainboard interface. This interface can be found in the Gadgeteer core library.

The class's fully-qualified name should be ManufacturerName.Gadgeteer.MainboardName, as follows:

```
namespace MSR.Gadgeteer
{
    public class ExampleMainboard : Gadgeteer.Mainboard
    {
```

Similarly, the assembly should be named ManufacturerName.Gadgeteer.MainboardName.dll.

Visual Studio Mainboard Template Project

The mainboard template installed with the GadgeteerBuilderTemplates installer found at http://gadgeteer.codeplex.com/ provides an easy way to build software for a mainboard that is compliant with the specifications in this document. The template generates an installer (MSI) that can be distributed to end users. It also generates an installation-merge module (MSM) that can be used with the Kit Template (also found in the GadgeteerBuilderTemplates installer) to make kit installers that include modules, mainboards, templates, and other components that might be needed in a development kit.

This version of the Mainboard Builder's Guide is specific to Gadgeteer 2.42.700 and later. The most up-to-date information about building .NET Gadgeteer mainboards is included in the ReadMe file included with the module template project. Version 2.42.700 of the core libraries support mainboards with built-in modules and modules that only work on particular mainboards. Version 2.42.700 fragments some elements of the core, such as the Networking, Serial, and DisplayModule classes. Refer to the ReadMe.txt for information pertinent to each version.

The templates can build modules for both version 2.42 and the earlier version 2.41. You can implement either or both versions. Implementing both versions will enable your mainboard to work with more modules.

Both Visual Studio and Visual Studio Express support the Mainboard template, though only Visual Studio supports the Kit template. To build the installer MSI automatically, the template requires WiX Toolset 3.5 (or newer) to be installed. The WiX Toolset is available at http://wixtoolset.org/.

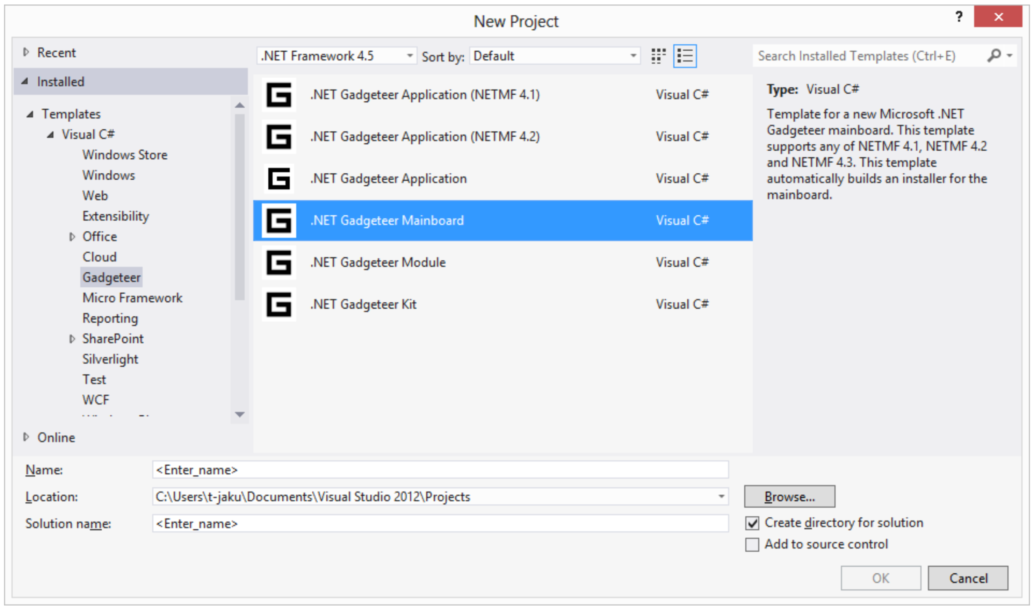
Important: Be sure to install WiX before you create a project to build mainboard or kit installers.

A Readme file installed with the template also contains the instructions that are detailed in this section. The Readme file installed with the template contains updates of the instructions in this section. The templates currently support modules built on either or both 2.42 and 2.41 assemblies. The 4.3 release of the .NET Micro Framework facilitates this upgrade to the .NET Gadgeteer core libraries, but there are numerous 2.41 modules in production. .NET Gadgeteer developers can implement drivers for either 2.41 or 2.42 assemblies or for both versions of the assemblies.

You can create a Visual Studio project that includes the mainboard software template by following these steps.

To create a Visual Studio project that includes the mainboard software template

1. Select **New Project** from the File menu.
2. Select **Gadgeteer** from the Visual C# installed templates.
3. Select **.NET Gadgeteer – Mainboard Template** from the options visible, as shown in the following illustration.
4. Name the project and click OK.


New Project
Recent
Installed
Templates
Visual C#
Windows Store
Windows
Web
Extensibility
Office
Cloud
Gadgeteer
Micro Framework
Reporting
SharePoint
Silverlight
Test
WCF
Online
.NET Framework 4.5
Sort by: Default
Search Installed Templates (Ctrl+E)
.NET Gadgeteer Application (NETMF 4.1) Visual C#
.NET Gadgeteer Application (NETMF 4.2) Visual C#
.NET Gadgeteer Application Visual C#
.NET Gadgeteer Mainboard Visual C#
.NET Gadgeteer Module Visual C#
.NET Gadgeteer Kit Visual C#
Browse...
OK
Cancel

Type: Visual C#

Template for a new Microsoft .NET Gadgeteer mainboard. This template supports any of NETMF 4.1, NETMF 4.2 and NETMF 4.3. This template automatically builds an installer for the mainboard.

Name: <Enter_name>

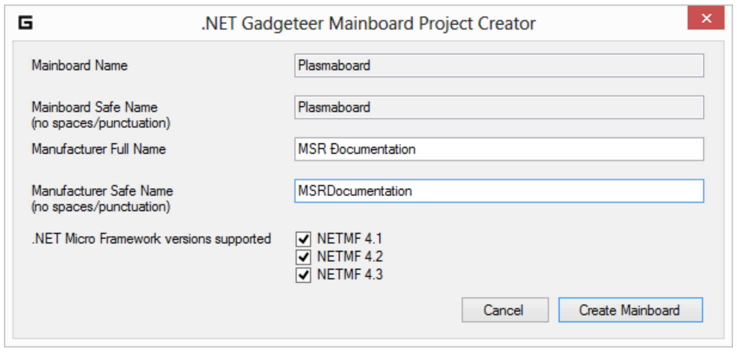
Location: C:\Users\t-jaku\Documents\Visual Studio 2012\Projects

Solution name: <Enter_name>

☑ Create directory for solution

☐ Add to source control

5. The dialog that opens provides textboxes for Manufacturer Full Name and Manufacturer Safe Name. It also allows you to specify whether your mainboard targets version 4.1 or 4.2 of the .NET Micro Framework, or both versions.



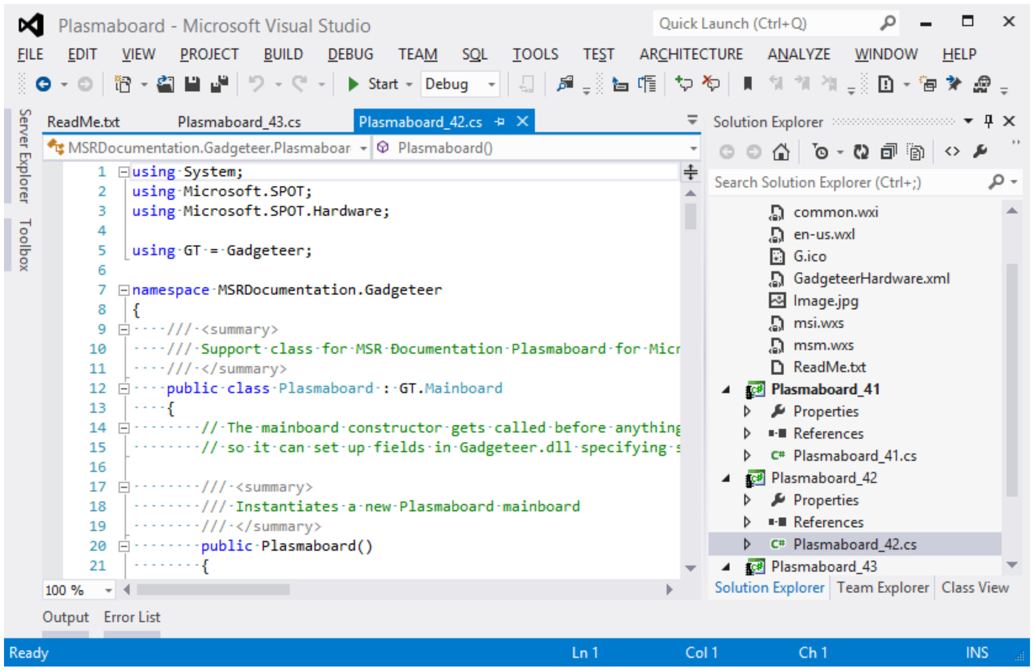
Mainboard Name	Plasmaboard
Mainboard Safe Name (no spaces/punctuation)	Plasmaboard
Manufacturer Full Name	MSR Documentation
Manufacturer Safe Name (no spaces/punctuation)	MSRDocumentation
.NET Micro Framework versions supported	☑ NETMF 4.1 ☑ NETMF 4.2 ☑ NETMF 4.3

This will replace matches in: [MainboardName].cs, common.wxi, GadgeteerHardware.xml, and AssemblyInfo.cs. You can skip the matches in the Readme file.

Implementing the Mainboard Driver

You do not have to write assembly-language code or do low-level programming to implement the software driver for a Gadgeteer mainboard. The mainboard template creates a file named [MainboardName].cs. This file uses the C# programming language and the features of managed code. There are comments and examples in the [MainboardName].cs file to assist you with the implementation of the software for your module.

The following illustration shows a view of the [MainboardName].cs file in which a hypothetical PlasmaBoard mainboard is implemented.



```
using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

using GT = Gadgeteer;

namespace MSRDocumentation.Gadgeteer
{
    /// <summary>
    /// Support class for MSR Documentation Plasmaboard for Micr
    /// </summary>
    public class Plasmaboard : GT.Mainboard
    {
        // The mainboard constructor gets called before anything
        // so it can set up fields in Gadgeteer.dll specifying s

        /// <summary>
        /// Instantiates a new Plasmaboard mainboard
        /// </summary>
        public Plasmaboard()
        {
```

The `PlasmaBoard.cs` files created by the template includes example code that shows how to create, configure, and register each socket on the mainboard with Gadgeteer.dll. For more information, see the heading Socket Types in this document.

The triple-slash "///" comments shown in the following code segment are used by the build process to create an XML file named MSRDocumentation.Gadgeteer.PlasmaBoard.XML that supports IntelliSense and documentation for the software interface.

The following code shows the constructor for the hypothetical PlasmaBoard mainboard.

```
public class Plasmaboard : GT.Mainboard
{
    // The mainboard constructor gets called before anything else in
Gadgeteer (module constructors, etc),
    // so it can set up fields in Gadgeteer.dll specifying socket types
supported, etc.

    /// <summary>
    /// Instantiates a new Plasmaboard mainboard
    /// </summary>
    public Plasmaboard()
    {
        // uncomment the following if you support NativeI2CWriteRead for
faster DaisyLink performance
        // otherwise, the DaisyLink I2C interface will be supported in
Gadgeteer.dll in managed code.
        GT.Socket.SocketInterfaces.NativeI2CWriteReadDelegate nativeI2C =
null; // new
```

```
GT.Socket.SocketInterfaces.NativeI2CWriteReadDelegate(NativeI2CWriteRead)
;

        GT.Socket socket;

        // For each socket on the mainboard, create, configure and
register a Socket object with Gadgeteer.dll
        // This specifies:
        // - the SupportedTypes character array matching the list on the
mainboard
        // - the CpuPins array (indexes [3] to [9].  [1,2,10] are
constant (3.3V, 5V, GND) and [0] is unused.  This is normally based on an
enumeration supplied in the NETMF port used.
        // - for other functionality, e.g. UART, SPI, etc, properties in
the Socket class are set as appropriate to enable Gadgeteer.dll to access
this functionality.
        // See the Mainboard Builder's Guide and specifically the Socket
Types specification for more details
        // The two examples below are not realistically implementable
sockets, but illustrate how to initialize a wide range of socket
functionality.

        // This example socket 1 supports many types
        // Type 'D' - no additional action
        // Type 'I' - I2C pins must be used for the correct CpuPins
        // Type 'K' and 'U' - UART pins and UART handshaking pins must be
used for the correct CpuPins, and the SerialPortName property must be
set.
        // Type 'S' - SPI pins must be used for the correct CpuPins, and
the SPIModule property must be set
        // Type 'X' - the NativeI2CWriteRead function pointer is set
(though by default "nativeI2C" is null)
        socket = GT.Socket.SocketInterfaces.CreateNumberedSocket(1);
        socket.SupportedTypes = new char[] { 'D', 'I', 'K', 'S', 'U', 'X'
};
        socket.CpuPins[3] = (Cpu.Pin)1;
        socket.CpuPins[4] = (Cpu.Pin)52;
        socket.CpuPins[5] = (Cpu.Pin)23;
        socket.CpuPins[6] = (Cpu.Pin)12;
        socket.CpuPins[7] = (Cpu.Pin)34;
        socket.CpuPins[8] = (Cpu.Pin)5;
        socket.CpuPins[9] = (Cpu.Pin)7;
        socket.NativeI2CWriteRead = nativeI2C;
        socket.SerialPortName = "COM1";
        socket.SPIModule = SPI.SPI_module.SPI1;
        GT.Socket.SocketInterfaces.RegisterSocket(socket);

        // This example socket 2 supports many types
        // Type 'A' - AnalogInput3-5 properties are set and
GT.Socket.SocketInterfaces.SetAnalogInputFactors call is made
        // Type 'O' - AnalogOutput property is set
        // Type 'P' - PWM7-9 properties are set
        // Type 'Y' - the NativeI2CWriteRead function pointer is set
(though by default "nativeI2C" is null)
        socket = GT.Socket.SocketInterfaces.CreateNumberedSocket(2);
        socket.SupportedTypes = new char[] { 'A', 'O', 'P', 'Y' };
        socket.CpuPins[3] = (Cpu.Pin)11;
        socket.CpuPins[4] = (Cpu.Pin)5;
```

```
            socket.CpuPins[5] = (Cpu.Pin)3;
            socket.CpuPins[6] = (Cpu.Pin)66;
            // Pin 7 not connected on this socket, so it is left unspecified
            socket.CpuPins[8] = (Cpu.Pin)59;
            socket.CpuPins[9] = (Cpu.Pin)18;
            socket.NativeI2CWriteRead = nativeI2C;
            socket.AnalogOutput = new Plasmaboard_AnalogOut((Cpu.Pin)14);
            GT.Socket.SocketInterfaces.SetAnalogInputFactors(socket, 1, 2,
10);
            socket.AnalogInput3 = Cpu.AnalogChannel.ANALOG_2;
            socket.AnalogInput4 = Cpu.AnalogChannel.ANALOG_3;
            socket.AnalogInput5 = Cpu.AnalogChannel.ANALOG_1;
            socket.PWM7 = Cpu.PWMChannel.PWM_3;
            socket.PWM8 = Cpu.PWMChannel.PWM_0;
            socket.PWM9 = Cpu.PWMChannel.PWM_2;
            GT.Socket.SocketInterfaces.RegisterSocket(socket);

        }
```

The example code that is provided by the template is designed to assist you in the implementation of a driver in managed code for the mainboard. For more information, see the comments in your [MainboardName].cs file.

Building the project in the **Debug** configuration creates the mainboard class library (DLL). The build process takes longer in **Release** mode, so using the **Debug** configuration until the implementation is complete and tested will save time. If you see the error "The system cannot find the file..." try **Rebuild** rather than **Build**.

The **Debug** configuration does not build the installer .msi or .msm files. In Visual Studio, you can turn off the installer build and speed up the build process. Go to Menu->**Project**->[MainboardName] **Properties**->**Build** tab, and select the "Define DEBUG constant". To build the installers, you must change the configuration to **Release** mode.

Designer F1 Help

The .NET Gadgeteer Designer includes an XSL transform that creates html reference pages for F1 help. The Designer displays these reference pages when the user clicks on a module or mainboard in the Designer and pushes the F1 key.

Both Intellisense and .NET Gadgeteer Designer F1 help originate in triple slash comments /// in C# source files. The .NET Gadgeteer Designer F1 reference pages include remarks and code samples that Intellisense does not display.

To use this feature, include triple slash nodes for <remarks/> and/or <example/> and <code/> in the source code for your module or mainboard. You can use only remarks or both remarks and code examples.

The following example shows triple slash source comments with example code for a mainboard named Plasmaboard. Intellisense will display only the <summary/> node, but the .NET Gadgeteer Designer contains embedded XSL that transforms the <example/> and <code/> nodes into html format for display by F1 help.

```
using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

using GT = Gadgeteer;

namespace ManufacturerName.Gadgeteer
{
    /// <summary>
    /// Support class for Microsoft Plasmaboard for Microsoft .NET
Gadgeteer
    /// </summary>
    public class Plasmaboard : GT.Mainboard
    {
        // The mainboard constructor gets called before anything else in
Gadgeteer (module constructors, etc),
        // so it can set up fields in Gadgeteer.dll specifying socket
types supported, etc.
        ...
```

HelpUrl Attribute of Mainboard Template

If you supply triple slash comments in your module or mainboard source file, the F1 help will support a link to additional support information for your module or mainboard. The support link is displayed along with F1 help based on the /// comments in source code, so it will not work if F1 help is not in use. To supply the support link along with F1 help, assign the HelpUrl attribute of the Module Software Template. The HelpUrl attribute is contained by the <ModuleDefinition/> node of the GadgeteerHardware.xml file that the builder templates provide.

The following excerpt from the GadgeteerHardware.xml file shows the HelpUrl attribute.

```
<?xml version="1.0" encoding="utf-8" ?>
<GadgeteerDefinitions
xmlns="http://schemas.microsoft.com/Gadgeteer/2011/Hardware">
  <MainboardDefinitions>
    <!-- This mainboard definition should be filled in.  Mouse over any
attribute name to get
    more help about that attribute. -->
    <MainboardDefinition
            Name="Plasmaboard"
            Type="ManufacturerName.Gadgeteer.Plasmaboard"
            HardwareVersion="1.1"
            Image="Resources\Image.jpg"
            BoardWidth="77"
            BoardHeight="67"
            MinimumGadgeteerCoreVersion="2.42.500"
            HelpUrl="http://MicrosoftResearch.com/sample.htm"
            >
```

Testing the Mainboard Software

Testing is most easily accomplished by adding a new .NET Gadgeteer Application project to the same solution as is used to build the mainboard software.

To add a new Gadgeteer project and reference the module library

1. Right click on the module Solution in **Solution Explorer**.
2. Click Add->New Project.
3. Select .NET Gadgeteer Application.
4. In the Solution Explorer, right click the test project.
5. Click **Add Reference**.
6. Browse to the DLL for your mainboard and add a reference to it.
7. In the Program.cs file, replace `InsertMainboardTypeHere()` with your mainboard type [Manufacturer Name].Gadgeteer.[Mainboard Name].

The following example instantiates a hypothetical mainboard named PlasmaBoard by the manufacturer name MSR.

```
    static void InitializeMainboard()
    {
       // Specify the mainboard type below.
       // You may need to use the menu item
       // "Project->Add Reference" to add a reference to the
       // mainboard's DLL library.
       Mainboard = new MSR.Gadgeteer.PlasmaBoard();
    }
```

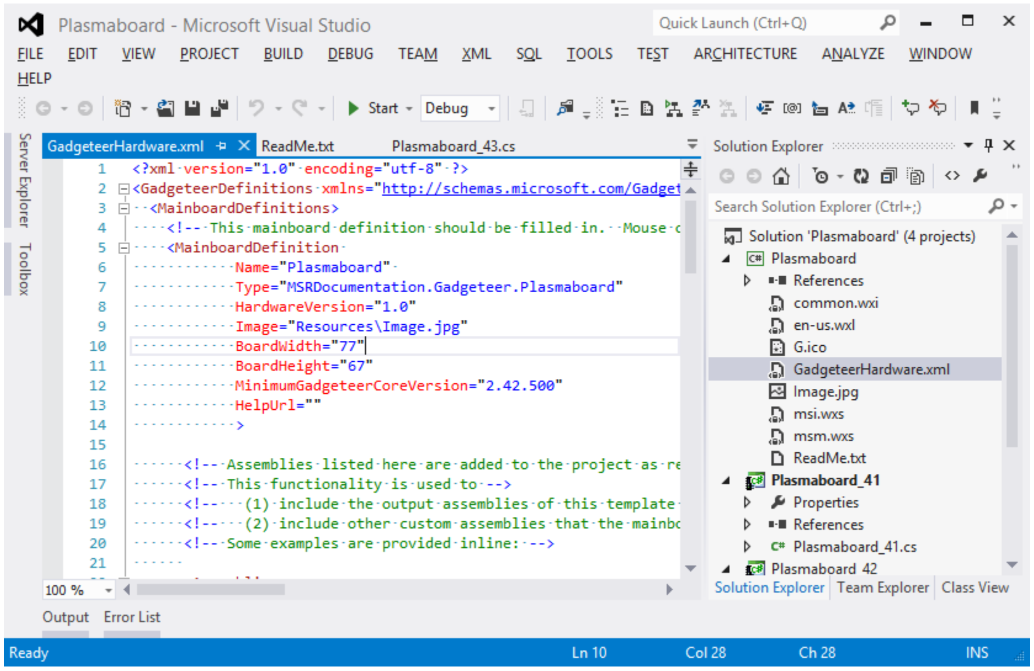
Building the MSI and MSM Installers

When the mainboard implementation is complete and tested, you are ready to build the installers for the module software. The template supports .msi and .msm installers. The output of the template includes both an .msi installer for the mainboard and .msm installer. The .msm installer can be used for kit installers that incorporate one or more modules and a mainboard. Kit installers are described in this document under the heading **Visual Studio Kit Template Project**.

To build the mainboard installer .msi and .msm files:

1. Edit the GadgeteerHardware.xml file that specifies information about your mainboard's assembly name and pin mappings. Change the file as needed beyond the changes to the manufacturer, description, and fully qualified assembly name. IntelliSense information is available as you mouse over each element of the xml document. For more information about pin and socket types, see the heading **Connecting Modules to Mainboards** heading in this document.

The following illustration shows the `GadgeteerHardware.xml` file in Visual Studio.



```
<?xml version="1.0" encoding="utf-8" ?>
<GadgeteerDefinitions xmlns="http://schemas.microsoft.com/Gadget
  <MainboardDefinitions>
    <!-- This mainboard definition should be filled in.  Mouse c
    <MainboardDefinition
            Name="Plasmaboard"
            Type="MSRDocumentation.Gadgeteer.Plasmaboard"
            HardwareVersion="1.0"
            Image="Resources\Image.jpg"
            BoardWidth="77"
            BoardHeight="67"
            MinimumGadgeteerCoreVersion="2.42.500"
            HelpUrl=""
            >

      <!-- Assemblies listed here are added to the project as re
      <!-- This functionality is used to -->
      <!--   (1) include the output assemblies of this template
      <!--   (2) include other custom assemblies that the mainbo
      <!-- Some examples are provided inline: -->
      
```

2. Change the Resources\Image.jpg file to a **good** quality top-down image of the mainboard with the socket side facing up. The image should be cropped tight, no margin, in the same orientation, not rotated, as the width and height specified in GadgeteerHardware.xml specification. The **Resources** folder included by the .Net Gadgeteer Module Template includes a blank Image.jpg file.

3. Edit the Setup\common.wxi file to specify parameters for the installer. The parameters are described in the .wxi file. You don't need to edit any other file in the **Setup** directory.

4. Change the build configuration to **Release**, and build the mainboard project. When the build completes, you should find the .msi and .msm files in the \bin\Release\Installer directory. The example used for illustrations in this example builds files named `PlasmaBoard.msi` and `PlasmaBoard.msm`.

Making Changes to Mainboard Software

If you make want to release a new version of your mainboard, make sure to change the version number in Setup\`common.wxi`. Otherwise, the auto-generated installer will not be able to upgrade the older version correctly, and an error message will result. It is also necessary to change the versions in Properties\`AssemblyInfo.cs`, else the existence of the newer assembly will not be picked up by Visual Studio.

If you want to change the name of your mainboard, be sure to search all the files for instances of the name. As explained in this document, the software mainboard name should match the name printed on the mainboard itself. The `ManufacturerName` should match the manufacturer name printed on the mainboard (remove any spaces/punctuation).

Comments in `AssemblyInfo.cs` describe the version numbers, as shown below.

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

[assembly: AssemblyTitle("$manufacturer$.Gadgeteer.$safeprojectname$")]
[assembly: AssemblyDescription("Driver for $projectname$ mainboard made
by $registeredorganization$ for Microsoft .NET Gadgeteer")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("$registeredorganization$")]
[assembly: AssemblyProduct("$projectname$")]
[assembly: AssemblyCopyright("")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// This is the assembly version.  It is often useful to keep this
constant between major releases where the API changes
// since a dll relying on this dll will need to be recompiled if this
changes.
// Suggestion: use a version number X.Y.0.0 where X and Y indicate major
version numbers.
[assembly: AssemblyVersion("1.0.0.0")]

// These numbers must be changed whenever a new version of this dll is
released, to allow upgrades to proceed correctly.
// Suggestion: Use a version number X.Y.Z.0 where X.Y.Z is the same as
the installer version found in Setup\common.wxi
[assembly: AssemblyFileVersion("1.0.0.0")]
[assembly: AssemblyInformationalVersion("1.0.0.0")]
```

Similar comments in the `common.wxi` describe how to change the version.

```
<!-- Change this whenever building a new installer.  The fourth number is
ignored, so change one of the top three.
  Otherwise, users will not be able to upgrade properly; Windows
Installer will exit with an error instead of upgrading. -->
  <!-- Also change the version numbers in Properties/AssemblyInfo.cs -->
  <?define MainboardSoftwareVersion = "1.0.0.0" ?>
```

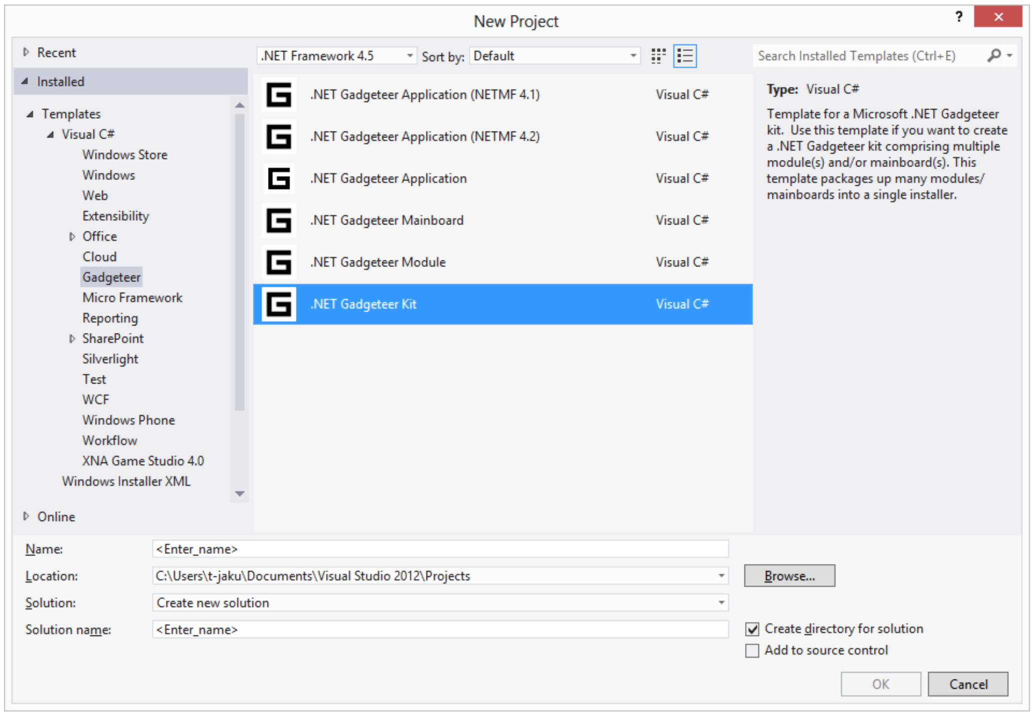
Visual Studio Kit Template Project

The kit template provides a way to build a kit installer that incorporates one or more mainboards and modules for Microsoft .NET Gadgeteer as well as one or more project templates oriented to the parts in the kit. Using this template auto-generates an installer (MSI) that can be distributed to end users.

To create a Visual Studio project that includes the kit software template

1. select **New Project** from the **File** menu.
2. Select **Gadgeteer** from the Visual C# installed templates.

3. Select **.NET Gadgeteer – Kit Template** from the options visible, as shown in the following illustration.
4. Name the project, and click OK.



.NET Framework 4.5 Sort by: Default

.NET Gadgeteer Application (NETMF 4.1)	Visual C#
.NET Gadgeteer Application (NETMF 4.2)	Visual C#
.NET Gadgeteer Application	Visual C#
.NET Gadgeteer Mainboard	Visual C#
.NET Gadgeteer Module	Visual C#
.NET Gadgeteer Kit	Visual C#

Type: Visual C#

Template for a Microsoft .NET Gadgeteer kit. Use this template if you want to create a .NET Gadgeteer kit comprising multiple module(s) and/or mainboard(s). This template packages up many modules/ mainboards into a single installer.

Name: <Enter_name>
Location: C:\Users\t-jaku\Documents\Visual Studio 2012\Projects
Solution: Create new solution
Solution name: <Enter_name>

✓ Create directory for solution
☐ Add to source control

Including a Getting Started Guide

You can include a *Getting Started Guide* with .NET Gadgeteer kit installer to introduce the mainboard and modules in the kit. Write the guide as an HTML document, named `GettingStarted.htm`, and place it along with any images used by the document in the directory at [Solution]\Getting Started Guide\.

Preparing the Kit Installer

Edit the file `en-us.wxl` to specify your company name in the `DistributorFull` element. You can also change the kit name in this file.

Edit the installer `msi.wxs` file in the following ways:

- List the getting-started-guide image files under the comment line: <!-- List all images used in the getting started guide here -->. If you don't have a getting started guide, remove the WiX elements relating to the *Getting Started Guide*.
- Follow the instructions to reference merge modules (msm files) for the mainboard(s) and module(s) you are including in this kit under the comment line: <!-- List merge modules for mainboard(s) and module(s) -->.
- Follow the instructions to reference merge modules under the comment line: <!-- List all merge modules above here -->

Building the Kit

The msi.wxs file relies on merge module (MSM) files being in the right locations. The .msm files were built by the mainboard and module projects, using the respective .NET Gadgeteer templates. These templates only build the MSM file when the build configuration is **Release**. So, one way to build the Kit Installer is to open up each of the individual mainboard/module solutions, before building the kit template, and build them under **Release** configuration to make the MSMs.

Another convenient way to rebuild the kit is to add the mainboard/module projects to the Kit solution.

To add the mainboard/module projects to the Kit solution

1. Right click on the solution in **Solution Explorer** and choose "Add->Existing Project", to link to each of the projects of each module/mainboard.
2. Then, make sure the project dependencies are set up right (Menu->Project->Project Dependencies). Ensure that each module/mainboard installer project relies on its driver projects, and ensure that the Kit WiX project relies on all the module/mainboard merge module installer projects.

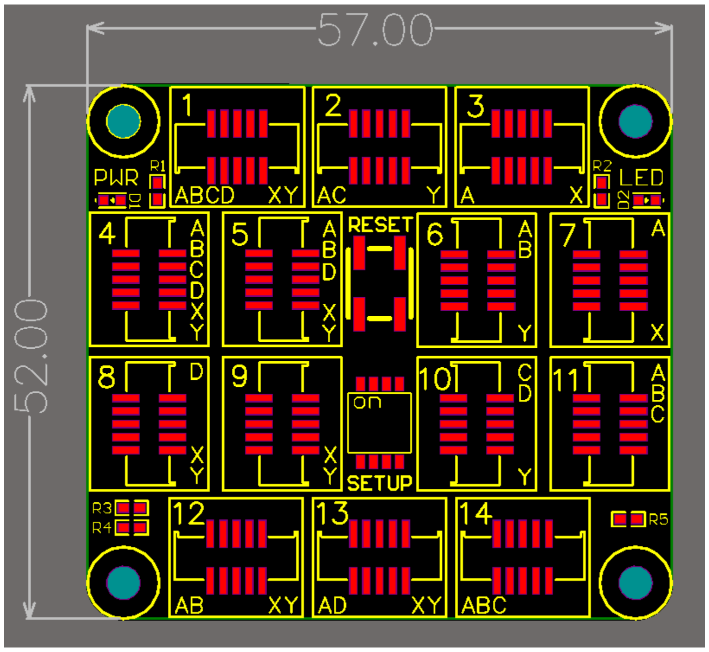
The kit installer will build in **Debug** or **Release** configuration, so one way to ensure that you don't accidentally rebuild all your MSMs is to stay in **Debug** configuration.

Printed Circuit Board Layout and Silkscreen Guidelines

The following guidelines specify recommended practices for PCB layout and silkscreen design.

Sample Design

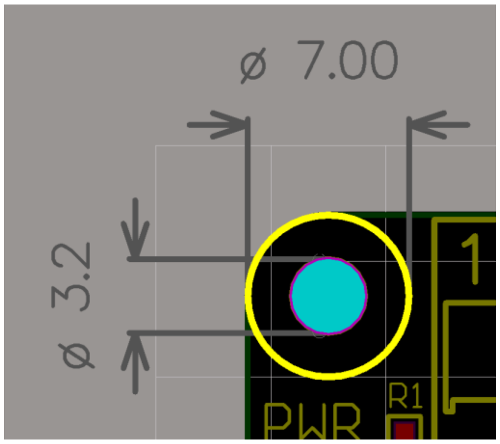
The mainboard in the following illustration shows a PCB that has been implemented according to the guidelines in this document. The example mainboard dimensions are 57x52 mm, but actual mainboard designs may be of any size or aspect ratio.

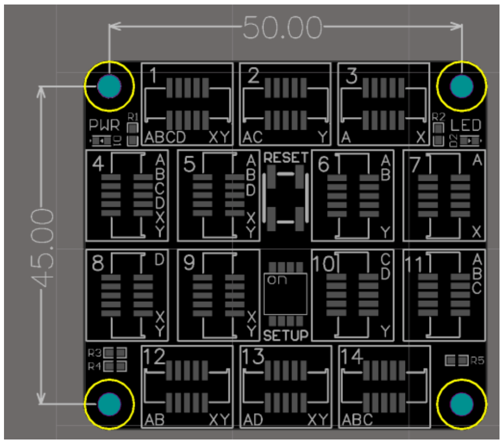


Mounting Holes

The following illustration shows that a mounting hole should be placed at each corner of the board. The mounting holes should have a 3.2-mm diameter, with a 7-mm-diameter component keep-out area around them. The keep-out area should be clearly delimited in the silkscreen on both sides of the PCB.

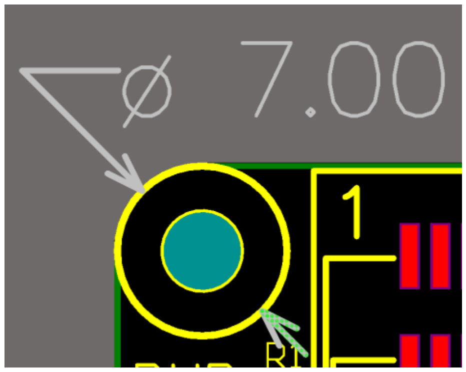
All mounting holes should be placed in a 5-mm grid, that is, the distance between adjacent holes should be a multiple of 5 mm.





Corners

Corners should be rounded, with a 7-mm-diameter curve that is concentric with a mounting hole’s keep-out area, as shown in the following illustration.



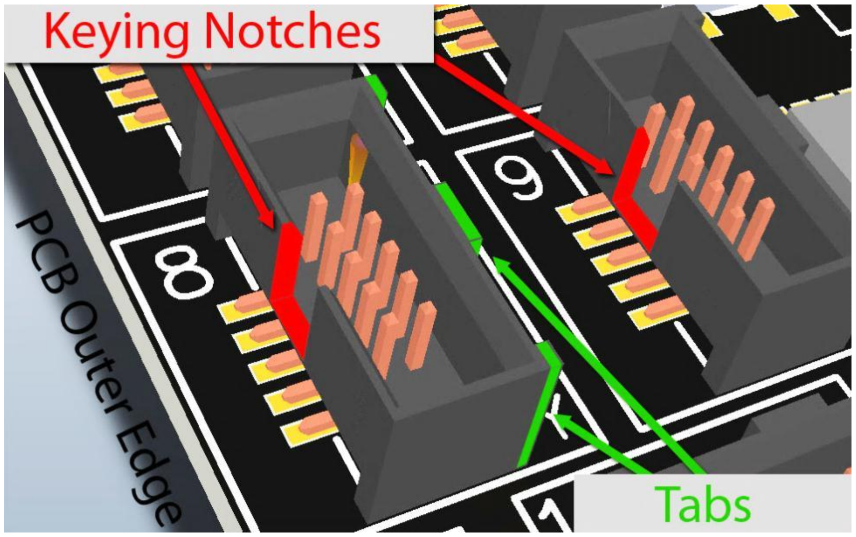
Connectors

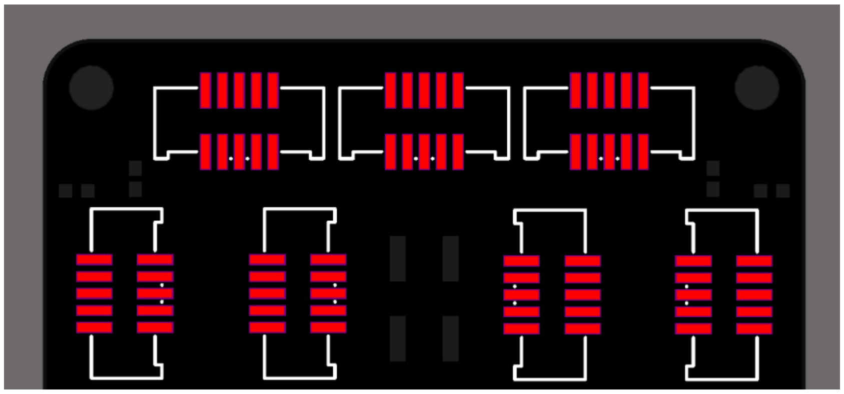
Connectors must be compatible with the reference Samtec part SHF-105-01-L-D-SM. This section contains two illustrations. The first illustration shows the connector location and notch orientation. The second illustration shows the silkscreen markings that surround connectors.

The number of standard 10-pin Gadgeteer module connectors varies among mainboard designs.

On mainboards, connectors must be oriented with their keying notch facing outward, toward the nearest outer edge of the board. On modules, connectors may be placed anywhere on the board but, when placed near an outer edge of the PCB, they should be placed with their notch facing inward. This difference between mainboards and modules lets a cable easily go from a mainboard to a module, with a socket near the edge of the mainboard.

Connectors should also comply with the silkscreen guidelines in "Connector Silkscreen" later in this document. Connectors may have tabs as well as the required notch.





Connector Silkscreen

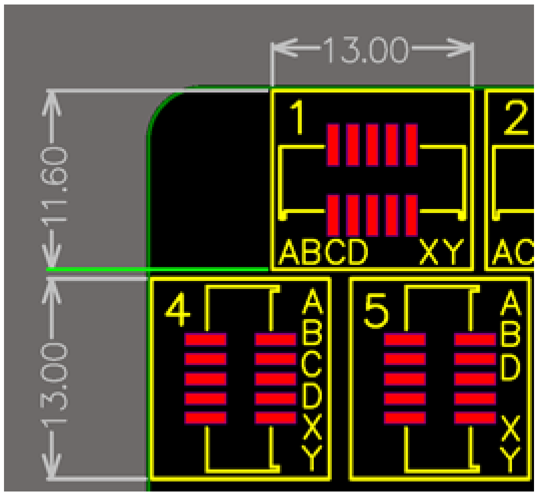
Each of the standard Gadgeteer Module connectors should be surrounded by a silkscreen design that includes the following three elements:

- A bounding box
- The connector's socket number identifier
- A series of letters that indicate the connector's socket types

Bounding Box

Each connector should be surrounded by a 13x11-mm bounding box. The connector should be positioned exactly in the middle of this bounding box. The recommended line thickness for the bounding box is 0.25 mm. The following illustration shows connector positions and bounding boxes.

We recommend a gap (1 mm or greater) between bounding boxes, but to save space, two adjacent bounding boxes may share an edge. However, two bounding boxes should not overlap any further than a shared edge.

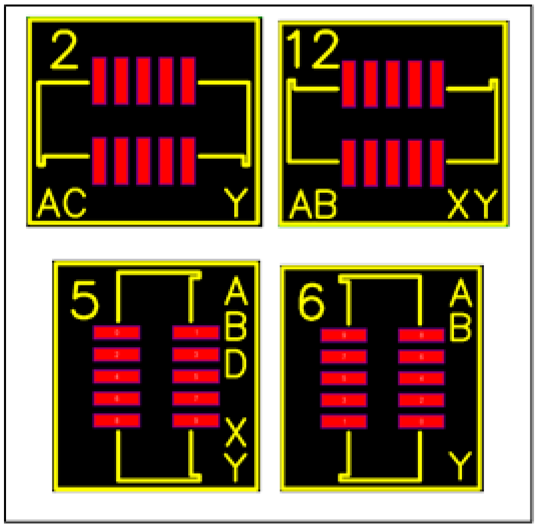


Socket Number Identifier

Connectors should be numbered sequentially with a socket number, with the first number at the top left of the board and the last number at the bottom right. The following illustration shows socket numbering as specified.

The number should appear on the top left of the bounding box, regardless of the orientation of the connector.

We recommend a no serif typeface with a 2-mm height and 0.25-mm stroke width. Be sure that the number does not intersect the bounding box or is not obscured by the connectors, particularly for double digits adjacent to the connector's tabbed side.

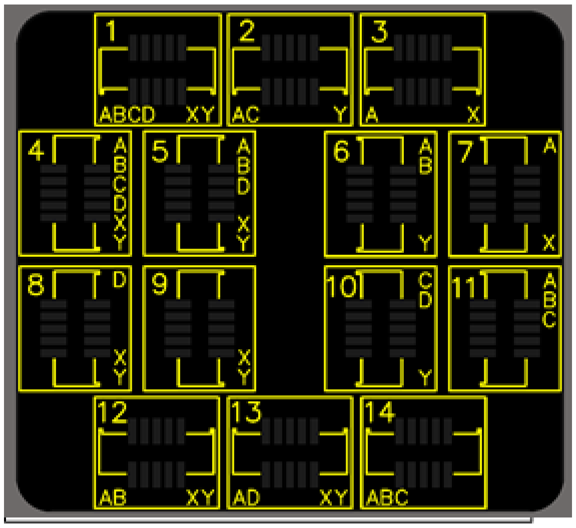


Socket Type Letter Indicators

Each connector should display its associated socket types as a series of letters. The letters should appear on the opposite side from the socket number identifier, regardless of whether the connector is oriented vertically or horizontally. The following illustration shows socket lettering as specified. Socket-compatibility labels are case sensitive.

Letters should be listed alphabetically—left to right for horizontal connectors and top to bottom for vertical connectors.

We recommend a no serif typeface with a 1.5-mm height and 0.2-mm stroke width. This allows up to six letters in either orientation. With the exception of letters X and Y, letters should be left- or top-justified, depending on the connector orientation. If present, letters X and Y should be bottom- or right-justified.

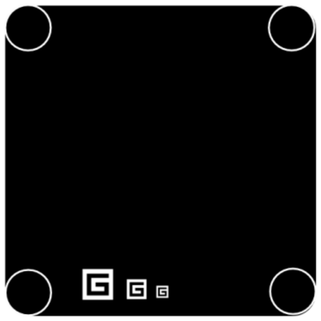


Manufacturer, Mainboard Name, and Version Number Labels

Mainboards should be clearly labeled with the manufacturer's or designer's name, an abbreviation of that name, or a URL clearly containing that name, e.g. www.ManufacturerName.com. They should also have a unique name so that they can be referred to in documentation. This name should be the same as the mainboard class library name, as specified under "Software Provided to Users" earlier in this document. The minimum recommended type is no serif, 1.5-mm high, and with 0.2-mm stroke width. These labels should not be surrounded by a bounding box of any kind.

Gadgeteer Graphic

The following illustration shows the suggested Gadgeteer identifier. We recommend that it be placed in a visible location and further from the socket than any socket type letter so that no confusion exists that the module is for a hypothetical socket G. The identifier is shown in three sizes, but only one should be used on a mainboard or module in production.



- We specify the following guidelines for use of the graphic. You may use the graphic for any purpose that is consistent with its Creative Commons license. We recommend printing it on the PCBs of any .NET Gadgeteer–compatible hardware. To view a copy of the license, visit the Creative Commons website.
- You may use terminology such as "For .NET Gadgeteer" to describe mainboards or modules where appropriate.
- You may not use ".NET" or "Gadgeteer" in your product name.

Additional Guidelines

The following are additional guidelines for mainboards:

- The solder resistance should be black.
- Silkscreens should be white.
- All vias should be tented with solder resistance.
- Mainboards should include a reset button that is tied to the processor's reset pin.
- Mainboards should include one low-current LED—labeled PWR and tied to the power supply—that is lit whenever the mainboard is powered.
- Mainboards should include a second low-current light-emitting diode (LED)—labeled LED—that can be controlled from a pin on the processor.
- All zeros on silkscreens should have diagonal lines through them because "0" is too similar to the letter O and could be misread.
- Mainboards should be small as possible, while they still follow these guidelines.

Shields

These guidelines detail how to create a dedicated Gadgeteer mainboard. Another possibility is to create a Gadgeteer *shield* for existing .NET Micro Framework boards such as a daughter board that interfaces with the existing board on one side and has Gadgeteer sockets and markings on the other side. You can use this method to provide Gadgeteer functionality to an existing installed base of users.

For such shields, we recommend that you follow the guidelines in this document as much as possible. However, you may disregard some of the PCB layout guidelines so that you can make the shield physically similar to the underlying processor PCB. We still recommend following the silkscreen recommendations.

Note that it is not sufficient to simply create the hardware PCB for a shield. You must follow the software guidelines to provide a DLL that implements the Gadgeteer.Mainboard interface. If not, users might be able to plug in Gadgeteer modules and the modules might be electrically compatible, but users would not benefit from the Gadgeteer software core libraries and could not use the libraries that module manufacturers provide along with their modules and high-level, user-friendly APIs for their modules.

Reaching Users

At the .NET Gadgeteer website , Microsoft is hosting a list of compatible mainboards, modules and kits so that end users can easily find them for purchase. This will provide Gadgeteer users with a single point of reference to find Gadgeteer-compatible hardware.

Open Source Community Participation

The .NET Gadgeteer team is looking for ways to establish communication in the broader community of vendors and users that helps ensure that the value proposition of the Gadgeteer ecosystem continues to grow and is not diminished through fragmentation. We plan to set up a "counsel" of vendors and users as a focal point for input on the direction of the technology. We would like your input about how best to engage the community and take advantage of this approach to building devices. If you have further ideas, contact. netgt@microsoft.com.

Resources

Additional information about the .NET Gadgeteer and the .NET Micro Framework can be found at the following sites:

Microsoft

.NET Gadgeteer
http://netmf.com/gadgeteer/

.NET Gadgeteer SDK
http://gadgeteer.codeplex.com/

.NET Micro Framework
http://netmf.com/

.NET Micro Framework SDK
http://netmf.codeplex.com/

Other

Creative Commons
http://creativecommons.org/licenses/by/3.0/us/.