The ADAPTable Sensor System: User Manual

Program:

ADAPTable Sensor System
Defense Advanced Research Projects Agency

Team:

The *Samraksh* Company

Principal Investigator:

Kenneth W. Parker, Ph.D.

Contributors:

Rinka Singh

Nived Sivadas

Nathan Stohs

Christopher Morse

Michael McGrath

Mukundan Sridharan, Ph. D.

Professor Anish Arora

Version 1.0

1/31/2014

Table of Contents

TABLE OF FIGURES

Introduction

A few decades ago long-range sensing or remote sensing was the focus of much of the best research. Although there is still work to be done on remote sensing, some of the research emphasis has shifted to distributed-sensing. One of the motives for this shift is the near universal inability of long-range sensors to "see though" obstructed areas. This problem is illustrated by the almost total lack of point sensors for urban combat. The distributed solution to this problem avoids any attempt to physically see through walls by placing sensors on both sides of all walls and using the power of the network to virtually see through the walls.

Figure 1. A video game showing the Augmented Reality concept of seeing around corners via short range sensors

The Augmented Reality afforded by distributed sensing can require deploying a large number of short range sensors. Deploying a small number of long range sensors is simpler and more economical than deploying a large number of short range sensors. However, long range sensors require long unobstructed fields of view. This severely limits the scope of applicability. When terrain limits the field of view, deploying a long range sensor that can only be used as a short range sensor may be very uneconomical. Recent advances in technology have made it possible to produce much cheaper, smaller, short range sensors requiring far lesser infrastructure and be deployed in larger numbers. This in turn has made many new sensing applications (and domains) tractable and that has driven the interest in short range sensing.

Unfortunately, most of the promise of distributed sensing is yet to be operational zed. We believe this is primarily because the high degree of cross-disciplinary integration required is tough to surmount. The capabilities required to create a state-of-the-art distributed sensing system include:

- Rich set of sensors and radios.
- Rich sensor processing framework with a high degree of in-network processing capability.
- Scalable network co-ordination framework.
- Operating system support for co-existence of multiple application modes, and multiple applications.
- Rich application development environment, so system integrators can program applications with a high degree of flexibility and code reuse, while assuring quality of service (i.e., low false alarm rate) and network lifetime demanded (i.e., energy efficiency).
- Remote manageability by humans.
- Integration with the net-centric enterprise.

Figure 2: Capabilities required to build a better distributed sensing system

The ADAPTable Sensor System remedies this problem for a rich class of distributed sensing

problems. It offers a rich hierarchy of computing, sensing, and radio modes. These include modes that respectively rely on the FPGA, the Micro-controller, the Application Processor (aka PC Mode), and the GPU (aka Supercomputer Mode).

Each computing mode is relevant for different types of sensing and radio/network processing, and requires distinct infrastructure software, i.e., software for the Operating System, Sensing, and Networking services and software for the Development Environment. For instance, the PC Mode is supported by conventional Smartphone software infrastructure: Android provides the Operating System, the development environment developed by SAIC is Android-centric, SAS is added to provide the networking service, and signal processing libraries are added developed by various ADAPT teams (such as the BAE team) provide additional sensing services.

The Micro-controller Mode plays a particularly important role for supporting distributed sensing, which is not substituted by programming in other modes on the ADAPTable Sensor System:

- Building upon the software infrastructure for the PC mode risks a significant loss of energy efficiency compared to the state of the art. For example, Android processing (even when stripped down to exclude redundant components) does not allow apps whose sampling rate is nontrivial (say >10Hz) to sleep between the respective handling of sensor samples; this is due to the latency of its sleep and wakeup operations being no less than several tens of milliseconds. The power consumption of an always-on PC mode reduces the lifetime of the ADAPT node to be of the same order as that of conventional smart phones. As another example, very precise sampling (i.e., with jitter as low as a few microseconds) and ultra-low latency (again within several microseconds) is not afforded by the PC mode due to the relatively heavy processing involved in the Android kernel. This limits the scope of the sensor processing that is well matched to the PC mode. In contrast, the Micro-controller mode delivers greater efficiency than the PC mode, while still supporting flexibility and code reuse.
- Building upon the software infrastructure for the FPGA mode is energy efficient as the power consumption of the FPGA can be low enough that it can be kept almost always on. However, the ADAPT FPGA is inadequate for accommodating more than just the basic sensors due to the severe space constraints of the FPGA.
- Conversely, the Supercomputer Mode consumes the highest power among these processors, and is not a practical option for offering almost always off operation.

This manual focuses on the infrastructure software for the Micro-controller Mode, which we call the **eMote**. eMote is based on the .NET Micro Framework (MF), which is substantially more minimal than Android. It implements a highly energy efficient computing mode, by supporting almost always off computation. This is because it can wake up from and sleep into a very low power state of the ADAPTable System within a few hundred microseconds. It includes soft real-time extensions (the basic MF is not per se a real time executor) due to which it can applications can sample sensors within of tens of microseconds with the jitter being within a few microseconds. And it can operate at a substantially lower speed, to further reduce the power consumption when active.

Equally importantly, eMote offers the benefit of flexibility and code reuse: Application developers program in C# (managed C++ will be supported eventually), while leveraging its sensing and

networking framework services. Since each app runs as a Virtual Machine (VM), multiple apps can be deployed and managed independently. And apps are easily ported across different versions of the ADAPTable System, since the eMote VM offers extreme hardware independence. The tradeoff involved in supporting flexibility and code reuse is a modest amount of performance: Programming the Microcontroller Mode in C# on the .NET Micro Framework is in fact slightly less optimal than the alternative of a single stand-alone C++ app constructed on top of a real-time executor. And the alternative also enjoys a very small footprint and very precise control of power management, but it suffers from limited flexibility and little or no cross app code reuse. The eMote thus offers a resource that system integrators will find valuable uses for, in the form of longer deployments, more complex signal processing, and richer apps.

The Manuals from Samraksh and navigating them

The Documentation provided by Samraksh is as follows:

The ADAPTable Sensor System: User Manual

You are currently reading this manual. The manual introduces the ADAPTable Sensor System, setting up the ADAPT board and getting ready to start writing applications for the board. The manual has the following details:

- An introduction to the ADAPTable system and the eMote platform gives an overview of the system
- An introduction to C# to help programmers who are new to the language. This includes a comparison between Java and C#.
- A brief introduction to the .NET Micro Framework
- A step-by-step guide on setting up your ADAPT board for the first time including downloading and running a pre-compiled application on it.
- Developing your first application
- Links and references are provided to help the developer build deeper knowledge on the ADAPTable system.

Details on programming the ADAPTable Sensor System are provided in the additional guides listed below.

The ADAPTable Sensor System: Programmers Guide

This manual is meant for developers who plan to develop software for the ADAPTable Sensor System. It presumes that you are familiar with the C# language and have setup the ADAPT board.

The manual will detail various C# programs, debug, test and run it on the ADAPTable System. This manual is currently under development.

The ADAPTable Sensor System: the eMote API reference

This document is the companion document of the .NET Micro Framework on the ADAPTable Sensor System: Programmer's Guide. It explains the various APIs provided to developers working on the ADAPTable Sensor system and is meant to be read as you work on developing your applications that will run on the ADAPTable System.

This manual is also under development.

Introduction to the ADAPTable System

The ADAPTable Sensor System is a Machine-to-Machine Hardware Platform that leverages commercial cell phone technology to allow connectivity with general sensor devices for many applications. This manual is intended to provide information to a new user of the ADAPTable System in order that she can understand its functionality, set the board up and serve as an introduction to the companion Manuals on Programming to the ADAPT board.

System Description

The elements of the ADAPT system are divided into three areas: A reusable hardware core, reusable software, and sensor-specific applications. The reusable hardware core aims to leverage low-cost commercial components available at the time of manufacture, enabling it to be refreshed at the rate of commercial technologies. For example, commercial consumer electronic products are typically developed using Original Design Manufacturers (ODM) who promote fabless product development in factories that make a large number of variations on similar products. ADAPT seeks to use ODMs for design and production rather than the common practice of using contract manufacturers.

The ADAPT hardware architecture is based on the MSM8960 mobile processor that realizes various forms of wireless connectivity. This processor is an ARM CPU (it was used in the Galaxy S3 US-Edition) and was chosen for the ADAPT board because it is an established smart phone processor with a support ecosystem already in place and the ability to leverage commercial ODMs and other infrastructure.

The ADAPT board is similar in size to a smart phone motherboard. The RF interfaces are available as coaxial connections using miniature U.FL-type connectors to offer flexibility in antenna selection and application. The device is meant to be incorporated in a system with an enclosure and a power supply.

The ADAPTable System is based on technology from the commercial smart cell phone industry. A typical cell phone incorporates several types of radios for communication and several sensors that provide information about the orientation of the board.

In addition to the standard sensors that would be found on a smart phone, the ADAPTable System also includes an on-board analog-to-digital converter (ADC) and a digital-to-analog (DAC) converter. This gives the ADAPT board the ability to interface with external analog sensors directly. Further, the ADAPT board provides an external interface using a Field Programmable Gate Array (FPGA) that gives the module flexibility when interfacing to application specific digital sensors that are not located on the module. As a result, the ADAPT board provides a flexible solution for gathering and processing data from on-board standard sensors and from application unique sensors that are located off-board.

A USB OTG 2.0 port is provided for debug purposes during software development. The ADAPT board may be powered by the USB port up to the 500mA current limit imposed by the USB standard.

You would find more details about the ADAPT Board in the next section.

Figure 3: The ADAPT debug Board

While the ADAPT system originally supported Android, Samraksh provides a modified little kernel that allows programmers to develop their applications to use Windows Visual Express and C#.

ADAPTable Sensor System Configuration and Limits

The following is the system level configuration information. The hardware of the ADAPTable Sensor System is based on the Qualcomm MSM8960 and consists of an application processor, cellular modem, wireless connectivity (WLAN/BT), GPS receiver, and a configurable digital interface

- CPU: Qualcomm Snapdragon S4 MSM8960 mobile processor.
- RAM Memory: 1 GByte LPDDR2
- Non-volatile Memory: 8 GByte e-MMC NAND Flash

- SD Card Interface
- The other interfaces supported by the ADAPT board are the Keypad, LCD and Touchpad
- The Interface devices supported by the ADAPT board are an external Sensor Interface connector and a debug connector. In both cases, the ADAPT connector is the Panasonic AXK5SA2277YG-A and the mating connector is the Panasonic AXK6SA2477YG
- USB Connector: The ADAPT board provides a micro-B USB receptacle for diagnostic communication. The ADAPT Connector is HRS ZX62D-B-5PA8 and the mating Connector is a Standard USB micro-B Plug
- The radios supported include a cell modem, Wi-Fi, Bluetooth, GPS and FM. The cell modem has a WTR1605L transceiver that can be used in multimode, multiband applications including Rx diversity
- Wi-Fi and Bluetooth: The Broadcom BCM4330 is used to implement WLAN. It includes an integrated dual-band direct conversion radio that supports either the 2.4GHz or the 5 GHz band. And BCM4330 supports 802.11a/b/g/n.
- There are four antenna connectors on the ADAPT board. All of the antenna connections on the board use a u.fl style connector.
- The ADAPT board has the following sensors:
    - Accelerometer.
    - Pressure / Temperature sensor.
    - Gyroscope sensor.
    - E-compass sensor.
    - Ambient light and proximity sensor.
- GPS: the WTR1605L IC integrates advanced GNSS receiver functions. It supports simultaneous GNSS – allows GNSS processing at the same time as CDMA, WCDMA, or LTE processing of the primary and diversity paths (MRD).
- D/A Converter: The DAC8562 is the Digital to Analog converter. It is a 2-channel 16-bit DAC. The two channels are single-ended output. It is a precision DAC with dual 16-bit, low power offered.
- A/D Converter: The AD7689 is the analog to digital converter. It is an 8-channel, 16-bit, charge redistribution successive approximation register (SAR) analog-to-digital converter (ADC).
- Camera Interfaces: The ADAPT board has two MIPI camera interfaces. These are based on Micron's MT9P111.
- FPGA: The Microsemi AGL400is known for its low power consumption and has 400k logic gates. The FPGA also has a 32kByte serial SRAM and a real-time clock.
- Battery Requirements: The ADAPT board supports both the Lithium-ion and the Lithium-ion polymer batteries
    - The ADAPT board supports battery voltages in the range of 2.6 volts to 4.2 volts. If the battery voltage is in the range of 2.6V-3.4V the board will trickle charge the battery. If the battery voltage is 3.4V-4.2V, the board will charge the battery normally (non-trickle charge mode).
- Power Consumption: The ADAPT board's consumption varies depending upon its mode of operation. The following describes the typical current consumption. A full characterization has not yet been performed to detail the current consumption as a function of the various operating modes.
    - During Boot-up, the board's power requirements is 660 mA

  - During regular operation, the board's power requirement is about 115 mA
- The ADAPT board supports three primary power methods: USB, battery or an external power supply. The details of these are as follows:
  - External Power Method: The USB Voltage is +5.00 +/-0.25 Volts
  - Battery is +2.6 volts to +4.2 volts
  - External Power Supply should provide voltage between +4.5 volts and +6.5 volts

  The ADAPT board requires approximately 660mA of current during the boot process. A PC USB port can only supply up to 500mA per the USB specification. Therefore, the ADAPT board will not be able to boot-up when powered solely from a PC USB port. A battery device or other external power source must be used to supply the needed current. A USB charger capable of 1Amp is capable of powering the device.
- Charge LED: The ADAPT board has a red LED that is used to indicate that battery connected to the board is charging. The LED will slowly flash ON and OFF when the battery is connected to a USB port on a PC or a USB charger or external power supply. The LED will be OFF if the board is running solely on battery power or if the battery is fully charged.
- Battery Connector: The ADAPT board has a three-pin connector for battery connections. The ADAPT Connector is ATOM010070040067 while the mating Connector is a Molex 51021-0300
- Code, binaries, and documentation have been verified with ADAPT hardware core / debug revision EVT2 and SIMcom software release V05.

The eMote Platform

The eMote platform is an extension of Microsoft's .NET Micro Framework meant for the ADAPTable sensor system. "eMote" refers to the pre-installed software that lets you write programs in C#.

The eMote platform delivers wireless networking out of the box and also offers real time extensions to the .NET Micro Framework's CLR. These are described in the sections below.
Microsoft's .NET MF scheduler is a weighted priority non pre-emptible scheduler. It supports multi-threaded applications similar to the other .NET versions. Each thread is assigned a priority. Interrupt and Timer events run inside their own threads and have the highest priority. Once a thread starts running it cannot be pre-empted by other application threads and they usually run till the current activity is complete or their allocated *time quantum (20ms* by default*)* is over. This non pre-emptible nature of the scheduler poses timing issues for asynchronous interrupt driven applications with strict timing requirements. Samraksh's Real Time extensions addresses these issues as described below.

The Samraksh Real Time Extensions

Samraksh has modified the CLR and Scheduler to make it a pre-emptible scheduler. This is needed in developing the real-time capability.

For the C# user, the real-time extension provides a timer interface called RealTimeTimer; it is similar to the standard C# timer. But the execution of this timer handler will happen in interrupt mode; that is, the managed handler of the RealTimeTimer can interrupt all other threads includ-

ing the regular Interrupt and Timer threads. The result is a timer with very accurate timing properties. The other difference between the RealTimeTimer and the standard timer is that the minimum timer period supported by the standard timer is 1ms, while we support a timer period of a few 100 micro-seconds.

The Samraksh Real Time Timer and Garbage Collection

In the standard MF, if the Garbage Collector (GC) starts running, all threads belonging to the managed application are suspended until the GC finishes cleaning the heap and also possibly compacting the heap (which can be a time consuming operation). The GC can interrupt the application threads, including timer and interrupt threads.

The RealTimeTimer provides a mechanism to avoid this problem, provided each and every single variable and object used in the RealTimeTimer handler is allocated either statically or globally. If this is done, then the code in the RealTimeTimer handler will continue to execute even when Garbage Collector is active. This feature makes the RealTimeTimer very accurate, in most cases with an average jitter of a few microseconds.

Measurements have shown that jitter performance can be less than 3 micro-seconds for timer values above 500 ms. You will find details on the various Samraksh APIs in the document: eMote .NET Micro Framework on the ADAPT platform: API reference

Introduction to C# and comparing it to Java

Designing and developing software for distributed sensing is complex. This is one of the main reasons why a higher level language like C# was chosen. Some of the advantages of using C# are:

- A high level language makes it easy to develop with and a robust IDE is available
- The ability to use OOPS for design and development simplifies and speeds up software development.

The ADAPTable system supports C# currently and will support C++ in future. The following is a brief recap of C#. For more details, you will find the definitive reference at Microsoft's pages on C#.

C# (pronounced "C sharp") is an object-oriented, type-safe programming language. Created by Anders Hejlsberg, the father of Delphi, C# was designed to be an optimal Windows development language and is based on C++, Delphi and Java.

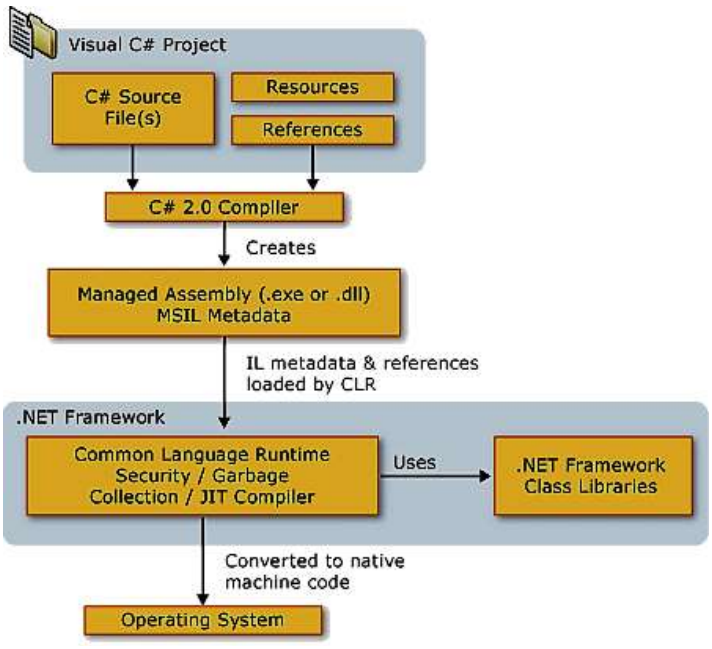


Figure 4: The Visual C# Project

The integrated development tool from Microsoft is the Microsoft Visual Studio©. The tool comes in two flavors – a full-featured priced version Visual Studio and the free version Visual Studio Express©. The Microsoft Visual Studio© is a full development system and supports various optimization capabilities, data access, multi-language capabilities and various other features. The Microsoft Visual Studio Express© is a free download. However, you do need to register your copy with Microsoft. The Visual Studio Express is a full-fledged IDE that allows the developer to develop and debug C# applications.

C#, like Java, is executed indirectly through an abstract computer architecture called the Common Language Runtime (CLR). C# programs are compiled to an intermediate language called MSIL or Common Intermediate Language (CIL).

The CLR transforms the CIL to assembly instructions for a particular hardware architecture. This is termed as jit'ing or Just-in-time compiling. While there is an initial performance cost, the jitted executable is cached for further execution and subsequent execution has no performance penalty. Some performance gains are also possible since the CLR can target the specific architecture that the code is compiled for.CLR is actually an implementation by Microsoft of the Common Language Infrastructure (CLI). CLI is an open specification while CLR is really a platform specific implementation.

Common Language Infrastructure (CLI)

CLI allows for cross-language development and has four components:

- Common Type System (CTS)
- Assemblies and Meta-data.
- Common Language Specification – behaviors that all languages need to follow.
- A Virtual Execution System (VES).

The Common Type System (CTS) is a specification for how types are defined and how they behave. Types have no syntax specified and can contain zero or some of the following members:

- Field
- Method
- Property
- Event

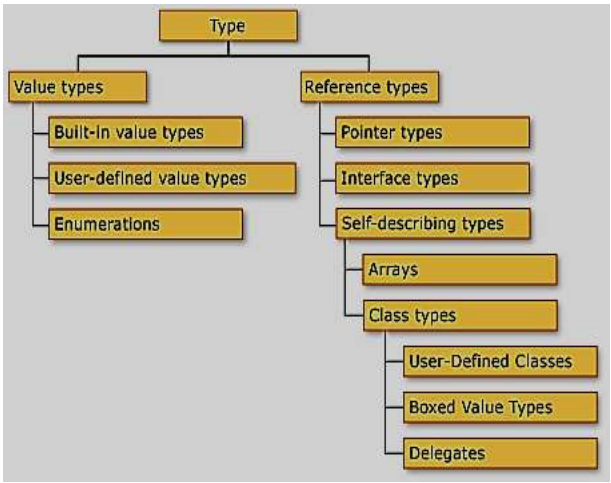


Figure 5: Various Types supported by C#

CTS also specifies the rules for visibility and access to members. This is done by declaring the member one of a type:

- Private
- Family
- Family and Assembly
- Assembly
- Family or Assembly
- Public

There are three types of applications supported by C#. The first is the Console application. This usually runs in the console and communicates with the user via the standard streams (out, in, err). The GUI for such an application can be added manually. The second type is the Windows application: This is GUI based and does not support the standard streams (out, in, err). The main thread is shared by the GUI message pump and your code. And finally, the third type is the Service application. This doesn't support standard streams (out, in, err) or GUI. Its main thread is commandeered by the Service Control Manager (SCM).

Assemblies are specific to the .NET framework. Both .EXE and .DLL assemblies can be referenced or loaded by your .NET application. An assembly can be a single a .DLL or .EXE file, but not all .DLLs and .EXEs are assemblies. The assembly manifest is just one part of the assembly that contains metadata describing the assembly.

The assembly manifest:

- Enumerates the files that make up the assembly.
- Governs how references to the assembly's types and resources map to the files that contain their declarations and implementations.
- Enumerates other assemblies on which the assembly depends.
- Provides a level of indirection between consumers of the assembly and the assembly's implementation details.
- Renders the assembly self-describing.

You can add and change some aspects of the assembly manifest from your code if you need to change the metadata associated with that specific assembly. The assembly manifest may be stored as its own physical file, but it is still considered part of the assembly.

The assembly is composed of CIL code, metadata that describes the types defined by the CIL code, the assembly manifest which is metadata that describes the assembly and other resources such as static images needed by the assembly. Therefore, the assembly manifest is just one part of an assembly as is the metadata. The assembly can be compiled into either an .EXE or .DLL file, but can also be comprised of multiple files.

Metadata is binary information describing your program that is stored either in a common language runtime portable executable (PE) file or in memory. When you compile your code into a PE file, metadata is inserted into one portion of the file, and your code is converted to Microsoft intermediate language (MSIL) and inserted into another portion of the file. Every type and member that is defined and referenced in a module or assembly is described within metadata. When code is executed, the runtime loads metadata into memory and references it to discover information about your code's classes, members, inheritance, and so on.

Metadata describes every type and member defined in your code in a language-neutral manner. It stores the following information:

- Description of the assembly.
- Identity (name, version, culture, public key).
- The types that are exported.

- Other assemblies that this assembly depends on.
- Security permissions needed to run.
- Description of types.
- Name, visibility, base class, and interfaces implemented.
- Members (methods, fields, properties, events, nested types).
- Attributes.
- Additional descriptive elements that modify types and members.

The Common Language Specification (CLS) is a fundamental set of language features supported by the Common Language Runtime (CLR) of the .NET Framework. CLS is a part of the specifications of the .NET Framework. CLS was designed to support language constructs commonly used by developers and to produce verifiable code, which allows all CLS-compliant languages to ensure the type safety of code. CLS includes features common to many object-oriented programming languages. It forms a subset of the functionality of common type system (CTS) and has more rules than defined in CTS.

CLS defines the base rules necessary for any language targeting common language infrastructure to interoperate with other CLS-compliant languages. For example, a method with parameter of "unsigned int" type in an object written in C# is not CLS-compliant, just as some languages, like VB.NET, do not support that type.

The Virtual Execution System (VES) provides an environment for executing managed code. It provides direct support for a set of built-in data types, defines a hypothetical machine with an associated machine model and state, a set of control flow constructs, and an exception handling model. To a large extent, the purpose of the VES is to provide the support required to execute the Common Intermediate Language instruction set.

Here's a sample program to give an idea of what C# programs look like:

```
using System;
namespace HelloWorldApplication
{
   class HelloWorld
   {
      static void Main(string[] args)
      {
         /* my first program in C# */
         Console.WriteLine("Hello World");
         Console.ReadKey();
      }
   }
}
```

Let us examine various parts of the above program:

- `using System;` : The using keyword is used to include the System namespace in the program. A program generally has multiple using statements. Note the ";" that ends the line. The ";" is used similar to Java or C++ and is used to denote the end of a command.
- `namespace HelloWorldApplication` : A namespace is a collection of classes. The

HelloWorldApplication namespace contains the class HelloWorld.

- `class HelloWorld`: This is the class declaration for the class HelloWorld. This class contains the data and method definitions that the program uses. Classes generally contain more than one method. Methods define the behavior of the class. However, the HelloWorld class has only one method Main.
- `static void Main(string[] args):` The method Main is the entry point for all C# programs. The Main method states what the class will do when executed. You can pass various parameters in with the string [] declaration and the number of parameters with the variable arguments.
- `/* my first program in C# */` is a comment and will be ignored by the compiler. The syntax follows the C programming syntax for comments. An alternative mechanism of adding comments is using "//" as used in Java and C++.
- `Console.WriteLine("Hello World");:` The Main method specifies its behavior with this statement. WriteLine is a method of the Console class defined in the System namespace. This statement causes the message "Hello, World!" to be displayed on the screen.
- `Console.ReadKey();:` The last line Console.ReadKey(); is for the VS.NET Users. This makes the program wait for a key press and it prevents the screen from running and closing quickly when the program is launched from Visual Studio .NET.

It is worth noting the following points:

- C# is case sensitive.
- All statements and expression end with a semicolon ";".
- The program execution starts at the Main method.
- Unlike Java, file name can be different from the class name.

For more on C# programming and using the Visual Studio IDE, please refer to Microsoft's page on C# programming.

Comparing C# with Java

As described above, C# has been and continues to be influenced by Java. Both languages are updated to keep up with each other. Given the similarities, Java developers will find it quite easy to migrate to C#. This section is meant for Java developers who would like to understand C# better. Microsoft's page has the most definitive comparison. Wikipedia also has a list. Here's a brief list of the differences between the two languages:

Java is relatively more "platform independent" as compared to C#. The best way to explain this is when a new release of Java is done it is simultaneously available on all platforms it supports as compared to the features which will be missing in the Mono (An open source, cross-platform, implementation of C# and CLR) implementation. On the other hand C#'s JIT Compiler generates machine code tuned for the specific platform it's running on, this may allow for faster execution as compared to Java which generates class files that run on JVMs.

Some of the differences between C# and Java are:

- Generics are completely different between the two; Java generics are compile-time while C# and .NET generics are maintained at execution time too, and work for value types as well as reference types, keeping the appropriate efficiency.
- Java doesn't have any of the following capabilities:
    - operator and conversion overloading
    - iterator blocks for simple implementation of iterators
    - any equivalent of LINQ
    - expression trees
    - C# 3.0 anonymous types
    - implicitly typed local variables
    - extension methods
    - object and collection initializer expressions
    - Pre-processor directives (#define, #if etc. in C#).
    - C#'s ref and out for passing parameters by reference
    - partial types, nullable or unsigned integer types
    - Language support for a decimal type. (**java.math.BigDecimal** provides something like **System.Decimal** with differences - but there's no language support)
    - "unsafe" code
    - creation of user-defined value types
- The access modifiers are somewhat different - in Java there's (currently) no direct equivalent of an assembly, so no idea of "internal" visibility; in C# there's no equivalent to the "default" visibility in Java which takes account of namespace (and inheritance)
- The order of initialization in Java and C# is subtly different (C# executes variable initializers before the chained call to the base type's constructor)
- Interoperability is easier in C# (and .NET in general) as compared to Java's JNI
- Java treats enums differently, they are much more object-oriented.
- And some of the features that C# doesn't have:
    - C# interfaces cannot declare fields
    - C# doesn't have checked exceptions
    - C# doesn't have anonymous inner classes
    - C# doesn't have Java's inner classes at all, in fact - all nested classes in C# are like Java's static nested classes

.NET Micro Framework

The .NET Micro Framework (MF) is an open source .NET platform for rapidly creating embedded device firmware with Microsoft Visual Studio. It provides the means to develop firmware for devices that are too constrained to run Microsoft Windows CE and the .NET Compact Framework. With the MF, you can use Visual Studio to write, test, and debug software on your hardware just as you would when developing software for the .NET Framework or the .NET Compact Framework.

It includes a small version of the .NET CLR and supports development in C#, Visual Basic .NET, and debugging (in an emulator or on hardware) using Microsoft Visual Studio. The MF features a subset of the .NET base class libraries (about 70 classes with about 420 methods), an

implementation of Windows Communication Foundation, a GUI framework loosely based on Windows Presentation Foundation, and a Web Services stack based on SOAP and WSDL.

MF was created from the ground up with very small devices in mind. The range of functionality in these devices are typically constrained by some combination of cost, memory, processing capability, and power. Devices with the MF running on them typically need at least 256 KBytes of flash and 64 KBytes of RAM. The framework has been used in some of the smallest devices, including watches, GPS navigation devices, and Microsoft Windows Vista SideShow displays. Being targeted at the new, less expensive, and more power-efficient 32-bit processors that are readily available makes it possible for the MF platform to run on significantly reduced resources. Additionally, it offers power management interfaces to your code, allowing you to maximize the battery life of your devices.

The architecture of the MF provides you with the tools necessary to develop your firmware as managed C# applications. It does this by making use of specialized versions of the .NET common language runtime (CLR) and the .NET Framework class library. You will find the link to the MF SDK in the section Documentation.

The .NET Micro Framework architecture makes it possible to use Microsoft Visual Studio and C# to program an embedded system. The devices covered are:

- Sensor networks
- Robotics
- GPS navigation
- Wearable devices
- Medical instrumentation
- Industrial automation devices
- Other small devices that require an efficient, low-resource-consuming Microsoft .NET client

The following diagram shows the .NET Micro Framework hardware and software architecture.

Hardware Layer

The hardware layer contains the microprocessor and other circuitry. The .NET Micro Framework runs on a growing number of hardware platforms. It can also run on top of operating systems such as Microsoft Windows® and other operating systems. The Micro Framework SDK includes a device emulator that runs on Windows.

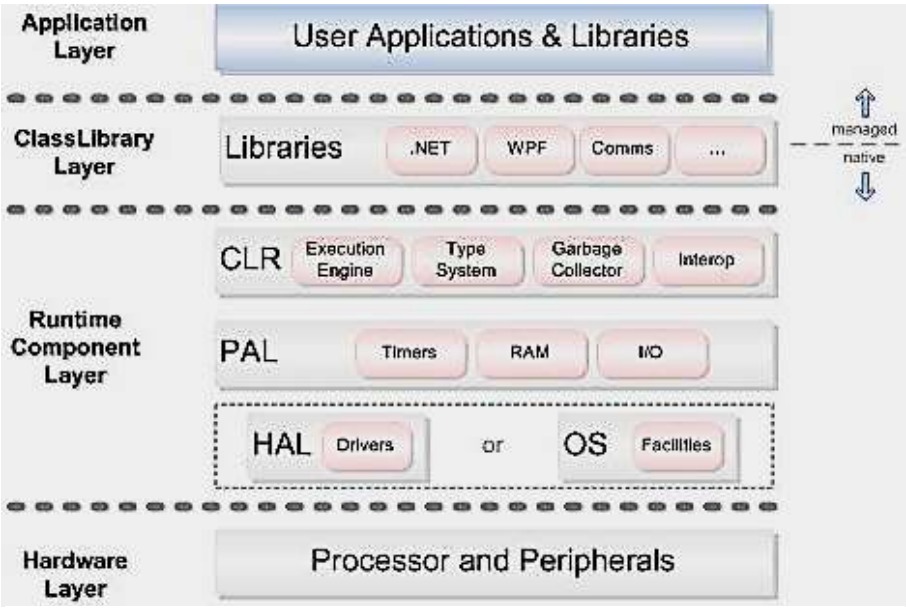


Figure 6: The layers of the .NET Micro Framework Architecture

Runtime Component Layer

The runtime component layer (RCL) includes three components: the .NET Micro Framework common language runtime (CLR), a hardware abstraction layer (HAL), and a platform abstraction layer (PAL).

Common Language Runtime (CLR)

The .NET Micro Framework CLR is a subset of the .NET Framework CLR, the run-time environment provided by the .NET Framework. CLR provides application support, manages memory, thread execution, code execution, and other system services.

The CLR's memory footprint is about 390 KB of memory when all of the provided functionality is being used. The CLR performs at about 15,000 managed method calls per second at 27.6 MHz. (These results are based on the average method-call complexity developed for the SPOT watch application.)

The CLR can be programmed and debugged using the Visual Studio .NET and supports only C# as its programming language.

Features of the .NET Micro Framework CLR include:

- Numeric types, class types, value types, arrays (one-dimensional only), delegates, events, references, and weak references
- Synchronization, threads, and timers
- Reflection
- Serialization
- Garbage collection
- Exception handling
- Time classes, including **DateTime** and **TimeSpan** (using INT64 arithmetic)
- Time-sliced thread management

Exceptions to and extensions of the CLR include:

- Execution constraints that limit call durations and prevent failures.
- Strings represented internally as UTF-8 and exposed as Unicode.
- No support for multidimensional (sparse or jagged) arrays.
- A global, shared string table for well-known values (types, methods, and fields) that reduces RAM and ROM and reduces the number of cross-references among assemblies.

- No virtual tables for method resolution, which saves RAM.
- A **WeakDelegate** class to handle dangling references to delegates.
- Support for extending the hardware chipset by programming its drivers directly in C#, using the Managed Drivers Framework. Hardware devices compatible with industry communication standards (e.g. GPIO, serial, SPI, or I2C) can be added to the chipset and used by a managed C# application.

Hardware Abstraction Layer (HAL), and the Platform Abstraction Layer (PAL)

The HAL and the PAL control the underlying system hardware. Both the HAL and the PAL are groups of C++ functions called by the CLR. The PAL functions are independent of the hardware and should not need to be ported. However, the HAL is specific for the hardware platform.

The bootstrap code is an additional piece of code that is associated with the HAL. The bootstrap code initializes the low-level hardware when the device is turned on. It then starts the CLR, which performs the higher-level initializations. The bootstrap code performs its tasks through calls to the HAL and assembly-language routines. Other than starting the CLR, it has no interaction with the code preceding it in the software architecture.

Class Library Layer

The class library included with the .NET Micro Framework is an object-oriented collection of reusable types that can be used to develop embedded applications. It includes C# libraries that provide support for the following:

- Encryption
- Debug, graphics, and shell DLLs
- The CLR API class library and the CLR core lib
- Access for managed C# applications to extended chipsets that support specific communications standards, such as GPIO, serial, SPI, or I2C

Application Layer

The top layer of the .NET Micro Framework contains managed applications that are created by the programmer for the ADAPTable Sensor System.

Setting up the ADAPT board

This section describes the steps you need to take to bring up the ADAPT board for the first time. The overall sequence is as follows:

- Get the material needed together
- Install the Visual Studio IDE
- Install the Micro Framework
- Download and install the Android SDK since you will use components of the SDK to setup your board and download your apps.
- Prepare the board, connect it to the host and load your first application.

These steps are detailed in the sub-sections below.

Step 1: Getting the material together

Ensure you have the following things before you start:

- The ADAPT board and its various peripherals. You would get this from Samraksh or one of its affiliates
- The Host is a Microsoft Windows 7 64bit workstation. Please note: You need a 64 bit workstation. You will have issues with getting the whole system up and running on a 32 bit workstation.
- Ensure you have downloaded all the various documents, drivers and binaries from various sites. Details on each of these are covered in the subsequent sections:
    - Visual Studio and Micro Framework from Microsoft.
    - The Android SDK from the Android website
    - The SI Labs drivers from their website
    - The various documentation, MF and drivers from Samraksh's SmartFile site. You should download the following components:
        - android-v5-boot.img
        - LK-v5.mbn
        - myapp.dat
        - Samraksh-LK.mbn
        - tinyclr.bin
        - usb_driver.zip (explode this file – it contains the drivers you need)
- A USB cable with a micro-B USB interface. Ideally, you should get two cables, one to connect the host to the board for downloading your images and the second to connect the board's UART to a console for watching the console messages from the board

Figure 7: The Battery Port

- Battery – make sure it is charged. You can charge it by connecting the board to the USB port of the host, connecting the battery to the battery port and leaving it to charge. Charging time is usually 3 hours. Please note:
    - You should switch the board over to the Android kernel and charge the battery at this point. The eMote kernel does not support battery charging at this point. We are working on it and will release it in a future version.
    - You do not need the battery if you have the USB port connected to the workstation as the ADAPT board can run off USB power. It is only needed if you have just one USB cable and are using it to connect your workstation to the UART port.
- Download the Serial USB Driver from SI Labs. Download and install it by running the executable contained there.
- Download and install PuTTY from http://www.putty.org/. Alternatively you can use Hyperterminal from Microsoft. This manual uses PuTTY as the tty tool.

Step 2: Install the Visual Studio IDE

Step 2a: Visual Studio 2012 Windows Desktop is used as a C# development environment for eMote. If you do not have it installed, then you can download and install the free Express version from http://www.microsoft.com/visualstudio/eng/downloads

NOTE: Visual Studio 2013 does not support the .NET Micro Framework SDK version 4.3 currently. We are working with Microsoft to have this resolved. Please use Visual Studio 2012 till then.

Step 2b: Once installed, you need to register your software with Microsoft. The procedure to register and unlock the Express Edition is as follows:

- On the **Help** menu, click **Register Product**.
- Click the **Register now** link to get your free key to unlock your Express Edition.
- Sign in to **Passport** and then fill out the **online registration form**. The key is mailed to your mail account.
- Paste the key in the **Registration Key text box** and then click **Complete Registration**.

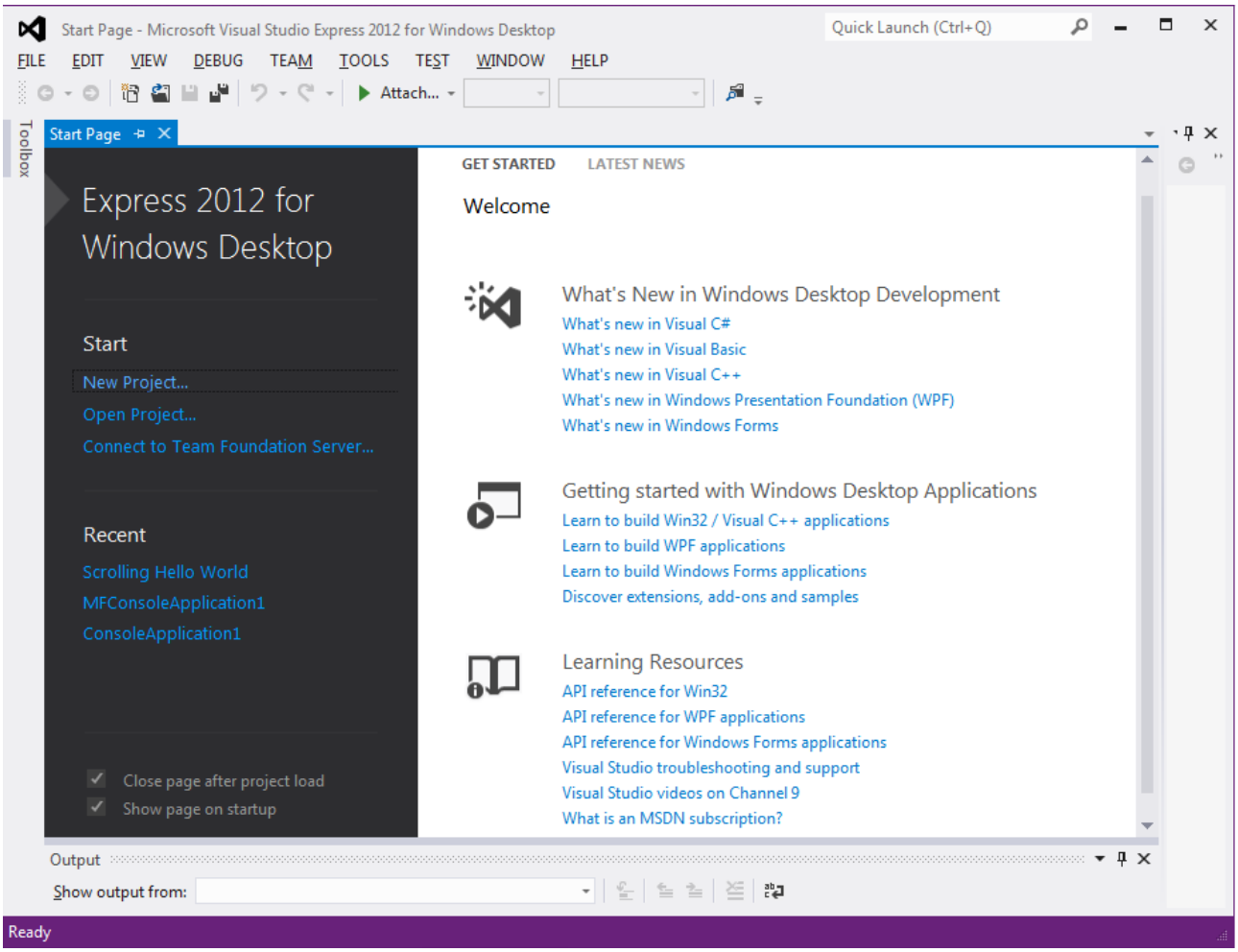


Figure 8: Screenshot of Visual Studio Express 2012 for Windows Desktop when first opened

You are now ready to write C# applications for the .NOW platform using the Visual Studio 2012 Windows Desktop. For the purposes of this manual we assume the Express version, but procedures are similar for the paid versions.

Step 3: Install the Micro Framework

Install the Micro Framework SDK 4.3. In order to do so, you need to first download the SDK. This is a Microsoft install file .msi. Click on the installer to start the install process after downloading.

Step 4: Install Android SDK

The Micro Framework can be deployed via either MFDeploy and/or Visual Studio. Currently deployment of a MF application requires the use of the Android SDK tool 'fastboot'. Native deployment through Visual Studio is not yet supported. fastboot is used to write a modified bootloader, MF binary, and MF app to the eMMC flash of the ADAPT board.

Step 4a: Download the Android SDK toolset (in particular 'fastboot'). You should download the 64 bit version (since you are using a 64 bit machine). Once downloaded, unzip into an appropriate directory. You will find more on setting up the SDK toolset at the link.

Step 4b: To ensure you have unzipped and set up the SDK toolkit properly, on the command prompt do the following:

- Change directory to SDK\Platform-Tools
- Run the executable "fastboot"
- If you get errors, that is because you usually haven't installed a Java Runtime (JRE) on your machine. Both, the SDK manager and fastboot will need a JRE to function. Please see the section Frequently Asked Questions for details.

Step 4c: The SDK is really needed only for downloading the Android device drivers and the command fastboot. You should first update the SDK and then install the Google USB driver. This is done as follows:

- Run the SDK Manager – this executable is found in the top directory containing the toolkit
- You should install the following packages:
  - Check Android SDK tools and Android SDK Platform tools in the section tools
  - You don't need to update any of the Android API. They are not needed.
  - Check the Google USB driver in the section Extras to install them
- Once done, click on the "Install n Packages" button to install these where "n" is the number of packages that you will install. "n" is usually between 3-4 packages depending on the version of the SDK available at that point in time.

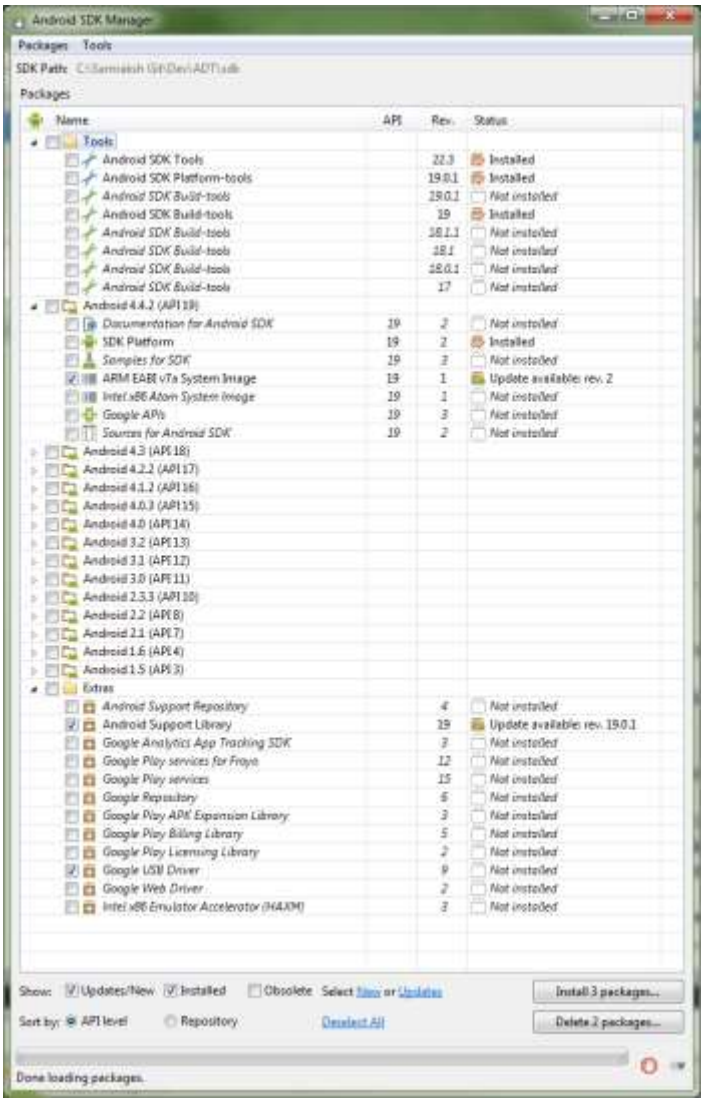


Figure 9: SDK Manager

Step 4d: Run the SDK manager and ensure the Android library and USB drivers are updated.

Step 4e: Once installed, you should append the new "platform-tools" path to your $PATH to use fastboot from the command line. You do this as follows on Windows 7.0:

- Right-click on **My Computer**, click on **Properties**
- In the System Properties window, click on the **Advanced** tab and then the button **Environment Variables**.
- Highlight the system variable "**Path**" and click edit.
- Scroll to the end of the path and append the path to "platform-tools" there. Please ensure you put a ";" before the platform-tools path.

Step 4f: You will not be able to connect to your board without the Android USB drivers. To install the Android USB drivers, you need to do the following:

- Power-off the board by removing the USB cable and the battery (if you've plugged it in).
- Start the Windows Device Manager. You do this by clicking on the **Start button**, **Control Panel**, **Hardware and Sound** and finally the **Device Manager link** under the **Device and Printers** section.
- Hold the VOL+ button down on the board and insert the USB connector into the USB port (located on the upper left of the ADAPT board). This powers the board up.
- Continue holding the VOL+ button down for 10 sec and then release it.
- Your board will boot into Android since that is the current bootloader running on the device. You will see an uninstalled "Android device" with an "Android bootloader interface" under "Other Devices" in the Device manager.
- Right-click on the "Android bootloader interface" and install the android device driver from the Android SDK.

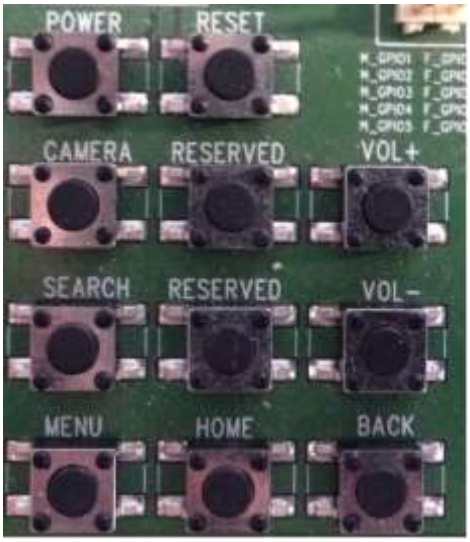


Figure 10: Keyboard labels on the ADAPT board

Step 4g: Once installed, test the fact that you are connected to the board by typing "fastboot devices" at the command prompt of the host. You should see a device listed. Please note: The Android device drivers do not install on a 32 bit machine.

Recent versions of the Android SDK should be compatible with or include drivers for the ADAPT board. For Windows, the latest drivers are included in this deliverable as "usb_driver.zip" or can be downloaded from Samraksh's SmartFTP site.

Step 5: Install the Serial over USB Driver

Step 5a: Install the Serial over USB driver. Connect a standard Micro-B USB cable to the connector to your PC. See the picture below. Typically, the operating system should be able to download the necessary driver and install it. If there are problems with the driver installation, you can get it from the SI Labs site.

Step 5b: Download the zipfile and unzip it. You should run the application included to install the drivers.

Step 5c: Once installed, connect the second USB Cable to the workstation and to the USB port marked USB for UART. The driver should now appear under Ports in Window's Device Manag-

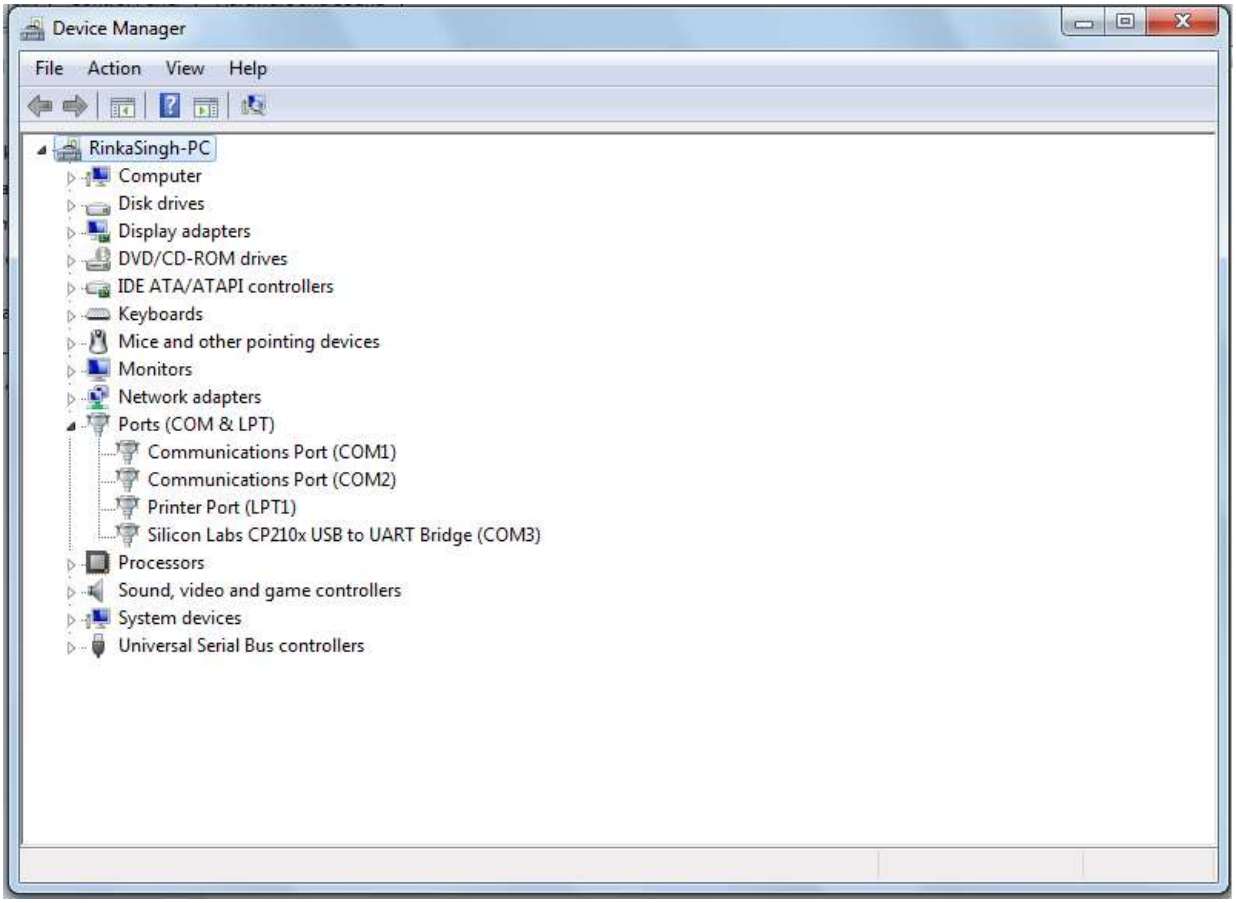


Figure 11: List of ports

er.

Step 5d: Run "**fastboot devices**" at the command prompt to ensure you can see the various devices. Please see Figure 12 for more.

Step 5e: Get the USB port number from the Device Manager as shown in the figure above. Start PuTTY and set the configuration to the port number above and the baudrate to 115200. Restart the board while holding down VOL- and you should start seeing the Android init messages scrolling on the PuTTY console.

Step 6: Install Modified 'Little-Kernel'

Samraksh uses the Android 'boot' partition to store the MF and application binaries. Normally this partition holds the Android Linux Kernel image. The Little-Kernel (LK) is the last bootloader in the normal Android loading sequence, it normally launches the Linux Kernel. Samraksh has modified the LK to launch the MF instead of Android. The pre-compiled LK from Samraksh is

"samraksh-LK.mbn". This LK also tweaks the boot sequence depending on the keys pressed at startup on the debug board. The changes are as follows:

- By default the board will boot to fastboot mode.
- If the VOL+ key is pressed it will launch MF.
- If BACK is pressed it will launch the SIMcom loader for use with SIMcom tools.

Step 6a: Write the modified LK to the ADAPT board as follows:

- Power down the board and remove the USB connector, hold down the VOL- key on the debug board and plug in the USB cable. Keep holding the VOL- key for 10 sec and then release. This will boot the ADAPT core to fastboot mode.
- Plug the second USB cable into the USB port for UART on the ADAPT board and ensure that the Device Manager page shows the Android device.
- Enter '**fastboot devices**' at the command line. You should list a device as present.
- Enter '**fastboot flash aboot samraksh-LK.mbn**' at the command line. This will write the modified LK. Example output is Figure 12.

```
E:\vmshare>fastboot devices
0123456789      fastboot

E:\vmshare>fastboot flash aboot samraksh-LK.mbn
sending 'aboot' (231 KB)...
OKAY [  0.021s]
writing 'aboot'...
OKAY [  0.058s]
finished. total time: 0.079s
```

Figure 12: Writing the modified LK to the ADAPT board

- Restart the board as follows: Unplug all the power sources, hold down the VOL- key and plug in the USB. Continue holding down the VOL- key for 15 sec and then release. You should see the Samraksh logo on the screen

Step 6b: To validate that you have installed the Samraksh LK properly, follow the following steps:

- Power down the board and connect the serial cable to the UART port.
- Start PuTTY on the host and set the following parameters for the PuTTY console:
  - Set the port to the USB port that you see in the Device Manager screen
  - Set the baud-rate to 57600 bauds
- Hold down the VOL- key, connect the USB cable and release the VOL- key after 10 seconds.
- You should see the init messages scrolling on the console.
- If you see gibberish on the console, you probably haven't installed the Samraksh LK in the boot partition. Here's how you check if the Android LK is still there or has been over written.
- Power down the board
- Start PuTTY on the host and set the following parameters for the PuTTY console:
  - Set the port to the USB port that you see in the Device Manager screen
  - Set the baud-rate to 115200 bauds

  And then start the PuTTY console.

Step 6c: To verify that you have flashed the LK properly, do the following:

- Hold down the VOL- key, connect the USB cable and release the VOL- key after 10 seconds.
- If you see the Android init messages, you haven't flashed the LK properly. You need to repeat **step 6a** above.

This step needs to be done every time you want to load a new LK and is not required for uploading new MF binaries or apps afterwards. Once you load Samraksh's LK, Android will no longer boot.

Should you need to go back to making Android the default boot, Samraksh has also provided "android-v5-boot.img". You only need to execute the following step to go back to Android:

- Enter '**fastboot flash aboot android-v5-boot.img**' at the command line

Step 7: Deploying the pre-compiled application

A pre-compiled application **MFout.bin** is available. You need to do the following steps to download and run this application on the board:

- Enter '**fastboot flash boot MFout.bin**' at the command prompt to load this application on the board.
- Once loaded, restart the board and you should see the LEDs flash in sequence.

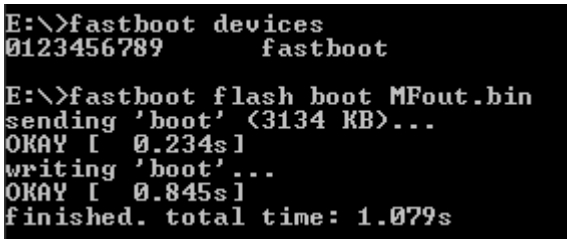


Figure 13: Loading the application on the ADAPT board

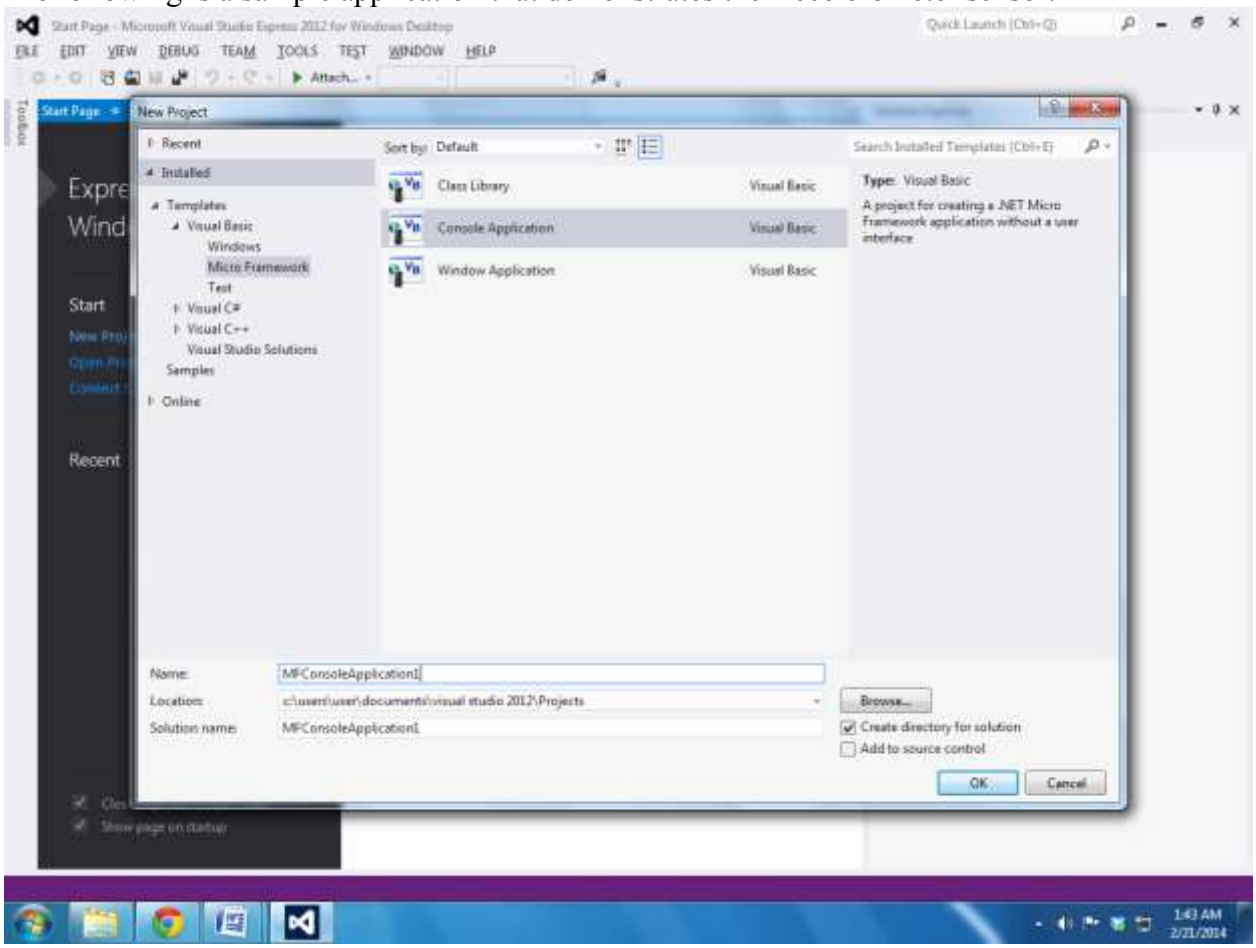
Remember: the parameter **aboot** is used to write to the boot partition while the parameter **boot** writes the image to the application partition.

First application for the ADAPT Board

We will develop, build and deploy our first application onto the ADAPT board in this section. For this we will use a demonstration C# code written for the accelerometer.

Sample Application: using Accelerometer

The following is a sample application that demonstrates the Accelerometer sensor.



This code imports a bitmap from the project resources, reads the Adapt accelerometer's X and Y coordinates and then displays the bitmap on the LCD screen based on the orientation of the Adapt core. This is done using the Samraksh Adapt library and dll as follows:

- The Samraksh dll needs to be added to the references in your C# project. You include the Microsoft and Samraksh libraries with the following commands
  - `using Microsoft.SPOT.Hardware;`
  - `using Samraksh.SPOT.Hardware.Adapt;`
  - You would then instantiate objects from Samraksh library. Here's the instantiation for the accelerometer: Accelerometer accelObject = new Accelerometer();
  - And for initializing the GPIO output pin with its initial state as high: OutputPort gpioObject = new OutputPort(Pins.GPIO_Debug_PIN3, true);

- You can now use the functions associated with that object.
- The accelerometer is initialized as follows: `accelObject.ADAPT_Accel_GetX();`
- And to set the test GPIO pin low: `gpioObject.Write(false);`

The following is the demonstration code for the accelerometer. You would find more samples and a detailed explanation of how to develop software for the ADAPTable System in the Programmer's guide.

```
namespace Graphics
{
      public class Program
      {
            // the accelerometer object
            private static
            Samraksh.SPOT.Hardware.Adapt.Accelerometer accel;

            public static void Main ()
            {
                  int16 xVal, yVal;             // x & y axis values
                  int rotation;

                  Bitmap LCD = new Bitmap (200, 142); // create disp. bitmap

                  Bitmap image = new Bitmap (   // bitmap resource -> display
                        Resources.GetBytes (Resources.BinaryResources.logo),
                        Bitmap.BitmapImageType.Bmp);

                  // Initialize accelerometer
                  accel = new Samraksh.SPOT.Hardware.Adapt.Accelerometer ();
                  accel.ADAPT_Accel_Init ();

                  while (true) {

                        // get the x & y accelerometer values
                        xVal = accel.ADAPT_Accel_GetX ();
                        yVal = accel.ADAPT_Accel_GetY ();

                        // determine simple rotation
                        if (xVal > 12000) rotation = 0;
                        else if (xVal < -12000) rotation = 100;
                        else if (yVal > 12000) rotation = 270;
                        else rotation = 90;

                        LCD.Clear ();           // Clear bitmap
                        LCD.RotateImage (rotation, 0, 0, image,
                                    0, 0, image.Width, image.Height, 0xff);
                        LCD.Flush ();           // draw image to screen
                  }
            }
      }
}
```

To compile a project into a .dat file:

1) Build your .NET MF Visual Studio project as outlined elsewhere in this guide.
2) Open a command prompt:
3) Navigate to the top-level directory of the MF Release e.g.:
   'cd c:\MF_ADAPT_Release-20140220v001'
4) Set up environment variables (needed once per shell).
   'setenv.cmd'
5) In your shell, navigate to the top-level directory of your Visual Studio project (where the .sln file is located).
6) Use the provided script (now in your path) to generate the .dat file:
   'generate_mf_dat project_name'
7) The above will produce "output.dat" in your current directory. This is the final product.

Load MF application to the ADAPT board

The MF binary and app are written to the ADAPT board as follows:

- Use Visual Studio to write and compile your application. This will generate a file "YourAppName.dat". This is the application binary.
- To create your output file enter the following at the Command Prompt: "**Copy tinyclr.bin /b + YourAppName.dat /b MFout.bin**"
- The output file MFout.bin will be flashed onto the board.
- Using fastboot you can load your new app with the command "**fastboot flash boot MFboot.bin**"
- You now need to reboot the board to use the new application. You do that by:
  - powering down the board
  - pressing VOL+ key and powering up the board
  - continue holding down the VOL+ key for 10 seconds
- Once a new application is loaded, it is stored in flash and will always restart whenever the board is powered on.
- Please be careful you don't flash MFBoot.bin to aboot otherwise you will overwrite the Samraksh LK and you would need to re-install the Modified Little Kernel section.

Frequently Asked Questions

Q1: I get an incompatible message while installing the Micro Framework SDK 4.3 with Visual Studio 2013.

The Visual Studio 2013 is incompatible with the Micro Framework SDK 4.3. We have filed a defect with Microsoft and are following up with them. In the meantime, you should:

- Download and install Visual Studio Express 2012 from http://www.microsoft.com/en-us/download/details.aspx?id=34673
- You now should be able to install the Micro Framework SDK 4.3 on this.

Q2: I have downloaded and installed the Android SDK but it does not work. I get a message JRE not found.

If you are installing the Android SDK on a new machine, you might find that it does not run because JRE is not installed. If you have a 64 bit machine, you should download the 64bit JRE (Java Runtime Environment) from http://www.java.com/en/download/manual.jsp and install it. You would also need to install a 32bit JRE for your browser.

References and Links

The following references were used to prepare this document.

- The SmartFTP page requires a login id which you should obtain from Samraksh
- .NET Micro Framework 4.3 SDK Documentation http://msdn.microsoft.com/en-us/library/bb417054.aspx
- http://www.cs.nyu.edu/artg/internet/Spring2006/lectures/DavidBuksbaum-ABriefIntroductionToCSharp.ppt
- http://www.cse.ohio-state.edu/~maung/cse459_csharp/Notes/CSE459_CSharp_01_CLROverview.ppt
- http://en.wikipedia.org/wiki/C_Sharp_(programming_language)
- .NET SDK: http://www.microsoft.com/downloads/details.aspx?FamilyID=fe6f2099-b7b4-4f47-a244-c96d69c35dec&DisplayLang=en
- MS Visual Studio C# Express Edition

Bibliography

[1] K. Zhou, J. C. Doyle and K. Glover, Robust and Optimal Control, Prentice Hall, 1995, p. 596.

[2] K. Zhou and J. C. Doyle, Essentials of Robust Control, Prentice Hall, 1997, p. 411.

[3] R. Zheng, J. C. Hou and L. Sha, "Asynchronous Wakeup for ad hoc networks," in *ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2003.

[4] F. Xue and P. R. Kumar, "Scaling laws for ad hoc wireless networks: An information theoretic approach," *Foundations and Trends in Networking ,* vol. 1, no. 2, pp. 145-270, 2006.

[5] X. Wang, L. Fu, X. Tian, Y. Bei, Q. Peng, X. Gan, H. Yu and J. Liu, "Converge-cast: On the capacity and delay tradeoffs," *IEEE Transactions on Mobile Computing,* vol. 99, no. 1, 2011.

[6] tcpdump/libpcap, "TCPDUMP/LIBPCAP public repository," 2010. [Online]. Available: http://www.tcpdump.org/. [Accessed 18 March 2013].

[7] P. L. a. R. M. Suresh, "ns-3-click: click modular router integration for ns-3," in *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, 2011.

[8] B. Sirkeci-Mergen and M. Gastpar, "On the broadcast capacity of high density wireless networks," in *Information Theory and Applications Workshop*, San Deigo, CA , 2007.

[9] G. Sharma, R. Maxumdar and N. Shroff, "Delay and capacity trade offs in mobile ad hoc networks: a global perspective," *IEEE Transactions on Networking,* vol. 15, no. 5, 2007.

[10] L. Rizzo, "Transparent acceleration of software packet forwarding using netmap," in *INFOCOM*, Pisa, Italy, 2012.

[11] J. Redi and R. Ramanathan, "The DARPA WNaN network architecture," in *Military Communications Conference (MILCOM)*, 2011.

[12] R. Ramanathan, R. Allan, P. Basu, J. Feinberg, G. Jakllari, V. Kawadia, S. Loos, J. Redi, C. Santivanez and J. Freebersyse, "Scalability of mobile ad hoc networks: theory vs practice," in *Military Communications Conference*, 2010.

[13] M. J. Neely and E. Modiano, "Capacity and delay tradeoffs for ad hoc mobile networks," in *International Conference on Broadband Networks (BROADNETS)*, 2004.

[14] S. Low, F. Paganini and J. Doyle, "Internet congestion control," *Control Systems,* vol. 22, no. 1, pp. 28-43, 2002.

[15] X. Lin and N. Shroff, "Towards achieving the maximum capacity in large mobile wireless networks under delay constraints," *Journal of Communications and Networks,* 2004.

[16] J. Li, C. Blake, D. S. J. D. Couto, C. Hu, H. I. Lee and R. Morris, "Capacity of ad hoc wireless networks," in *ACM International Conference on Mobile Computing and Networking (MobiCom 01)*, 2001.

[17] V. Kulathumani and A. Arora, "Distance sensitive snapshots in wireless sensor networks," in *Proceedings of the 11th international conference on Principles of distributed systems*, 2007.

[18] G. Kolata, "At last, shout of 'Eureka!' in age-old math mystery," [Online]. Available: http://www.nytimes.com/1993/06/24/us/at-last-shout-of-eureka-in-age-old-math-mystery.html. [Accessed 2 April 2013].

[19] E. e. a. Kohler, "The Click modular router," *ACM Transactions on Computer Systems (TOCS),* pp. 263-297, 2000.

[20] K.-H. Kim, Y. Zhu, R. Sivakumar and H.-Y. Hsieh, "A Receiver-Centric Transport Protocol for Mobile Hosts with Heterogeneous Wireless Interfaces," *Wireless Networks,* vol. 11, no. 4, pp. 363-382, July 2005.

[21] A. Keshavarz-Haddad, V. Ribeiro and R. Riedi, "Broadcast capacity in multihop wireless networks," in *ACM International Conference on Mobile Computing and Networking (MobiCom 06)*, NewYork, NY, 2006.

[22] A. Kandhalu, K. Laskhman and R. Rajkumar, "U-connect: A low-latency energy-efficient asynchronous neighbor discovery protocol," in *9th International Conference on Information Processing in Sensor Networks*, NewYork, USA, 2010.

[23] P. Jacquet, P. Mühlethaler, T. Clausen, A. Laouiti, A. Qayyum and L. Viennot, "Optimized link state routing protocol for ad hoc networks," in *International Multi Topic Conference, IEEE INMIC*, 2001.

[24] J. Hui and D. Culler, "IP is dead, long live IP for wireless sensor networks," in *6th International Conference on Embedded Networked Sensor Systems*, 2008.

[25] T. Henderson, "12.2.1 The Packet Class," 11 November 2011. [Online]. Available: http://www.isi.edu/nsnam/ns/doc/node132.html. [Accessed 18 March 2013].

[26] R. Hekmat and P. Mieghem, "Interference in wireless multi-hop ad-hoc networks and its effect on network capacity," *Wireless Networks,* vol. 10, no. 4, pp. 389-399, 2004.

[27] P. Gupta and P. R. Kumar, "The capacity of wireless networks," *IEEE Transactions on Information Theory,* vol. 46, no. 2, pp. 388-404, 2000.

[28] H. E. Gamal, "On the scaling laws of dense wireless sensor networks: the data gathering channel,"

IEEE Transactions on Information Theory, vol. 51, no. 3, pp. 1229-1234, 2005.

[29] P. Dutta and D. Culler, "Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications," in *ACM conference on Embedded Networking sensor Systems (SenSys)*, 2008.

[30] J. C. Doyle and M. Csete, "Architecture, constraints, and behavior," *Procedings of the National Academy of Science of United States of America,* vol. 108, pp. 15624-15630, 2011.

[31] I. D. Chakeres and J. P. Macker, "Mobile ad hoc networking and the IETF -- IETF 69," in *ACM SIGMOBILE Mobile Computing and Communications Review*, 2007.

[32] H. Cao, K. W. Parker and A. Arora, "O-MAC: A receiver centric power management protocol," in *Proceedings of the 2006 14th IEEE International Conference on Network Protocols*, Santa Barbara, California, 2006.

[33] L. Burbank, W. Kasch and J. and Ward, An Introduction to Network Modeling and Simulation for the Practicing Engineer, Wiley-IEEE Press, 2011.

[34] J. G. Andrews, N. Jindal, M. Haenggi, R. Berry, S. A. Jafar, D. Guo, S. Shakkottai, R. W. Heath, M. Neely, S. Weber and A. Yener, "Rethinking information theory for mobile ad hoc networks," *IEEE Communications Magazine,* vol. 46, no. 12, pp. 94-101, 2008.

[35] N. Alon, C. Avin, M. Koucky and G. Kozma, "Many random walks are faster than one," in *20th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2008.

[36] D. L. Alderson and J. C. Doyle, "Contrasting views of complexity and their implications for network-centric infrastructures," *Systems, Man, and Cybernetics Society,* vol. 40, no. 4, pp. 839 - 852, July 2010.

[37] D. e. a. Aguayo, “MIT roofnet,” in *Proceedings of the International Conference on Mobile Computing and Networking*, 2003.

[38] "TUN/TAP," [Online]. Available: http://en.wikipedia.org/wiki/TUN/TAP. [Accessed April 2013].

[39] "The Named Data Networking Project," [Online]. Available: http://www.named-data.net/.

[40] "Freestyle Chess," [Online]. Available: http://freestyle-chess.webs.com/. [Accessed 2 April 2013].

[41] "Eli Whitney - firearms and the birth of standardization," [Online]. Available: http://inventors.about.com/cs/inventorsalphabet/a/machine_2.htm. [Accessed 29 April 2013].

[42] Cesar A. Santivanez; Ram Ramanathan, "Hazy sighted link state (HSLS) routin: A scalable link state algorithm," BBN Technical Memorandum No.1301, 2001.

[43] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss and P. Levis, "Collection Tree Protocol," in *7th ACM*

Conference on Embedded Networked Sensor Systems, 2009.

[44] S. Madden, M. Franklin, J. Hellerstein and W. Hong, "TAG: A tiny aggregation service for ad hoc sensor networks," in *ACM/USENIX Symposium on Operating Systems Design and Implementation*, 2002.

[45] U. Feige, "A tight upper bound on the cover time for random walks on graphs," *Random structures and algorithms,* vol. 6, no. 1, pp. 51--54, 1995.

[46] U. Feige, "A tight lower bound for the cover time of random walks on graphs," *Random structures and algorithms,* vol. 6, no. 4, pp. 433--438, 1995.

[47] A. Frieze and C. Cooper, "The cover time of random regular graphs," *SIAM Journal of Discrete Mathematics,* vol. 18, no. 4, pp. 728--740, 2005.

[48] L. Lovascz, "Random walks on graphs: A survey," *Combinatorics, Paul Erdos 80,* 1993.

[49] C. Cooper, A. Frieze and T. Radik, "Multiple random walks in random regular graphs," *SIAM Journal of Discrete Mathematics,* vol. 23, no. 4, pp. 1738--1761, 2009.

[50] G. Ercal and C. Avin, "On the cover time of random geometric graphs," in *Automata, Languages, and Programming*, 2005.

[51] C. Avin and B. Krishnamachari, "The power of choice in random walks: An empirical study," in *International Symposium on Modeling, Analysis and Simulation of Wireless Mobile Systems*, 2006.

[52] M. Chandy and L. Lamport, "Distributed snapshots: determining global states of distributed systems," *ACM Transactions on Computer Systems,* vol. 3, no. 5, pp. 63-75, 1985.

[53] S. D. Servetto, "Sensing lena-massively distributed compression of sensor images," in *IEEE International Conference on Image Processing*, 2003.

[54] F. Dehne, A. Ferreira and A. Rau-Chaplin, "Parallel fractional cascading on hypercube multiprocessors," *Computational Geometry Theory and Applications,* vol. 2, pp. 144-167, 1992.

[55] J. Gao, L. J. Guibas, J. Hershberger and L. Zhang, "Fractionally cascaded information in a sensor network," in *Information Processing in Sensor Network*, 2004.

[56] R. Sarkar, X. Zhu and J. Gao, "Hierarchical spatial gossip for multi-resolution representations in sensor networks," in *Information Processing in Sensor Networks (IPSN)*, 2007.

[57] G. Pei, M. Gerla and T.-W. Chen, "Fisheye state routing in mobile ad hoc networks," in *ICDCS Workshop on Wireless Networks*, 2000.

[58] V. Kulathumani and Y. Fallah, "An infrastructure-less vehicular traffic information service with

distance-sensitive precision," in *IEEE Vehicular Technology Conference (VTC)*, 2012.

[59] Microsoft, ".NET Micro Framework Platform SDK," [Online]. Available: http://msdn.microsoft.com/en-us/library/jj610646(v=vs.102).aspx.

[60] Smart FTP, "https://lsr.smartfile.com/," [Online].

[61] D. Buksbaum, "A Brief Introduction to C#," [Online]. Available: http://www.cs.nyu.edu/artg/internet/Spring2006/lectures/DavidBuksbaum-ABriefIntroductionToCSharp.ppt.

[62] R. Crawfis, "Programming in C#," [Online]. Available: http://www.cse.ohio-state.edu/~maung/cse459_csharp/Notes/CSE459_CSharp_01_CLROverview.ppt.

[63] Wikipedia, "C Sharp (programming Language)," [Online]. Available: http://en.wikipedia.org/wiki/C_Sharp_(programming_language).

[64] Microsoft, ".NET Framework 2.0 Software Development Kit (SDK) (x86)," [Online]. Available: http://www.microsoft.com/en-us/download/details.aspx?id=19988.

[65] Microsoft, "Visual Studio Express," [Online]. Available: http://www.visualstudio.com/en-US/products/visual-studio-express-vs.

[66] Microsoft, "#define VISUAL_STUDIO," [Online]. Available: http://msdn.microsoft.com/vstudio/.

[67] W. Zeng, J. Li, K. W. Parker and A. Arora, "Comparative Factor Analysis and Experimental Validation of Low Power MAC Performance," The Ohio State University Technical Report OSU-TR-2010-21, 2010.

[68] Wikipedia, "Self-stabilization," [Online]. Available: http://en.wikipedia.org/wiki/Self-stabilization.

[69] A. Arora and M. Gouda, "Distributed Reset," *IEEE Transactions on Computers,* vol. 43, no. 9, 1994.