Porting the Arduino Library to the Cypress PSoC in PSoC Creator

Matt Durak

November 11, 2011

Design Team 1

Abstract

Arduino, the open-source electronic platform is a useful tool to hobbyists in building embedded systems. It provides an easy to use library which includes components to work with an Ethernet board, called the Ethernet shield. PSoC is a programmable system-on-chip made by Cypress Semiconductor. It is a very flexible platform which includes an ARM Cortex M3 processor. This application note includes the steps necessary to port parts of the Arduino library to the PSoC in order to use Arduino software and hardware, known as shields, with the PSoC. The note will cover many issues which must be overcome in porting this software.

Keywords

PSoC, Arduino, C++, C, Library, Software, Porting, PSoC Creator, Ethernet shield

Introduction

Arduino Library

The Arduino is an open-source electronics hardware platform that is designed primarily for students and hobbyists (1). Arduino provides the schematics to build the hardware, as well as kits which can be pre-assembled or just include the parts. This application note will focus on the software for Arduino. Arduino has its own open-source development environment based on Wiring, a platform for programming electronics (2). The software library used by Arduino is written in C++ and is also open-source and freely available (3).

This library is composed of a low layer which communicates directly with hardware registers and provides an abstraction for programmers to set whether a pin is an input or an output and to read and write to those pins. This low layer also handles the other hardware components on the Arduino, such as memory access, timers, pulse width modulators, and several forms of communication. The library also provides higher-level abstractions on top of this in order to make development easier.

The library contains sub-libraries for several Arduino daughterboards, which are called shields. This note will focus on the Ethernet shield as an example, although the note should be applicable to most

components of the library. The Ethernet library itself also has a layer which defines the low level registers to communicate with over serial peripheral interface (SPI), as well as high level abstractions which make application code easier to write (4). The Ethernet library uses a separate SPI library component to handle the communication. This library provides an abstract interface to transmit bytes of data. It wraps the actual low level hardware communication in its implementation.

PSoC Development

The Cypress Programmable System-on-Chip (PSoC), is a configurable piece of hardware which contains a number of programmable virtual components in order to implement a wide range of hardware features. This application note will focus on the PSoC 5, the latest family of PSoC. This version of the board contains an ARM Cortex M3 processor (5).

The development environment used by the PSoC is called PSoC Creator. This software is a full development environment for the PSoC. It contains a schematic layout tool in order to configure the programmable virtual hardware components. It also has a full C language IDE for developing code to run on the PSoC processor (6). This application note will focus on the programming side of PSoC Creator.

Objective

The objective of this application note is to provide the steps necessary to take the Arduino software libraries, and compile them for the PSoC. This would allow a developer using a PSoC to run Arduino code and to possibly interface with an Arduino shield. The objective is not to show how to actually interface the PSoC hardware with an Arduino shield or to use the schematic layout tool in PSoC Creator. This application note assumes that the reader has acquired the latest development version of Arduino, found on Github (7).

Issues

There are several issues to be aware of when porting code from one platform to another. The primary concerns in this case are the programming language, the standard libraries used by the code, and the implementation of the low-level details.

The issue with the programming language is mostly an minor incompatibility when using PSoC Creator. The Arduino libraries are primarily written in C++, although a few files are written in C. The PSoC Creator environment only officially supports C. However, PSoC Creator ships with the complete GNU Compiler Collection (GCC), which includes compilers for both C and C++. With a few minor modifications, PSoC Creator can compile C++ code. Another issue is the use of C++ in an embedded environment with very limited memory. C++ uses many advanced features such as polymorphism and virtual methods, templates, and run time type information which can add code bloat if used improperly (8). Fortunately, the standard Arduino board has a flash memory size of 32 KB, which is the same as the simplest PSoC 5 board. This means that the Arduino library has already been optimized to work with a similar memory constraint.

The next issue with the Arduino library code, is that the development library used by Arduino for its AVR ATMega processor uses standard AVR libraries with GCC. The PSoC uses the same version of GCC for embedded development; however it does not contain the AVR library used by Arduino. This requires some modification of the code to use the standard libraries available to PSoC.

Finally, the primary issue which will be addressed in this application note lies in the actual hardware implementations and low-level details of the Arduino library. This part of the code is very specific to the Arduino platform. The implementations will need to be re-written in order to make the library work on the PSoC platform.

Steps

Importing Arduino Sources

The process of porting the Arduino code begins with creating a new project in PSoC Creator. Next, it is necessary to create the digital and analog pins as well as any hardware components such as SPI and UART which will be used by the Arduino code. That process should be completed separately. The remaining steps assume that the PSoC Creator project contains a project with the schematic components necessary to emulate an Arduino board.

First, the following sources from the Arduino folder /hardware/cores/arduino/ must be added to the project: Arduino.h, Client.h, IPAddress.cpp, IPAddress.h, Print.cpp, Print.h, Printable.h, Server.h, Stream.cpp, Stream.h, Udp.h, WMath.cpp, binary.h, and new.cpp. It is recommended to add these to a new folder within the PSoC project in order to keep them separated from application code. Next, all files within the directory /libraries/Ethernet/, /libraries/Ethernet/utility/, and /libraries/SPI/ must be added to the project.

At this point, the project will probably build. This is because PSoC Creator does not compile files ending with the ".cpp" file extension. Fixing this requires manually modifying the ".cyprj" file. Open this file and find every ".cpp" file with a line like the following in Figure 1 and change the build action from "NONE" to "C_FILE" as in Figure 2. After repeating this for every C++ source file, save the file and PSoC Creator should reload the file. At this point, the files will all be built, but the compilation will fail.

```
version="3">
<CyGuid_31768f72-0253-412b-af77-e7dba74d1330 type_name="CyDesigner.Common.ProjMgmt.Moc
version="2" name="Dhcp.cpp" persistent="./lib/Dhcp.cpp">
<Hidden v="False" />
</CyGuid_31768f72-0253-412b-af77-e7dba74d1330>
<build_action v="NONE" />
<PropertyDeltas />
</CyGuid_8b8ab257-35d3-4473-b57b-36315200b38b>
<CyGuid_8b8ab257-35d3-4473-b57b-36315200b38b type_name="CyDesigner.Common.ProjMgmt.Moc
version="3">
<CyGuid_31768f72-0253-412b-af77-e7dba74d1330 type_name="CyDesigner.Common.ProjMgmt.Moc
version="2" name="Dhcp.h" persistent="./lib/Dhcp.h">
<Hidden v="False" />
</CyGuid_31768f72-0253-412b-af77-e7dba74d1330>
<build_action v="NONE" />
```

Figure 1 Project file before modification

```
version="2" name="Dhcp.cpp" persistent="./lib/Dhcp.cpp">
<Hidden v="False" />
</CyGuid_31768f72-0253-412b-af77-e7dba74d1330>
<build_action v="C_FILE" />
<PropertyDeltas />
</CyGuid_8b8ab257-35d3-4473-b57b-36315200b38b>
<CyGuid_8b8ab257-35d3-4473-b57b-36315200b38b type_name="CyDesig
version="3">
```

Figure 2 Project file after modification

Fixes to get C++ Working

Now that C++ files can be built by PSoC Creator, some changes need to be made to get the project to compile. The first change is to modify the build settings of the project under the “Compiler” section and “Command Line” subsection. That command should include “-I./lib” where “lib” is the folder name where the Arduino sources are located. The command must also include “-fno-rtti” in order to disable run time type information in C++. These two command switches must both be added to make the library work.

Some of the other idiosyncrasies with using C++ are handled nicely by the Arduino library. Take a look at the file New.cpp to see some of the implementation used in order to get C++ working on an embedded system (9). This file is required to get the library to compile and link in PSoC Creator.

At this point, the remaining issues are due to including files that do not exist, using Arduino implementations, and some misnamed functions. However, one final item to note is that when working with C++ and including a C header, it is important to include the code in Listing 1 below. Without the ‘extern “C”’, the code will compile, but it will not link due to C++ name mangling (10). It is important to include this section of code whenever C header files are needed.

```
#ifdef __cplusplus
extern "C"{
#endif

#include <device.h>

#ifdef __cplusplus
}
#endif
```

Listing 1 Including C headers in C++

Getting the Library to Compile

The first step to getting the library to compile is to trim out any missing headers and to remove any Arduino specific implementations. The project already only includes a small subset of the entire library, but this next step will polish those sources to get them to work.

All files should have any reference to “AVR” includes removed (Listing 2). This is the standard library used by the AVR ATMega and it is not available for PSoC. The Arduino.h file has these include statements which should be removed. Likewise, any conditional statements using AVR, such as that in Listing 3, should be removed so that only the “else” portion remains.

The following functions in Arduino.h should be removed: init, pinMode, digitalWrite, digitalRead, analogRead, analogReference, analogWrite, micros, delay, delayMicroseconds, pulseIn, shiftOut, shiftIn, attachInterrupt, detachInterrupt, setup, loop, makeWord, tone, and noTone. The functions marked extern should also be removed. The “defines” starting at analogInPinToBit and continuing through TIMER5C should all be removed. Finally, the “includes” for WCharacter.h, WString.h, HardwareSerial.h, and pins_arduino.h should be removed. Within the extern “C” section, an “include” to CyLib.h must be added. This will provide useful functions which were removed from Arduino.h The final source code is included in the appendix.

```
#include <avr/pgmspace.h>
#include <avr/io.h>
#include <avr/interrupt.h>
```

Listing 2 Example AVR includes to remove

```
#if defined(__AVR_ATtiny24__) || defined(__AVR_ATtiny44__) ||
defined(__AVR_ATtiny84__) || defined(__AVR_ATtiny25__) ||
defined(__AVR_ATtiny45__) || defined(__AVR_ATtiny85__)
#define DEFAULT 0
#define EXTERNAL 1
#define INTERNAL 2
#else
#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
#define INTERNAL1V1 2
#define INTERNAL2V56 3
#else
#define INTERNAL 3
#endif
```

Listing 3 Example AVR conditional statements

The next file to modify is Print.h and Print.cpp. Each of these files must be modified to remove the "includes" to WString.h and instead include string.h. All methods that take a "String" or a "__FlashStringHelper" should be removed (Listing 4). These are simply function overloads and it is much easier to not include WString.h. The classes will still work with char* arrays.

```
    size_t print(const __FlashStringHelper *);
    size_t print(const String &);

    //…

    size_t println(const __FlashStringHelper *);
    size_t println(const String &s);
```

Listing 4 Functions to remove

The next file to modify is WMath.cpp. This file must have an "include" to Arduino.h added. It must also change "srandom" and "random" to the standard "srand" and "rand" function calls. Finally, this is a good place to implement the "millis()" function. This function is used in Arduino to report the number of milliseconds which have elapsed since the device was powered on. It can be implemented using a counter in PSoC, but that is left to the reader to implement. This finishes the main parts of the Arduino library; next the Ethernet library must be fixed.

One change to make throughout several files here and any others added later, is to change all occurrences of "delay()" to "CyDelay()". This will be a simple find and replace operation, as the "delay()" function is declared in Arduino.h and now Arduino.h includes CyLib.h which declares "CyDelay()". This must be fixed in Dhcp.cpp, EthernetClient.cpp, Dns.cpp, and w5100.h.

It should be noted that there are two util.h files. These files should be merged from the main one in hardware to include the one found in the Ethernet library. The Ethernet library version of util.h defines the functions htonl, htons, ntohl, and ntohs in order to convert numbers from host to network endians

(which define the order of the bits in a number (11)). The merged version of util.h can be found in the appendix.

The final part of the Ethernet library to fix is the w5100.cpp and w5100.h. The w5100.h file must remove references to the avr headers and add includes to cytypes.h, util.h, and the digital pin used for the Ethernet slave select. The methods initSS, setSS, and resetSS must be changed in order to properly set the SPI slave select pin on the PSoC. The new implementation will not need to initialize the slave select, but it will need to write a "0" to the pin in setSS and write a "1" in resetSS. This code is shown in Listing 5 below. The w5100.cpp file must change delay to "CyDelay" as well as include CyLib.h for this function. There is also a required change in the function "void W5100Class::read_data(SOCKET s, volatile uint8_t *src, volatile uint8_t *dst, uint16_t len)". There is a compiler error, but the changed code in Listing 6 below will fix this.

```
inline static void initSS()    {};
inline static void setSS()     { D10_Write(0); /*PORTB &= ~_BV(2);*/ };
inline static void resetSS()   { D10_Write(1); /*PORTB |=  _BV(2);*/ };
```

Listing 5 The slave select code

```
void W5100Class::read_data(SOCKET s, volatile uint8_t *src, volatile
uint8_t *dst, uint16_t len)
{
  uint16_t size;
  const uint16_t newsrc = (uintptr_t) src; // FIXED
  uint16_t src_mask;
  uint16_t src_ptr;

  src_mask = newsrc & RMASK; // FIXED
  src_ptr = RBASE[s] + src_mask;

  if( (src_mask + len) > RSIZE )
  {
    size = RSIZE - src_mask;
    read(src_ptr, (uint8_t *)dst, size);
    dst += size;
    read(RBASE[s], (uint8_t *) dst, len - size);
  }
  else
    read(src_ptr, (uint8_t *) dst, len);
}
```

Listing 6 Fixing a compiler error

The final library component to fix is the SPI. The file SPI.cpp requires major changes to the implementation. First the "include" to pins_arduino.h is removed. Next, all function implementation bodies may be removed. The only useful methods are "begin" and "end". The body for begin should simply call "SPIM_1_Start();" and the body for end should call "SPIM_1_Stop();". The other functions are configurations of SPI that must be made in the schematic layout mode. They are not necessary for the

Ethernet library. The file SPI.h must remove the “include” to the avr header and add an “include” to the automatically generated SPI master header (SPIM_1.h by default). In this file, the bodies of attachInterrupt and detachInterrupt can be commented out temporarily. They are not used in the Ethernet code and can be implemented at a later point when they are needed. The most important function is transfer. This function handles actual SPI communication. The body of the function should be replaced with that in Listing 7. This uses the PSoC implementation of SPI. This is the final and most important implementation change to make.

```
byte SPIClass::transfer(byte _data) {
  SPIM_1_WriteTxData(_data);

  while (!(SPIM_1_ReadTxStatus() & SPIM_1_STS_SPI_DONE));
  while (SPIM_1_GetRxBufferSize() == 0);

  return SPIM_1_ReadRxData();
}
```

Listing 7 The SPI transfer method

Results

After following the steps of the application note, and creating the proper virtual components in the schematic layout tool, the Arduino library can be built and programmed onto the PSoC. This is a subset of the library including the base components and the Ethernet and SPI libraries. Code that worked on Arduino can be copied into PSoC Creator and compiled to run on the PSoC with very minor modification. After building the hardware to connect the PSoC to an Arduino Ethernet shield, the Arduino Ethernet demos can be run on the PSoC with minor modification.

Conclusions

Porting code from one embedded system to another can be a challenging task. There are many small implementation details which need to be considered. Likewise, it can be tricky to get C++ code working in an embedded environment which expects C code. However, the end result of this port which leaves the abstract interface intact is that code from one platform can be ported to the other platform for free. Porting the library is an investment which will make future work much easier.

References

1. **Arduino Team.** Arduino - Homepage. *Arduino.* [Online] September 24, 2011. [Cited: November 9, 2011.] http://www.arduino.cc/.

2. **Wiring.** About Wiring. *Wiring.* [Online] November 9, 2011. [Cited: November 9, 2011.] http://wiring.org.co/about.html.

3. **Mellis, David A.** Introduction. *Arduino.* [Online] December 23, 2009. [Cited: November 9, 2011.] http://arduino.cc/en/Guide/Introduction.

4. —. Arduino. *Ethernet.* [Online] February 10, 2009. [Cited: November 9, 2011.] http://arduino.cc/en/Reference/Ethernet.

5. **Cypress Semiconductor.** PSoC 5 Introduction. *Cypress.* [Online] November 9, 2011. [Cited: November 9, 2011.] http://www.cypress.com/?id=2233&rID=37591.

6. —. PSoC Creator Overview. *Cypress.* [Online] November 9, 2011. [Cited: November 9, 2011.] http://www.cypress.com/?id=2494.

7. **Arduino.** Arduino. *Github.* [Online] November 9, 2011. [Cited: November 9, 2011.] https://github.com/arduino/Arduino.

8. **Neundorf, Alexander.** C vs. C++ for embedded development. *KDE Blog.* [Online] June 7, 2005. [Cited: November 9, 2011.] http://blogs.kde.org/node/1138.

9. **Zed, Rob.** GCC C++ Link problems on small embedded target. *Zedcode.* [Online] February 17, 2007. [Cited: November 9, 2011.] http://zedcode.blogspot.com/2007/02/gcc-c-link-problems-on-small-embedded.html.

10. **Cline, Marshall.** Why is the linker giving errors for C/C++ functions being called from C++/C functions? *C++ FAQ Lite.* [Online] July 28, 2011. [Cited: November 9, 2011.] http://www.parashift.com/c++-faq-lite/mixing-c-and-cpp.html#faq-32.7.

11. **Hall, Brian.** Beej's Guide to Network Programming. *Beej.us.* [Online] September 8, 2009. [Cited: November 9, 2011.] http://beej.us/guide/bgnet/output/html/multipage/htonsman.html.

Appendix

Arduino.h

```
#ifndef Arduino_h
#define Arduino_h

#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "binary.h"

#ifdef __cplusplus
extern "C"{
#endif
```

```
#include <CyLib.h>

#define HIGH 0x1
#define LOW  0x0

#define INPUT 0x0
#define OUTPUT 0x1

#define true 0x1
#define false 0x0

#define PI 3.14159265358979323846264338327950
#define HALF_PI 1.57079632679489661923132169163975
#define TWO_PI 6.28318530717958647692528676655900
#define DEG_TO_RAD 0.017453292519943295769236907684886
#define RAD_TO_DEG 57.295779513082320876798154814105

#define SERIAL  0x0
#define DISPLAY 0x1

#define LSBFIRST 0
#define MSBFIRST 1

#define CHANGE 1
#define FALLING 2
#define RISING 3

#define INTERNAL 3
#define DEFAULT 1
#define EXTERNAL 0

// undefine stdlib's abs if encountered
#ifdef abs
#undef abs
#endif

#define min(a,b) ((a)<(b)?(a):(b))
#define max(a,b) ((a)>(b)?(a):(b))
#define abs(x) ((x)>0?(x):-(x))
#define constrain(amt,low,high)
((amt)<(low)?(low):((amt)>(high)?(high):(amt)))
#define round(x)     ((x)>=0?(long)((x)+0.5):(long)((x)-0.5))
#define radians(deg) ((deg)*DEG_TO_RAD)
#define degrees(rad) ((rad)*RAD_TO_DEG)
#define sq(x) ((x)*(x))

#define interrupts() sei()
#define noInterrupts() cli()

#define clockCyclesPerMicrosecond() ( F_CPU / 1000000L )
#define clockCyclesToMicroseconds(a) ( ((a) * 1000L) / (F_CPU / 1000L) )
#define microsecondsToClockCycles(a) ( ((a) * (F_CPU / 1000L)) / 1000L )

#define lowByte(w) ((uint8_t) ((w) & 0xff))
#define highByte(w) ((uint8_t) ((w) >> 8))
```

```
#define bitRead(value, bit) (((value) >> (bit)) & 0x01)
#define bitSet(value, bit) ((value) |= (1UL << (bit)))
#define bitClear(value, bit) ((value) &= ~(1UL << (bit)))
#define bitWrite(value, bit, bitvalue) (bitvalue ? bitSet(value, bit) :
bitClear(value, bit))


typedef unsigned int word;

#define bit(b) (1UL << (b))

typedef uint8_t boolean;
typedef uint8_t byte;

unsigned long millis(void);

#ifdef __cplusplus
} // extern "C"
#endif

#ifdef __cplusplus

// WMath prototypes
long random(long);
long random(long, long);
void randomSeed(unsigned int);
long map(long, long, long, long, long);

#endif

#endif
```

Util.h

```
#ifndef UTIL_INCLUDED
#define UTIL_INCLUDED

#define _BV(bit) (1 << (bit))

#ifdef __cplusplus
extern "C"{
#endif

#include <CyLib.h>

#ifdef __cplusplus
}
#endif

#define htons(x) CYSWAP_ENDIAN16(x)
#define ntohs(x) htons(x)

#define htonl(x) CYSWAP_ENDIAN32(x)
#define ntohl(x) htonl(x)
```

```
#endif
```