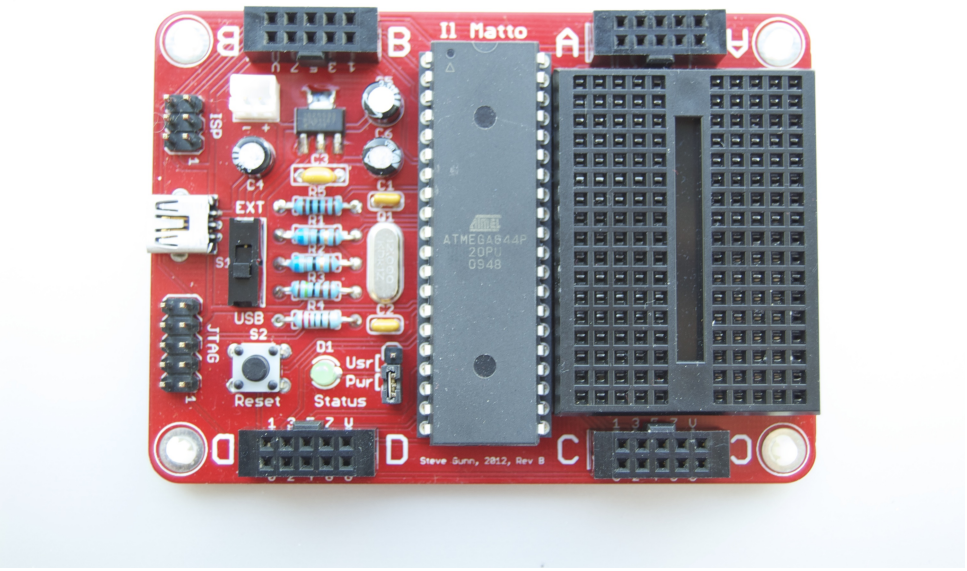
X2

An introduction to Circuit Construction and Testing

This exercise introduces techniques for constructing an electronic circuit. You will use these techniques to build a high-performance, low-power 8-bit microcontroller circuit using a double-sided PCB. The exercise then introduces techniques for testing new hardware using laboratory test equipment, alongside exploring some of the capabilities of the microcontroller circuit.



Schedule

Preparation Time	:	6 hours
Lab Time	:	6 hours

Items provided

Tools	:	Soldering Iron, Iron stand, Solder, Desoldering tool, Blue-tac
Components	:	Il Matto Kit, C232HM cable, Hook-up wire
		[*Components to be retained by the student after the exercise.*]
Equipment	:	Oscilloscope [12], Logic Analyser [9], Bench PSU [7], Multimeter [1]
Software	:	`AVRDUDE`, `avr-gcc`, `UrJTAG`

Items to bring

Electronics toolkit (Solder wick, Eye-glasses, Small-nose Pliers, Tweezers, Side-cutters, Flux pen)

×10 oscilloscope probe

Identity card

Laboratory logbook

Version: September 19, 2012

Steve R. Gunn
Electronics and Computer Science
University of Southampton

Contents

Before entering the laboratory you should read through this document and complete the preparatory tasks detailed in section 2. It is noteworthy that this is an extensive two day exercise, where you will spend one day in preparation and one day in the laboratory – good preparation is essential for a successful outcome in the laboratory.

> **Academic Integrity** – *If you wish you may undertake the preparation jointly with other students. If you do so it is important that you acknowledge this fact in your logbook. Similarly, you will probably want to use sources from the internet to help answer some of the questions. Again, record any sources in your logbook.*

Unlike most Part I Electronics Laboratory exercises, you will be working on your own in the laboratory, rather than with a partner. During the exercise you should use your logbook to record your observations, which you can refer to in the future – perhaps to write a formal report on the exercise, or to remind you about the procedures. As such it should be legible, and observations should be clearly referenced to the appropriate part of the exercise. As a guide the ✎ symbol has been used to indicate a mandatory entry in your logbook. However, you should always record additional observations whenever something unexpected occurs, or when you discover something of interest.

At the end of the exercise you will be marked on your constructed circuit, your ability in using the test equipment, and the contents of your logbook. You will need to demonstrate your circuit working and show that you have mastered the basics in using the test equipment.

You will keep your circuit, and subsequently use it in other parts of the programme.

Notation

This document uses the following conventions:

An entry should be made in your logbook

Care should be exercised

```
command input
```

Command to be entered at the command line

```
output response
```

Response produced at the command line

Remarkable text

A point of note

1 Introduction

This lab introduces you to three aspects of electronics:

Construction soldering techniques for circuit assembly

Testing using lab test equipment to test and verify the correct operation of circuits

Microcontrollers working with modern embedded systems

1.1 Outcomes

At the end of the exercise you should be able to:

- assemble components onto a printed circuit board (PCB)
- use a multimeter to perform basic circuit checks
- use the bench PSU to limit the current supplied to a circuit
- use an oscilloscope to observe circuit behaviour
- use a logic analyser to observe circuit behaviour
- use the command line to compile and download programs to a microcontroller
- understand some of the limitations of test equipment

1.2 Overview

The object of this exercise is for you to build and test a microcontroller development board based on an Atmel AVR microcontroller. A microcontroller is a small computer with associated peripherals all located together in one package. It is a versatile piece of electronic hardware which you can use for many different purposes by re-programming the device with a different program. In the exercise you will use a cross-compiler[1] running on the PC to transform programs written in `C` (a high-level programming language) to corresponding programs in machine code instructions understood by the microcontroller.

The emphasis of the exercise is to develop practical skills for construction and testing. At this stage it is not a requirement that you understand the operation of the circuit in detail. However, you will need a basic understanding and you will find further details about the circuit in appendix A and test programs in appendix C.

You will learn and apply the techniques of soldering to assemble the components onto a double-sided PCB. After the circuit is built it is essential to test it to see that it meets the specification and operates correctly. A circuit can fail for a number of reasons, including poor soldering (dry joints or

[1] terminology indicates that the program will not run on the platform that the program was compiled on.

solder bridges), components in the wrong place, components the wrong way round, faulty components or a poorly fabricated PCB. To test the circuit you will use a bench PSU, a digital multimeter, an oscilloscope and a logic analyser. This will give you the opportunity to learn how to use these instruments. The signals that you observe will include simple repetitive signals as well as more complex waveforms, that require more careful configuration of the test equipment to be observed clearly. During the testing phase you will also use the command line to compile some simple test programs and download these to your embedded target. This will give you valuable experience in executing programs from the command line.

The exercise has been designed to be platform neutral (all the software tools required are available for Linux/Mac/Windows[2]) and to use open software (source code for the software is available and it is free to use). After the exercise is complete you should setup this environment on your own computer so that you can develop programs for the AVR. You will find details on how to install the toolchain for developing programs for the microcontroller in the separate installation document [2].

[2] Nevertheless, it is not recommended to use a 64-bit version of Windows as you may have difficulty with the `libusb` driver installation.

2 Preparation

Read through the Laboratory Handbook statement on Safety and safe working practices and your copy of the Risk Assessment form. Make sure that you understand how to work safely.

2.1 Construction

Watch the video on How to Solder [8].

1. What is the typical composition of solder?
2. Why is an alloy used for solder?
3. What is flux and why is it important?
4. What is a dry joint and how can it be avoided?
5. What is a solder bridge and how can it be removed?
6. What safety precautions should you take when soldering?
7. What is a ground plane and why is it used?
8. What is a thermal relief and why is it helpful?
9. Why is it desirable to have more than a single layer on a PCB?
10. What is a via and what different types are available?

2.2 Testing

Please make sure that you state any assumptions when answering the following questions.

1. What precautions should be taken when handling ICs and why?
2. It is possible to insert `IC1` in two different orientations. How do you determine which is the correct one?
3. Estimate the voltage drop across the LED (`D1`) when it is illuminated.
4. Estimate the expected current range drawn by the circuit for the following configurations:

IC1	JP3	Minimum current	Maximum current
Not installed	Usr		
Not installed	Pwr		
Installed	Usr		
Installed	Pwr		

5. Determine suitable values for the voltage and current limits of your bench PSU for testing the voltage regulator in section 4.3 and testing the microcontroller current drain in section 4.7.
6. A multimeter and an oscilloscope can both be used for measuring voltages. What is the accuracy of each instrument? How do you decide which one to use?
7. What is a ×10 oscilloscope probe, and how is it used?
8. Describe the different triggering modes offered by the Oscilloscope [12].
9. An oscilloscope and a logic analyser can both be used for observing digital waveforms. How do you decide which one to use?
10. Which protocols are understood by the Saleae logic analyser.

2.3 Microcontrollers

Do some background reading into using the command line. At a minimum you should be comfortable with the following commands[3]: `cd, ls, mkdir, more`

1. What is a microcontroller?
2. What is the difference between volatile and non-volatile memory?
3. How much memory does the ATMEGA644P have for storing programs?
4. Describe the purpose of the fuses in the microcontroller.
5. Calculate the new values for the low and high fuses in section 4.8.3 and 4.8.4.
6. What is a boot loader?
7. When you have completed building your `Il Matto` why is it not possible to install the boot loader over the USB interface?
8. Which commands do you need to execute to compile the `C` source files into `hex` files suitable for downloading to the `Il Matto` board?
9. Which `AVRDUDE` command do you need to execute to download a program over the USB interface?
10. How many instructions per second does the microcontroller execute?

[3] replace `ls` with `dir` on Windows

3 Construction

In the lab you will build the `Il Matto` board. You will find further details about the circuit in appendix A. You will build a version with a mini-breadboard installed. A completed board is shown in figure 1.

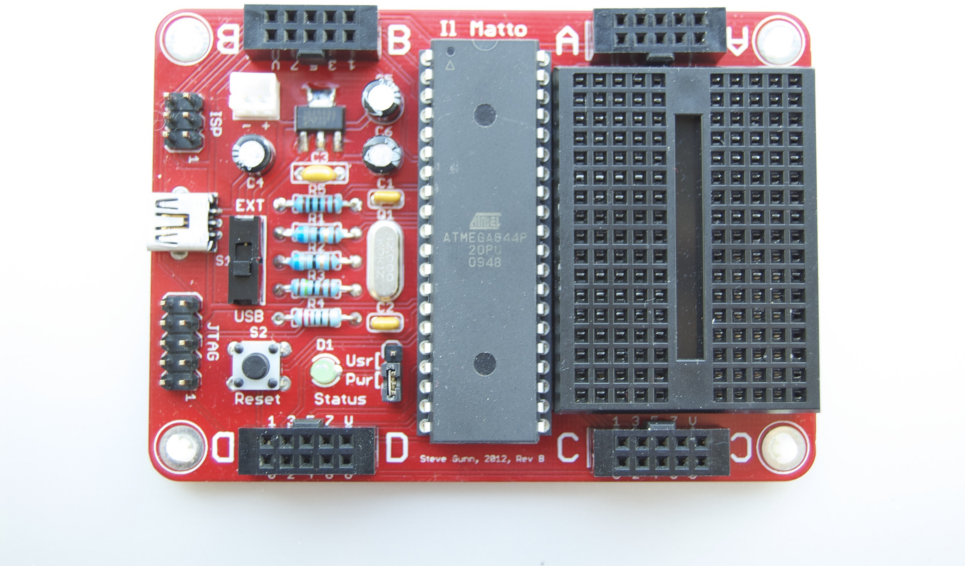


FIGURE 1: Finished `Il Matto`

3.1 Printed Circuit Board

Electronic circuits can be constructed using a number of technologies: breadboard, tri-pad, stripboard, wire-wrap, PCB, or dead-bug. For simple prototyping of new designs a breadboard can be a convenient method, but it is limited to low-speed circuits – as the frequency increases above 10MHz the parasitic capacitance between the contact strips can cause the design to fail. Furthermore, many modern components are only available in fine pitched surface mount devices making breadboarding difficult. Consequently, you will now find the PCB is often used in prototyping phases as well as for the construction of mature circuit designs.

A PCB is a multi-layer board, with each layer containing a number of copper tracks connecting the components as defined in the schematic. Simple boards are often two-layer with tracks on the top and bottom; boards designed for more demanding applications usually contain many more layers (4, 6 or 8 being common) with the additional layers being sandwiched inside and connections made with vias. The PCB you will use in this exercise is a double-sided board with two layers. The board has been designed to use the plated through holes (PTH) for the components to join the two layers together at appropriate points on the board; if necessary additional vias could have been added to connect a track from the top to bottom layer. PCB design is a challenging problem and it is good practice to reduce the number of vias as these increase the track impedance.

3.2 Components

Components are generally divided into two types: those that have wires or legs that are pushed through the board and soldered underneath (through hole) and those that lie on the surface of the board and are soldered on the same side (surface mount). Which type of package to use in a design depends upon a number of factors: component availability (modern devices are often only available as SMDs), assembly method (automatic pick-and-place machines may prefer SMDs), operating frequency (through hole component leads have higher parasitic inductance causing unwanted behaviour at higher frequencies), component density (SMDs can achieve a much higher density by their small size and by assembling them on both sides of the PCB), component alignment (a through hole component may align better) and mechanical strength (either type may have advantages). In practice often a design will be hybrid, containing at least one SMD or THC. The `Il Matto` is such a design, with `IC2` being a SMD.

The design contains a number of commonly used electronic components:

Resistor A device for limiting the flow of current. For through hole resistors the value is usually indicated by a set of colour codes. For SMDs the package is often too small to include any information. The device is not polarised and may be installed either way round. They can be fabricated using a range of technologies and they come in a range of tolerances, with 1% being the most common.

Capacitor A device for storing charge; often used for filtering and improving the local quality of power sources. The value and maximum voltage rating is normally written on the device. They can be polarised (typically values $\geq 1\mu$F) or un-polarised. They can be fabricated using a range of technologies and they come in a range of tolerances, with 10% being the most common.

Switch A device for selecting between a number of possible circuit configurations. There are a number of different variants – `S1`, `S2` and `JP3` are all switches. A common terminology is mPnT, where m describes the number of linked switches in the package and n describes the number of positions each switch can take, e.g. SPDT denotes a single pole double throw switch.

Crystal A device with a precise resonant frequency used for inducing timing in a circuit. The device is not polarised and may be installed either way round.

LED A polarised device which illuminates when current flows through it in the correct direction. Note that different colour LEDs have different voltage drops.

IC A multi-leaded package containing internal circuitry – often containing thousands or millions of components. Devices range from relatively simple voltage regulators with three pins to high performance microprocessors with many hundreds of pins.

Connector A device used for connecting to external circuitry. Often the devices come in male/female pairs, with one fitted to the PCB and the other used to attach a cable.

The full component list can be found in the bill of materials listed in table 1.

3.3 Tools

The basic tools for constructing circuits containing through hole and simple surface mount components are:

Soldering Iron Tool for heating the board pad and the component lead to the requisite temperature. Generally, you will want to work with a relatively small tip.

Solder Alloy which melts to form a good contact between the board pad and the component lead.

Desoldering tool Tool which is primed by pushing a plunger. A component joint can then be desoldered by heating the joint and releasing the plunger which sucks up the molten solder.

Solder wick A copper braid which is very effective at absorbing solder when heated.

Eye-glasses Safety protection for your eyes, if you do not wear prescription glasses.

Small-nose Pliers A tool for bending component leads.

Tweezers Tool for handling surface mount components.

Side-cutters Tool for removing excess wire from through hole components after soldering.

3.4 Techniques

The following sections introduce some of the basic techniques for soldering and de-soldering components. The important concepts to understand are that both pad and component lead must be raised to a sufficient temperature for good solder adhesion and that the flux in the solder is what makes the alloy flow nicely around the joint. The flux helps to remove any oxidation on the metal pads and component leads. Good joints will have a nice shiny finish to them.

> *Modern soldering irons heat up relatively quickly and will cause an unpleasant burn if brought into contact with your skin. Always leave the iron in its stand between the assembly of components, and when cooling down after use.*

Ensure the iron is placed in its stand and switch it on. The iron takes between 5 and 50 seconds to heat to its operating temperature - generally more expensive irons reaching operating temperature much quicker and have more flexible temperature adjustment. Before you start soldering, pick up the iron as you would a pencil and add a little solder to the tip. This tinning process helps to prevent oxidation of the tip and ensures that the tip lasts longer. It also confirms that the iron is up to operating temperature. If at any time you end up with too much solder on the tip you can remove it by wiping it on a wet sponge or using a modern brass sponge.

3.4.1 Soldering through hole components

1. Place the iron in good contact with **both** the pad and component lead.
2. Wait a couple of seconds for both pad and component to reach temperature.
3. Add a little solder to the joint, taking care not to add the solder to the iron tip, but aim it to meet the junction of the pad and the lead.
4. Add enough solder that the joint forms a volcano shape (figure 2A).
5. Remove the solder.
6. Remove the iron.
7. Ensure that the component lead is not moved while the solder cools (five seconds is enough) or the joint will fracture creating a dry joint.
8. Inspect the joint and ensure that it is shiny. If not reheat and add a little more solder to introduce some new flux.
9. Use the side-cutters to remove the excess lead, cupping your hand over the board as you do so to avoid the lead becoming a projectile.

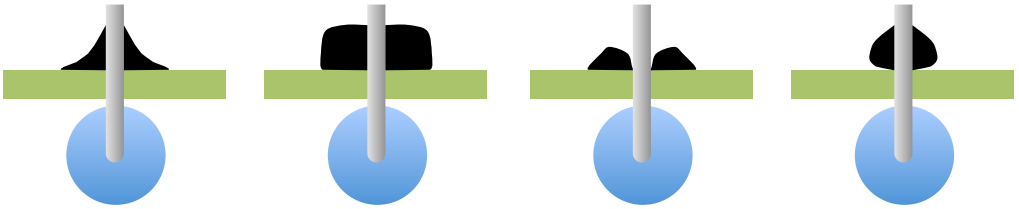


FIGURE 2: Examples of through hole Solder Joints (cross-section)
(A) Good, (B) Poor – excess solder, (C) Poor – no lead adhesion, (D) Poor – no pad adhesion

3.4.2 Soldering simple surface mount components

Flux is the magic ingredient which makes the solder flow smoothly around a joint. The solder you are provided with contains an internal flux core. However, when soldering SMDs often the solder is applied from the iron and as such the flux will have been burnt off. To counteract this it is possible to add flux directly to the board pads with a flux pen before starting to solder.

1. Shake the flux pen and depress the tip to initiate the flux flow. Wipe the tip over the SMD pads to be soldered.
2. Bring the iron to one pad and add some solder.
3. Pick up the component with the tweezers and move to the correct position.

4. Reheat the soldered pad and push down on the component so that it sits flush with the board.
5. Solder the remaining leads to their pads.
6. You may wish to go back and add a touch of solder to the first pad to add a little more flux.

3.4.3 Soldering components to the ground plane

When soldering components to the ground plane extra care must be taken as the plane will draw heat away from the iron. You will certainly need to wait a little longer before applying solder, but you may need to increase the power of the iron. Pay particular attention over these joints and look for cases where the joint looks like a ball of solder indicating that the ground plane did not reach a high enough temperature for good solder adhesion (figure 2D). Reheat offending joints and add a little more solder to add some new flux.

3.4.4 Removing a solder bridge

Modern PCBs contain a solder mask which helps to minimise the chances of an accidental bridge between two adjacent pads. However, when soldering fine pitch devices it may occur. Use a length of solder wick to absorb the excess solder, by touching the wick and iron over the offending area.

3.4.5 De-soldering components

To desolder components, the joint must be heated and either the solder removed, or the device is extracted whilst all joints are molten. Usually the former is simpler with the tools that you are using. There are two tools to aid in removal of solder: solder wick and a de-solder pump. When using solder wick, ensure that your iron temperature is high enough as the wick will draw heat away from the iron. Also ensure that you do not remove the iron before removing the wick from the board otherwise it will become stuck to the board. When using the de-solder pump ensure that it is placed as close as possible to the target. An alternative tactic when removing axial components, such as resistors, is to clamp the board and hold the lead of the resistor in a pair of pliers and heat from underneath, whilst pulling with the pliers.

3.5 Preparation

Firstly ensure you have a clear workbench so that when you are soldering components the PCB will be resting on a level surface. Ensure that your soldering iron sponge is not dry so that you are able to keep the soldering iron tip clean. Always wear the protective eye glasses provided when soldering to avoid possible solder splashes making contact with your eyes. After soldering always check your work for any dry joints or solder bridges.

Lay out the components from the `Il Matto` kit on a piece of paper and identify their values. Write down the component identifiers next to each component and ensure that none are missing. You can use your multimeter set to its resistance range to confirm the resistor values.

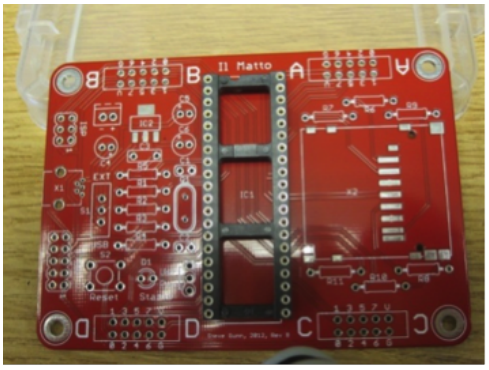
3.6 Assembly

There is not necessarily a best order in which to assemble the components, but some orderings are clearly inferior. A good procedure usually starts from the centre and works outwards, or from one edge working towards the other. This way there is unobstructed access for the soldering iron from one side, which becomes more important when working with SMDs. The following guide provides a suggested order of assembly. If you are experienced in construction and wish to follow your own procedure, do so, documenting your reasons in your logbook as preparation before the lab.

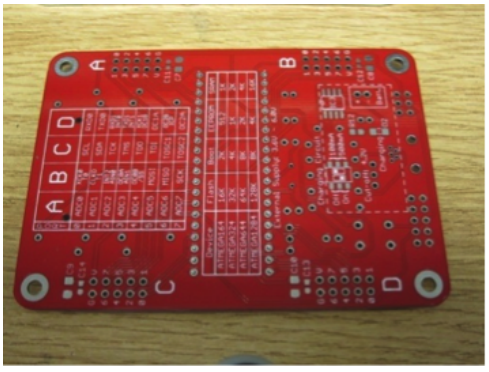
> *Take your time to carefully assemble the circuit. Desoldering components that are in the wrong place is more difficult than soldering. Remember, you need your circuit to work – other parts of your degree programme depend upon it.*

3.6.1 IC Socket

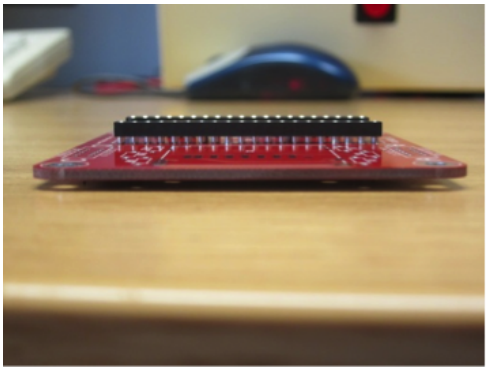
Insert the 40-pin DIL socket ensuring that the pin 1 indicator corresponds to the PCB indication.



Turn over the PCB so that it rests on the socket. Solder two diagonally opposite pins. Now turn the board over and confirm that the socket is flush with the board. If not, turn over and gently reheat the offending pin whilst applying a little downward pressure to the PCB.

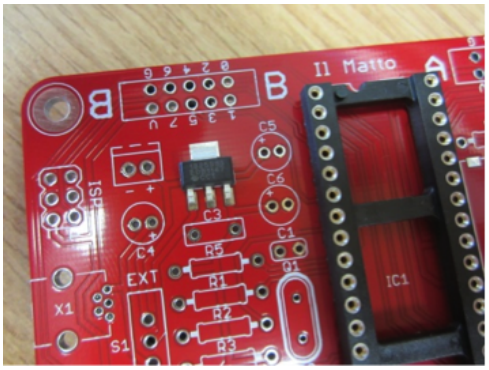


You can now solder the remaining 38 pins knowing that the socket is correctly inserted. If you have never soldered before, ask one of the demonstrators to take a look at your first few solder joints for feedback on your technique, before you proceed too far.



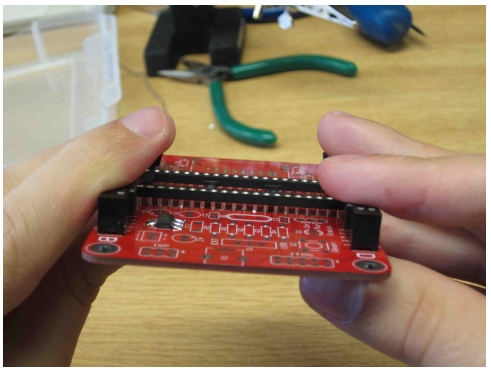
3.6.2 Surface mount component

First apply a little solder with the iron to one of the three small pads. Now using your tweezers pick up IC2 and bring it close to its correct position and reheat the pad with solder and push the IC into the correct position. Solder the remaining connections, remembering that you will need more heat to solder the heatsink tab as it is connected to the ground plane.

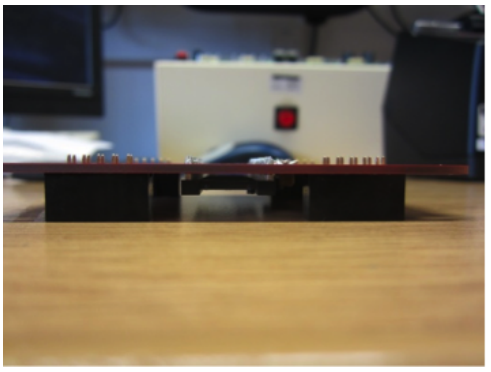


3.6.3 External port connectors

Insert the four port connectors into A, B, C and D with **the bumps facing inwards**. Using two hands place a finger on each of the connectors to hold them in place.

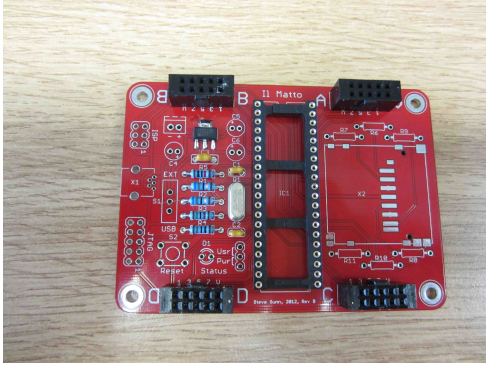


Then turn the PCB over so that it is resting on the four connectors. Solder the four connectors in place, by first soldering two diagonally opposite pins on each connector. Before soldering the remaining pins flip the board over and check that the connectors are aligned and flush with the board. Once satisfied, continue to solder the remaining pins.

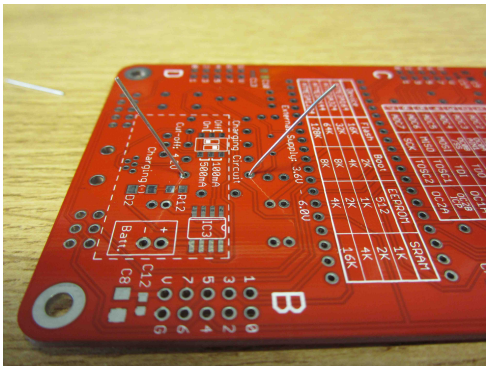


3.6.4 Non-polarised components

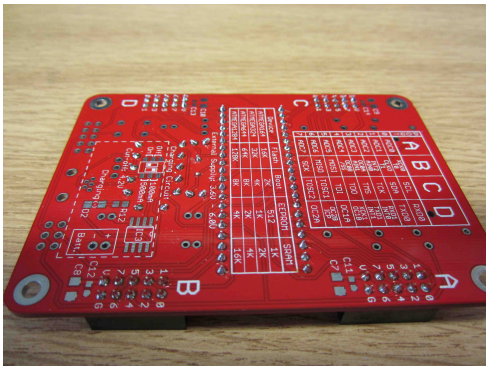
These two-leaded components have no polarity and can be inserted either way round. If you wish to be precise you can insert the resistors so that their colour codes read from left to right. A suggested order to fit the components is: Q1, C1, C2, R5, R1, R2, R3, R4 and C3.



Ensure that the components when fitting are not raised up from the PCB. Secure the components in place before soldering when required by bending the legs slightly.



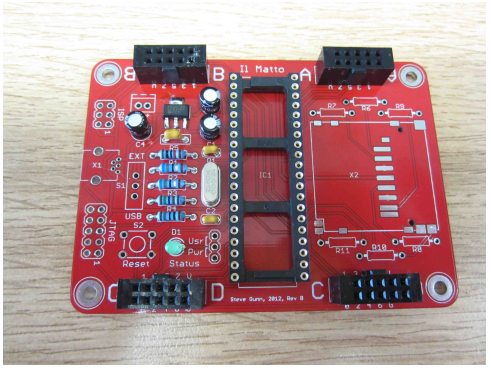
Remove surplus wire from the soldered components using the side-cutters. Hold, or place your hand over, the lead you are cutting to avoid it flying off as a projectile.



3.6.5 Polarised components

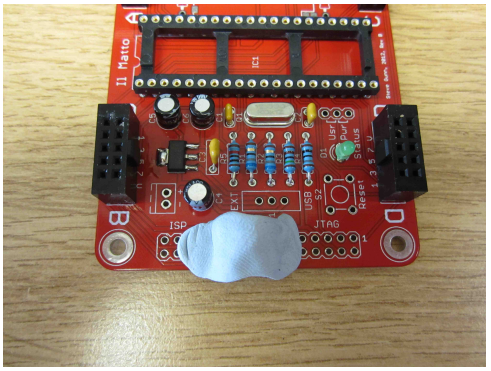
There are four polarised components which must be inserted with a particular orientation for the circuit to function correctly. The components are the electrolytic capacitors (C4, C5, C6) and the LED (D1). The electrolytics are usually marked with a minus sign on the case (also the shorter lead on a radial package). The shorter lead of the LED is the cathode, which also sometimes has a flat on the lens casing.

Solder these four components onto the PCB observing the correct polarity. The LED cathode is fitted adjacent to switch S2. Also take care to observe whilst all three electrolytics may look identical, C6 has a larger capacitance than the other capacitors.

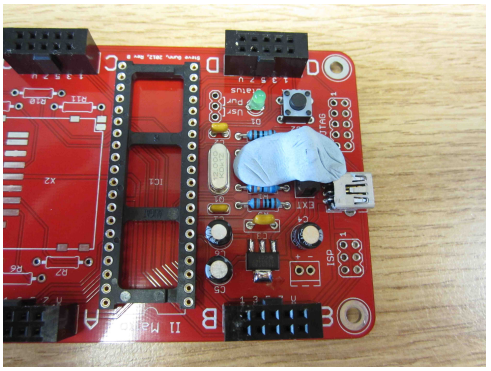


3.6.6 Switches and connectors

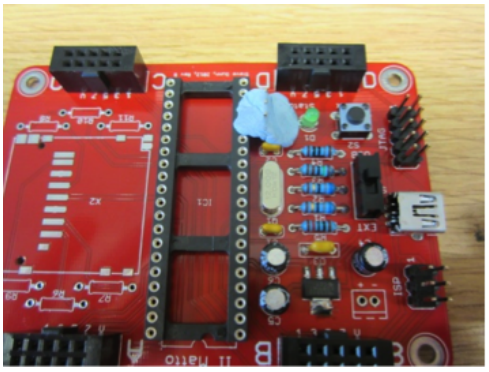
Use blue tac to hold in place the mini USB socket (X1). Start by soldering the two housing lugs. Then solder the five signal lines, taking care not to use too much solder as the pins are close together. Inspect the signal lines closely to ensure there are no shorts which could damage a USB port that the device is plugged into.



Using blue tac to hold it in place solder switch S1. Then solder S2 which is a snap-in type and should hold itself in place.



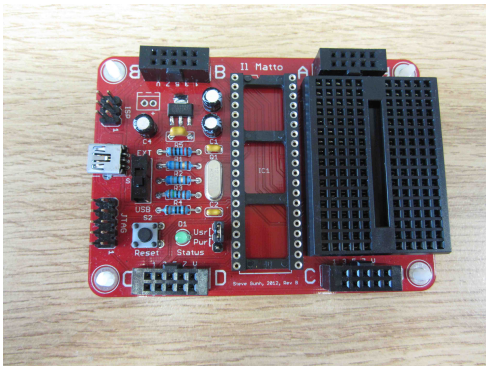
Using blue tac to hold them in place solder the 3-pin connector JP3, the 6-pin ISP connector JP1, the 10-pin JTAG connector JP2 and the external power connector X3. Take care to ensure X3 is inserted the correct way round by aligning it with the outline on the silkscreen.



3.6.7 Finishing Up

Attach the four plastic feet. Fit the mini-breadboard. You have now finished assembly of the Il Matto PCB.

Do not install IC1, or the jumper on JP3. You will undertake some testing before doing so.



4 Testing

At this stage it is very tempting to simply connect the power and turn the unit on. However, it is important to do some basic checks before applying power to any newly constructed piece of hardware to minimise any damage to faulty hardware. This section guides you through some standard test procedures for new hardware. It should be remembered that with the complexity of modern hardware it impossible to test for all possible scenarios and therefore tests must be designed to have good coverage of likely failure modes.

4.1 Visual inspection

Look closely at all of the solder joints and verify that they are to a good standard with no apparent dry joints. Also look for any solder splashes or excess solder which may unintentionally be bridging two adjacent parts of the circuit. If any joints do not look shiny you can reheat them and add a little more solder to improve them. If you have solder splashes or too much solder you can remove these with solder wick or a desolder pump.

4.2 Power input

Set `S1` to the `EXT` position to determine that power will be supplied from the external power connector `X3`. Using a multimeter set to its resistance mode, test the resistance between the pins on the external power connector `X3`. Repeat the test with the multimeter leads transposed and comment on any difference. What range would you expect the value to be in? If the value is too small or too large return to section 4.1 and use the multimeter to try to identify the problem.

4.3 Voltage regulator

> *When testing new hardware for the first time always try to use a bench PSU with an adjustable current limit to prevent too much current being drawn. Excess current can permanently damage devices. By limiting the current it is often possible to identify and rectify a problem without damage.*

The first powered test you will perform is to apply power to the external power connector (`X3`) and verify that the voltage regulator section is working correctly. The MCP1825 voltage regulator accepts an input voltage up to an absolute maximum of 6.0V and produces a regulated output of 3.3V at up to 500mA. It is a low dropout regulator (LDO) which means it can operate with an input voltage only slightly higher than the 3.3V output. The device is a *linear* regulator which means that that it behaves like a variable resistor, continually adjusting its resistance to produce the regulated output voltage (V_O) from the unregulated input voltage (V_I). The voltage drop is dissipated as waste heat,

$$P = (V_I - V_O)\,I$$

where I is the supply current. To help with heat dissipation the regulator has a ground tab which is soldered to the ground plane. The copper of the ground plane helps to diffuse the heat away from the device.

Based on your preparation set the voltage and current limit on the PSU to a suitable value and connect the PSU to the external power connector (X3). Turn on the power. If the current limit light on the PSU illuminates, turn off the power and check that the limit is set to a sensible value. If it is, then return to section 4.1 and use the multimeter to try to identify the problem. Measure the voltage at the output of the voltage regulator (IC2) and confirm that it is regulated correctly at 3.3V. Gradually reduce the PSU voltage and determine the absolute minimum input voltage for the correct 3.3V regulated output.

4.4 LED indicator

Put the jumper on JP3 in the Pwr position to specify that the LED should operate as a power indicator and verify that LED illuminates correctly. Use your multimeter to measure the voltage drop across the LED and confirm whether this matches your expectation. Perform a second measurement with the multimeter to measure the current flowing through the LED. If the LED does not illuminate make additional measurements with your multimeter to try to determine the problem: Is the LED in the wrong way round? Is there a dry solder joint? Try to use the multimeter to help build up evidence to identify problems, rather than randomly changing things.

> *Fault finding skills for hardware are similar to debugging skills for software. Try to remove as many assumptions as possible before beginning – do not assume anything works including the test equipment that you are using. Break the problem down in a modular way and methodically try to verify correct operation of each part, by developing simple tests which you can undertake with the test equipment. A common problem in electronic circuits is no power, or a poorly decoupled supply.*

4.5 Power distribution

Set your multimeter to its voltage range and make a test to determine whether IC1 will be powered correctly when the device is later inserted. Verify also that power (V) and ground (G) are supplied correctly to all four external ports on the PCB headers SV1–4, and also to the appropriate pins on JP1 and JP2.

> *Think carefully about how you will measure whether a point is at ground potential. Hint: What voltage does your multimeter measure when one of the leads is disconnected from the circuit?*

4.6 Reset switch

Connect your oscilloscope to the AVR reset signal using your ×10 probe[4]. This signal is easiest to access on either JP1 or JP2. Configure your oscilloscope to trigger on a falling edge, and set the capture to single shot. Press the reset button (S2) and confirm that you are able to see the signal transition from high-low (⎍). Experiment with setting the timebase and trigger delay controls to get a good close-up of the transition, including the pre- and post-edge behaviour, and sketch the resulting waveform in your logbook. If you are unable to observe the transition try to use the multimeter or the oscilloscope to find the problem. Repeat the experiment with the oscilloscope set to trigger on a rising edge and try to capture the full high-low-high (⎍) transition as the button is briefly pressed.

4.7 Microcontroller current

You will now insert the microcontroller into its socket. The socket on the board is a *turned pin* socket which provides greater reliability than standard IC sockets. However, this also means you will need greater force to insert the IC into the socket.

> *Always turn off the power before inserting or removing an integrated circuit from its socket, and avoid touching the pins as the device can be damaged by electrostatic discharge (ESD). When inserting, care must be taken to ensure that the pins are lined up before pressing down firmly on the IC. Support the board underneath the IC whilst doing this so as not to overly flex the PCB.*

Insert IC1 by carefully resting the IC over the socket and confirming that the pins align. If the pins do not line up with the socket, hold the IC firmly at each end and gently push the pins against the desk to bend them into alignment. Do this a little and then re-check for correct alignment. When the pins align, double check that IC1 is in the correct orientation and push it firmly into its socket.

Power on, and confirm that the current drain is in line with your expectations. If not, return to section 4.1 and use the multimeter to try to identify the problem. Connect your oscilloscope to the VCC rail, select DC coupling and confirm that the supply is at the correct voltage. Switch to AC coupling and increase the vertical sensitivity. Do you observe any deviation from the ideal regulator output?

Power off, and disconnect the external power connector from X3. Now that you have verified that the power drain is in line with expectations you will power the device from alternative sources.

4.8 ISP interface

In this section you will use a programming cable to talk to your new hardware over the ISP interface (JP1). You will use the AVRDUDE software running on the host PC to perform the download of a program to the flash memory and later to configure the fuses of the microcontroller. Finally, you will install a boot loader program to enable the USB interface on X1.

[4] Remember to always calibrate your ×10 oscilloscope probe before use.

4.8.1 Connecting the programming cable

To communicate with your newly constructed embedded platform you will use the FTDI C232HM cable (see appendix E). This is a versatile cable and you will be using it first to communicate with the ISP port of your microcontroller to enable you to download programs and to configure the fuses. This cable is able to supply power to the board as well as communicating with it. Connect the cable according to the diagram in figure 3 and double check your connections. Then plug in the other

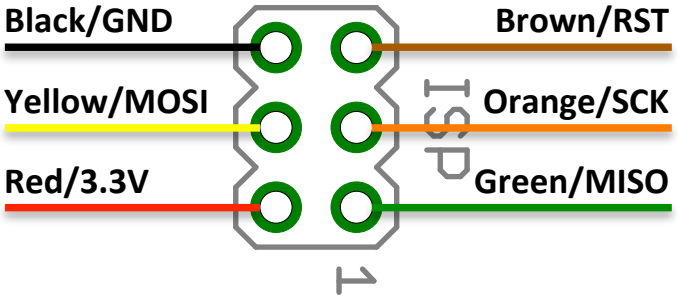


FIGURE 3: C232HM ISP connection

end of the cable to a USB port on your host PC. Open up a terminal window and type the following command at the command prompt.

```
avrdude -c c232hm -p m644p
```

The command tells the AVRDUDE software to use the C232HM programmer and tells it that it will be connecting to an ATMEGA644P device. If your circuit has been constructed successfully and your wiring is correct you should see the following message:

```
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.02s

avrdude: Device signature = 0x1e960a

avrdude: safemode: Fuses OK

avrdude done.  Thank you.
```

The device signature is a unique set of three bytes (24 bits) denoting the particular AVR microcontroller. If you see 0x1e960a it means you have successfully communicated with your new hardware and AVRDUDE has detected the microcontroller correctly. If not, you should use your test equipment to observe the ISP header signals and confirm that these are successfully reaching the correct pins of the microcontroller.

4.8.2 Downloading your first program

Ensure that JP3 is in the Usr position. Build program 1 from the source file LEDTest.c and then use AVRDUDE to download the LEDTest.hex program to your embedded target and confirm whether the LED flashes.

```
avrdude -c c232hm -p m644p -U flash:w:LEDTest.hex
```

```
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.02s

avrdude: Device signature = 0x1e960a
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "LEDTest.hex"
avrdude: input file LEDTest.hex auto detected as Intel Hex
avrdude: writing flash (234 bytes):

Writing | ################################################## | 100% 0.08s

avrdude: 234 bytes of flash written
avrdude: verifying flash memory against LEDTest.hex:
avrdude: load data flash data from input file LEDTest.hex:
avrdude: input file LEDTest.hex auto detected as Intel Hex
avrdude: input file LEDTest.hex contains 234 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.56s

avrdude: verifying ...
avrdude: 234 bytes of flash verified

avrdude: safemode: Fuses OK

avrdude done.  Thank you.
```

Connect your oscilloscope to pin PB7 using your ×10 probe and sketch the waveform in your logbook, making a measurement of the voltage, period and frequency of the signal.

Download the program again and watch the LED as the program is downloaded. What happens and why?

4.8.3 Configuring the fuses

The fuses in the microcontroller are used to configure it and can be considered as programmable switches. Fuses are non-volatile and hence they remember their state even when power is removed

from the microcontroller. There are three fuses in the ATMEGA644P with each fuse containing eight bits. These bits control the clock source (low fuse), whether parts of the device are enabled (high fuse) and what should happen if the power supply level drops (extended fuse). Details are given in table 3 including the default settings for a new AVR (the AVR supplied in your kit is a new device).

Use AVRDUDE to read the current value of the fuses on your board and confirm that the fuse values match the default settings in table 3. The following command will read all three fuses and store the values of each one in a separate text file in hexademical format.

```
avrdude -c c232hm -p m644p -U lfuse:r:lf.txt:h -U hfuse:r:hf.txt:h -U efuse:r:ef.txt:h
```

Most of the default values configure the AVR to a suitable state. However, by default the clock source is selected to use the low precision internal RC oscillator. Your board has a high precision external crystal oscillator and you will need to change to this source to fix the timing for program 1 and to be able to use the USB interface described in section 4.9. Additionally you will also need to disable the "Divide Clock by 8" fuse.

You can use AVRDUDE to modify the low fuse to make the required changes, using the following command; replace ?? with the correct 8-bit value for the low fuse in hexadecimal format (each hex digit corresponds to 4-bits).

```
avrdude -c c232hm -p m644p -U lfuse:w:0x??:m
```

Repeat the observation of pin PB7 using the oscillooscope and confirm that the timing is now correct for program 1.

4.8.4 Installing a boot loader

By installing the boot loader described in appendix D it is possible to enable the USB interface on X1. First you will need to configure the fuses to set the boot loader section to the correct size of 2048 bytes (1024 words) and enable the boot reset vector. These settings are both configured in the least significant bits of the high fuse.

> *Take care when changing the high fuse as it is possible to disable the ISP (SPIEN bit), the JTAG (JTAGEN bit) and the USB (BOOTSRST bit) interfaces making it impossible to communicate with the microcontroller using any of these interfaces.*

You can use the following command to configure the boot loader bits of the high fuse, replacing ? with the correct hexadecimal digit.

```
avrdude -c c232hm -p m644p -U hfuse:w:0x9?:m
```

The boot loader program has been pre-compiled for you and can be downloaded to the target with the following command.

```
avrdude -c c232hm -p m644p -U flash:w:atmega644pa-12mhz_2048.hex
```

You will confirm correct operation of the boot loader in the following section 4.9

4.9 USB interface

In this section you will use the USB interface to talk to your new hardware. You will use the newly installed boot loader in combination with the AVRDUDE software to perform the download of a program to the flash memory of the microcontroller – it is not possible to configure the fuses using this interface. This USB interface is the connection that you will normally use to program your microcontroller, but should your boot loader section become corrupted, or should you need to change the fuses, you can always revert to section 4.8.

4.9.1 Connecting the programming cable

Firstly, make sure to disconnect the C232HM cable as this is no longer required for programming and power will now be supplied from the USB port of the host PC. Trying to power the device from both sources could damage the C232HM cable. Set JP3 to the Pwr position and connect the supplied USB cable from X1 to any USB port on the your PC[5]. Set S1 to the USB position to provide power to the board from the USB host and confirm that the LED illuminates.

4.9.2 Entering the boot loader

Set JP3 back to the Usr position. Press S2 and the LED should illuminate indicating that the microcontroller has entered the boot loader code and is ready to communicate with AVRDUDE[6]. If this does not happen you should return to section 4.8.4 and re-check the installation of the boot loader. Try uploading the boot loader and verifying it and also reading the fuses and confirming that they are set correctly.

It is now possible to execute an AVRDUDE command and after execution the device attempts to start the program in the application section of the flash memory. Press S2 again if you wish to re-enter the boot loader, or consider the *terminal mode* of AVRDUDE if you wish to enter multiple commands (see appendix D).

> *Remember, when the boot loader is installed and enabled S2 is re-tasked to act as an "Enter Boot Loader" rather than a "Reset" switch. The microcontroller can be reset by removing and re-applying the power, which can be accomplished with S1.*

[5] In the lab it is best to use the four-port hub at the back of the bench.
[6] On windows the device requires the libusb driver to be installed [2].

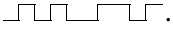
4.9.3 Downloading your second program

Build program 2 from the source file `HelloWorld.c` and then use `AVRDUDE` to download the program to your embedded target, this time using the USB interface.

```
avrdude -c usbasp -p m644p -U flash:w:HelloWorld.hex
```

Verify that the program generates the correct Morse code by first observing the LED and then capturing the full message on the oscilloscope. Find a method to print the oscilloscope trace and paste it into your logbook.

4.10 UART interface

In this section you will test the first of the two UART interfaces on the board. To do this you will download and execute program 3 which receives serial communication from the PC host and sends back a response. This asynchronous interface uses two lines, one for transmitting and one for receiving data. The data is encoded in a binary serial format using one start bit (always low), 8 data bits and 1 stop bit (always high), e.g. the ASCII character 'e' (`0x65`) is transmitted as  . You will use the versatile C232HM cable again, but this time configured to operate as a USB to UART interface, enabling your embedded device to communicate with the host PC. On the host PC you will use a terminal program to send and display received characters.

4.10.1 Connecting the UART cable

Both of the UART interfaces in the ATMEGA644P are located on port `D`. You will need to connect three wires from the C232HM cable (TX, RX and GND) to make the connection to the first interface on port `D`. You can use single strand hook-up wire to make the connection. Make sure to swap over the TX and RX lines from the cable to the board, so that the cable TX line goes to the RX line on the board and vice-versa. Why is the GND connection required?

4.10.2 Downloading your third program

Build program 3 from the source file `UARTTest.c` and use `AVRDUDE` to download the `UARTTest.hex` program to your embedded target using the USB interface.

Start up a terminal on the host PC. On a Linux or Mac machine you can use the command

```
screen /dev/tty.usbserial??????? 9600
```

On a Windows machine you can use `hyperterminal` or `PuTTY` to make the connection to the correct COM port (usually the highest number in the list - you can check in device manager by unplugging the C232HM cable and seeing which COM port disappears). Select 9600 for the baud rate, no flow control, no parity and one stop bit.

Try typing keys and you should see both the red and green LEDs flash on the C232HM cable indicating transmission and reception of data respectively. You should observe something like

```
You sent the character 'x'
```

being received by the host PC terminal. Verify that you can observe the data transmission by using the oscilloscope to monitor the TX and RX lines. Use the oscilloscope to measure the time delay between the end of the transmission from the host and the beginning of the transmission from `Il Matto` when a key is pressed.

> *Note that the C232HM cable will provide enough power over the TX line to power the microcontroller, preventing `S1` being able to fully power cycle and reset the device. In order to restart the program it is possible to reset the device by pressing `S2` and exiting the boot loader by executing `avrdude -c usbasp -p m644p`*

4.10.3 Using a logic analyser

One of the challenges when working with modern digital circuits is that often you will want to observe the data that is being transmitted from the point of view of the transmission protocol, rather than the raw voltage levels. For example, we know that the ASCII character 'e' (`0x65`) will be transmitted as ⎍⎍⎍⎍⎍⎍⎍⎍ over our serial connection. Observing this signal on the oscilloscope is not too bad for a single character, but usually data is transmitted in large streams and trying to decipher where the start of characters are and mapping the bits to the corresponding character is tiresome. This is where logic analysers that understand the communication protocol can be a big help.

You will now use the Saleae Logic, which is an 8-channel logic analyser that understands a number of commonly used communication protocols, to observe the data on the TX and RX lines:

Connection Attach the first two channels of the Logic to the TX and RX lines, and connect the ninth GND line to the board.

Protocol Configuration Start the Logic software and configure the first two channels to Async Serial with the correct baud rate.

Sampling Rate Reduce the sampling rate from 24MHz to 1MHz (as you are working with a relatively slow link).

Triggering Set the analyser to trigger on a rising edge of the TX signal.

Acquisition Hit 'Start' and go back to your terminal window and type a character. You should now see something like figure 4 in the Logic software.

Use the logic analyser to confirm the time delay measurement between the end of the transmission from the host and the beginning of the transmission from `Il Matto` when a key is pressed.

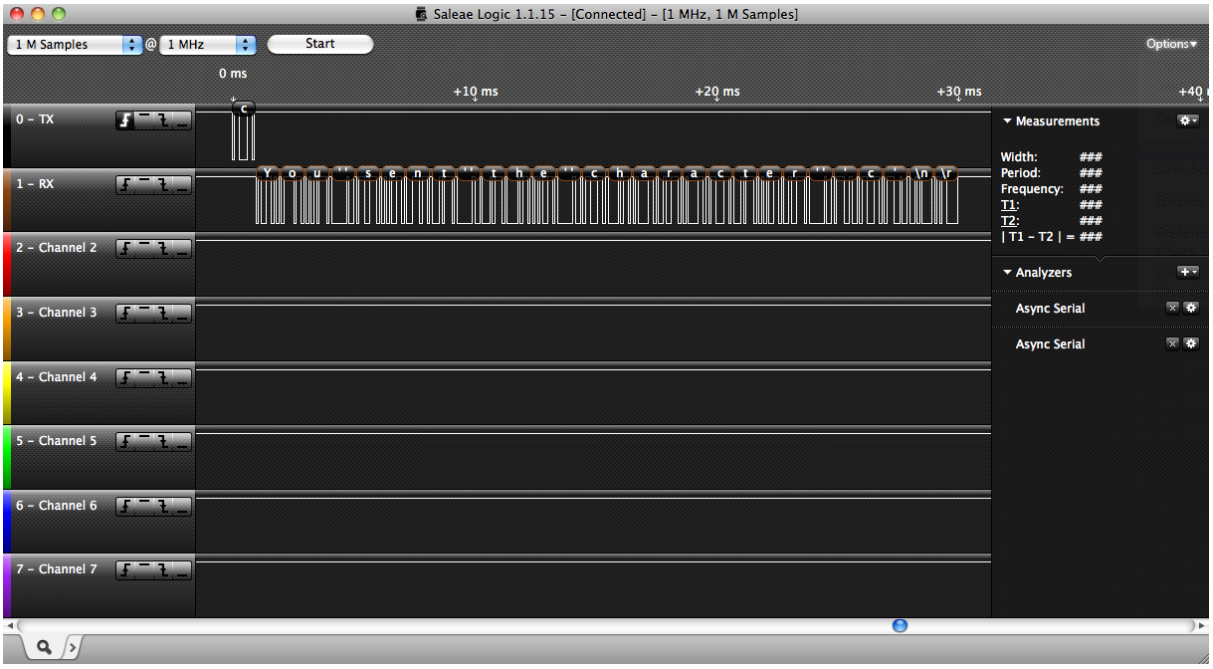


FIGURE 4: Protocol Analyser view of the UART communication

4.11 GPIO interface

In the following section you will test the GPIO interface on your board, to confirm that all pins can function correctly as inputs and outputs. The test program should identify a short circuit (e.g. a solder splash between adjacent pins) and an open circuit (e.g. a dry solder joint) condition.

4.11.1 Downloading your fourth program

Ensure that JP3 is in the `Usr` position. Build program 4 from the source file `GPIOTest.c` and then use `AVRDUDE` to download the `GPIOTest.hex` program to your embedded target. The LED should then extinguish and after approximately 8 seconds[7] it should come back on, indicating that the program has completed and entered the boot loader. The test results can be downloaded with the command

```
avrdude -c usbasp -p m644p -U eeprom:r:eeprom.txt:r
```

and the results can be viewed by typing

```
more eeprom.txt
```

4.11.2 Investigating the program behaviour

Connect your logic analyser to the 8-bits of port `A` and configure it to trigger on a falling edge on `PA7`. Re-run the program by power cycling the device with `S1` and explain what you observe.

[7] The runtime is largely dictated by the EEPROM size and will vary for devices other than the ATMEGA644P.

4.11.3 Testing the ports

Take 16 lengths of hook-up wire (approximately 10cm long) and use them to connect together the corresponding bits of ports A↔C and B↔D. You have now set the device for a loopback test and the test program will cycle through configuring one port as an output, writing signals, and attempting to read these signals on the other ports. Re-run the test program and confirm that your Il Matto board is functioning correctly by uploading the test results from the EEPROM and analysing the results. Discuss the results in your logbook.

4.12 JTAG interface

In this final section you will test the JTAG port. The JTAG port is the most capable of the programming interfaces and is widely adopted on many digital ICs, including microcontrollers, CPLDs, FPGAs and more complex processors. It is a four wire serial protocol and has the advantage that many different devices can be chained together, and all of them can be programmed with a single JTAG interface connector on a board. Some of the features of the JTAG interface include device programming (read-/write the state of memory and fuses), boundary scan (the state of all pins can be read) and on-chip debugging (programs can be debugged running on the hardware, by setting breakpoints and stepping through instructions).

4.12.1 Connecting the programming cable

To communicate with your board using JTAG you will use the FTDI C232HM cable (see appendix E) again. As with the ISP interface the cable is able to supply power to the board as well as communicating with it, and hence the USB cable should be disconnected. Connect the cable according to the diagram in figure 5 and double check your connections. Then plug in the other end of the cable to a USB port on your host PC. You can now use the UrJTAG program to communicate with the microcontroller over the JTAG interface.

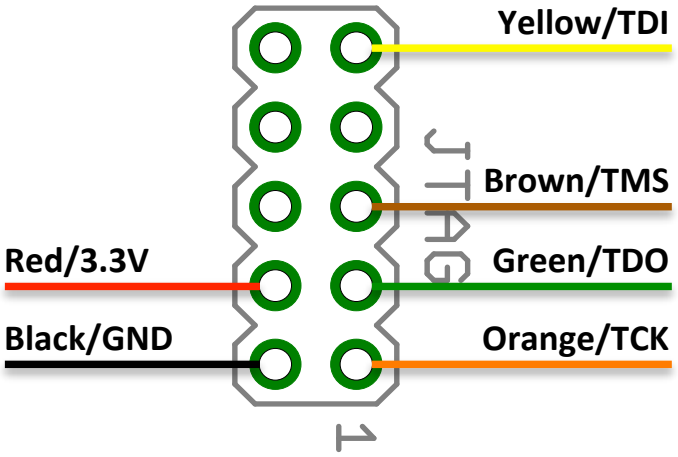


FIGURE 5: c232hm JTAG connection

4.12.2 Communicating with UrJTAG

Start up UrJTAG and you should be presented with a jtag> prompt. To begin you must instruct UrJTAG which programming cable you are using.

```
jtag> cable FT2232
```

Now you can instruct UrJTAG to detect which devices are connected to the chain.

```
jtag> detect
```

In this example the command should return information about the single AVR connected to the JTAG port, but on a more complex board you might see a long list of devices all connected in a chain. Confirm that you are able to detect the device correctly and record the response in your logbook.

The JTAG interface can be sent a number of high level commands that are stored in an SVF (Serial Vector Format) file. Execute the commands stored in the mystery.svf file.

```
jtag> svf mystery.svf
```

The scan command shows you which pins have changed state since the command was last executed.

```
jtag> scan
```

Using the scan command a number of times and by observing the circuit behaviour try to deduce the action of the commands in the mystery.svf file.

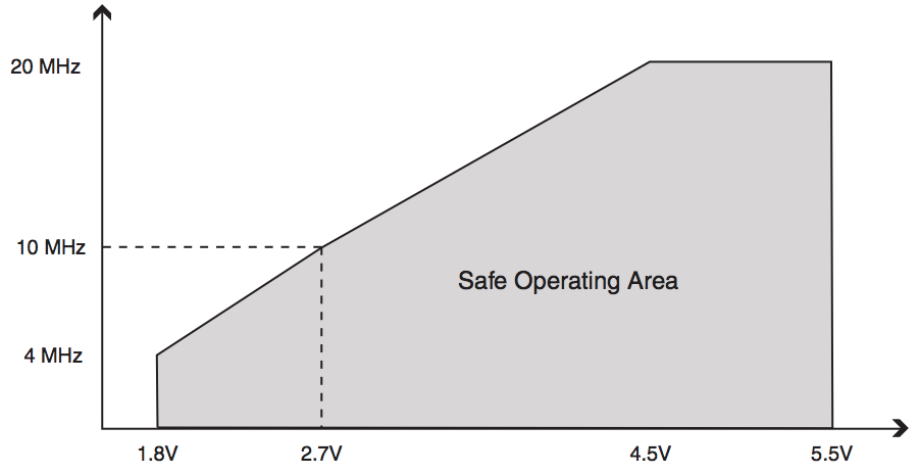
Appendix A Il Matto Development Board

`Il Matto` has been designed as a low-cost general purpose breakout board for an Atmel AVR microcontroller. The following section provides a brief design overview along with the full schematic (figure 6), the component layout (figure 7) and the bill of materials (table 1).

Device Support The board supports a family of four AVRs that offer the same pinout and peripherals, but differ in the amount of available memory. The compatible devices are listed in table 2 and for convenience also on the underside of the PCB (figure 7B). This family was chosen because it offers plenty of GPIO (4 × 8-bit ports), common peripherals, including an ADC, and a JTAG port. Additionally, the device is available in a number of packages including a 40-pin DIL package; whilst this package is relatively large it has the advantage that the device can be socketed and easily replaced should it be damaged by over-voltage or over-current on any of its pins.

Voltage Range The devices operate from 1.8-5.5V, but the `Il Matto` is configured to operate at 3.3V through the use of its on-board voltage regulator, which can accept up to a 6.0V external input. 3.3V is convenient since it is compatible with many modern devices and simplifies interfacing. Furthermore, by the choice of an LDO regulator it is possible to operate the device from a single LiPo battery (3.7V).

Frequency Range The device can operate at up to 20MHz at 5.0V, but when operating at the reduced 3.3V power supply the maximum operating frequency is 13.3MHz.



(Figure reproduced from the ATMEGA644P datasheet [5])

Programming Interfaces The device can be programmed using either the ISP or JTAG interfaces brought out to connectors `JP1` or `JP2` respectively. Additionally, by installing a boot loader program into the device and correctly configuring its fuses, it is also possible to program the device using the USB interface on `X1`.

Port Connectors The connectors have been chosen to enable three alternate modes of connection to external devices: a) by pushing single stranded wire directly into the sockets; b) by ribbon cable and a 10-way IDC header; c) a second circuit board with 10-way headers can sit on top.

Name Il Matto is Italian and means *the Fool*, *the Joker*, or *the Madman*. It is the first of twenty-two trump cards (Major Arcana) in the Tarot deck.

Dimensions The board is the size of a standard playing card (3.5" × 2.5").

Port IO All four ports may be configured as GPIO. Alternatively, each pin may also be configured to take on a special function. The special functions available at each pin are listed in table 4 and for convenience also on the underside of the PCB (figure 7B).

> *An important consideration is deciding on which pins to use for connecting external circuitry. If a special function is required then often the pin is dictated, but for GPIO care should be exercised when using Ports B or C since they may interact with ISP, USB, JTAG or SD card connections.*

Options A small number of additional options are provided to enable the board to be customised for various activities. These options are shown in dotted boxes in the schematic (figure 6):

SD Card The right hand side area of the PCB can accept an SD card socket which enables the AVR to read and write data to an SD card using the SPI interface. The Card Detect and Write Protect switches are connected to PB1 and PB0 respectively. Since PB7 is connected to SCK, the LED (D1) can be used as an SD activity light when the jumper JP3 is in the Usr position.

Mini-Breadboard If the SD card is not installed it is possible to fix a mini-breadboard in that area of the PCB to enable small circuits to be prototyped directly on the development board.

Port Decoupling On the bottom of the PCB eight surface mount capacitors (C7-14) can be mounted to provide additional decoupling of the power rails supplied to the four port connectors.

LiPo Charger The bottom of the PCB also contains an area where a LiPo charger circuit can be mounted, consisting of a controller IC (IC3), a charging current selection switch (S3a), a charging enable switch (S3b), and a charging indicator (R12 and D2). When a LiPo battery is connected to X3, the battery will be charged if the device is plugged into a USB port. The charging circuitry automatically disables itself when the battery voltage reaches 4.2V.

> *LiPo batteries can be dangerous if mis-handled. Always ensure that you follow the standard procedures when using them; summarised, avoid exposing them to physical shock, heat or short circuit.*

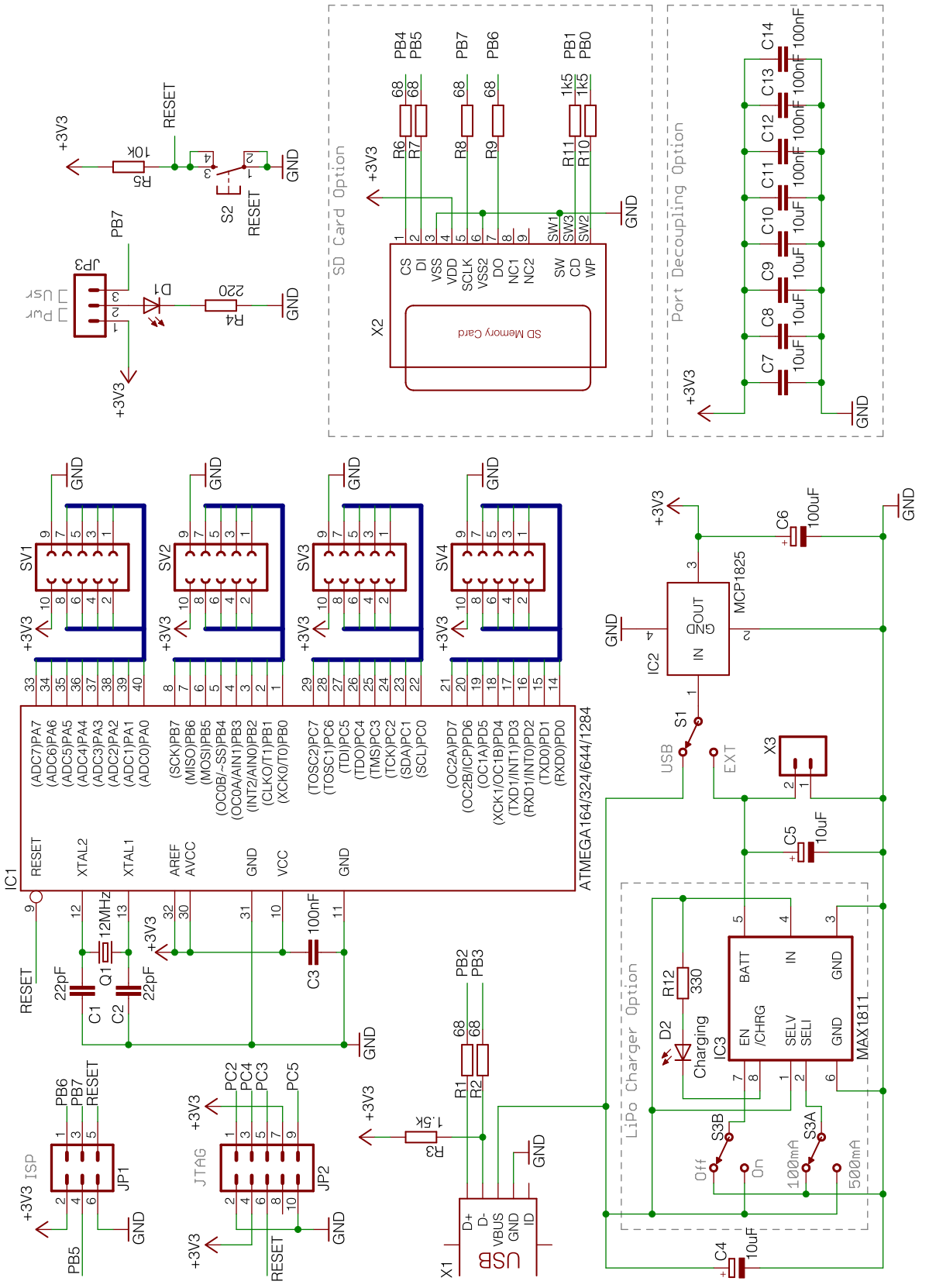
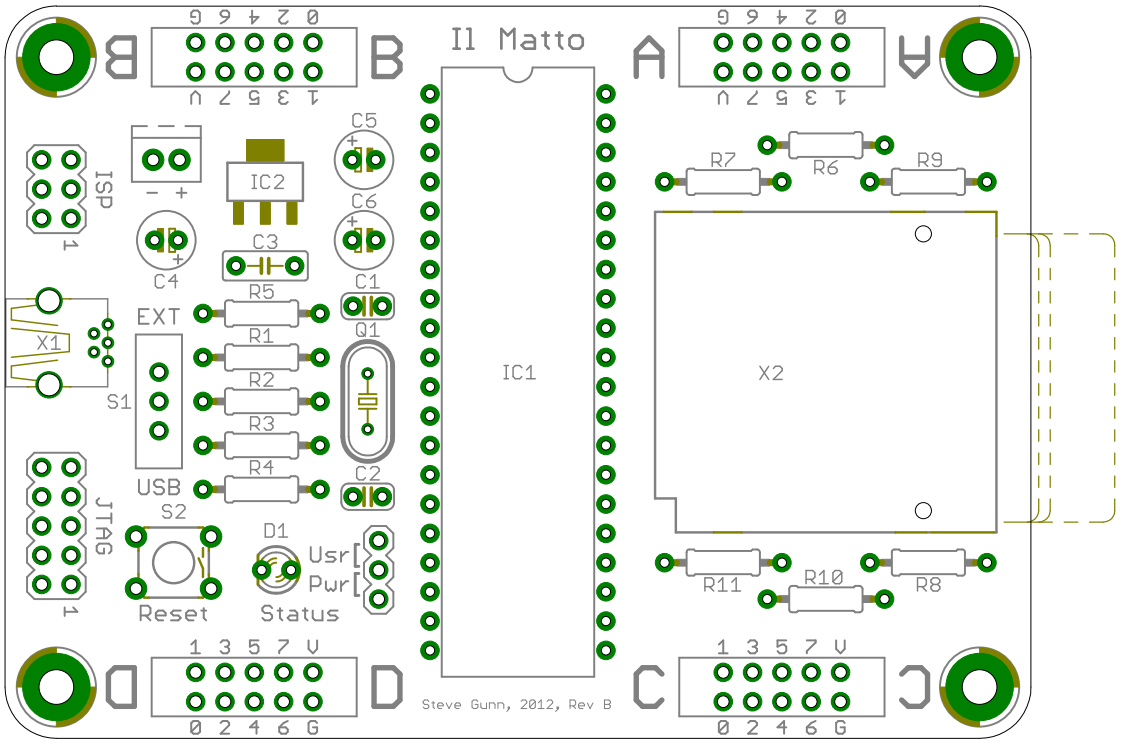
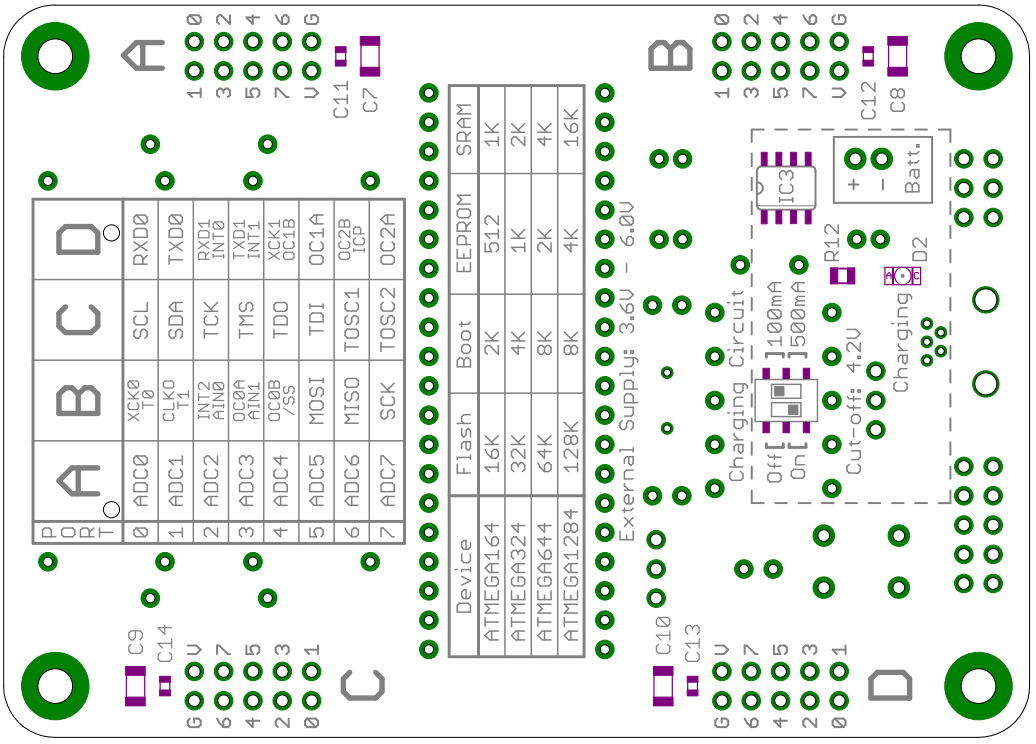


FIGURE 6: Il Matto Schematic



(A) Top Component Layout



(B) Bottom Component Layout

FIGURE 7: Il Matto board layout

TABLE 1: Il Matto Bill of Materials

Qty	Description	Identifier	Supplier	Supplier Part no.
2	68Ω resistor	R1, R2	Onecall	9342192
1	1.5kΩ resistor	R3	Onecall	9341323
1	180Ω resistor	R4	Onecall	9341420
1	10kΩ resistor	R5	Onecall	9341110
2	22pF capacitor	C1, C2	Onecall	1216407
1	100nF capacitor	C3	Onecall	1216440
2	10μF electrolytic capacitor	C4, C5	Onecall	9452281
1	100μF electrolytic capacitor	C6	Onecall	9452176
1	Green LED	D1	Onecall	1142502
1	12MHz crystal oscillator	Q1	Onecall	SC08194
1	ATMEGA644P IC	IC1	Onecall	1455122
1	MCP1825 IC	IC2	Onecall	1578404
1	SPDT switch	S1	DigiKey	EG1903-ND
1	Push button switch	S2	Onecall	1555982
1	Mini-B USB socket	X1	Onecall	9786465
1	6-pin header (ISP)	JP1	Toby	THD-03-R
1	10-pin header (JTAG)	JP2	Toby	THD-05-R
1	3-pin header	JP3	Toby	THS-03-R
1	Jumper for JP3		Toby	TSL-260-R-O
1	40-pin DIL socket		Toby	POS-640-S001-95
4	10-pin socket	SV1-4	Toby	B06d-10-AGAA1-G
1	Il Matto PCB		PCBCART	
1	2-pin JST connector	X3	Onecall	9491856
4	Mounting feet		Toby	DCB-4
1	Plastic box		RUP	0.2 litre
1	USB cable (Male-A to Male-Mini-B)		Onecall	1651027
4	68Ω resistor	R6-9	Onecall	9342192
2	1.5kΩ resistor	R10, R11	Onecall	9341323
1	SD card socket	X2	Onecall	9186174
1	Mini-breadboard		Sparkfun	PRT-08803
4	10μF	C7-10	Onecall	1754086
4	100nF	C11-14	Onecall	1414664
1	330Ω resistor	R12	Onecall	1576452
1	Orange LED	D2	Onecall	SC08097
1	MAX1811 IC	IC3	Onecall	1593327
1	DPDT switch	S3	DigiKey	CASD20GCT-ND
1	3.7V LiPo battery (900mAh)		Sparkfun	PRT-00341

Notes: The optional parts not included in the kit for X2 are shown with a grey background.

Appendix B AVR 8-bit Microcontroller

TABLE 2: AVR Memory for ATMEGA164/324/644/1284 family

Device	Flash	Boot	EEPROM	SRAM
ATmega164	16K	2K	512	1K
ATmega324	32K	4K	1K	2K
ATmega644	64K	8K	2K	4K
ATmega1284	128K	8K	4K	16K

TABLE 3: AVR Fuses for ATMEGA164/324/644/1284 family

Fuse	Bit	Default	Name	Description
Low	7	☒	CKDIV8	Divide clock by 8
	6	☐	CKOUT	Enable clock output on pin PB1
	5	☐	SUT1	Select start-up time
	4	☒	SUT0	
	3	☒	CKSEL3	Select Clock Source
	2	☒	CKSEL2	
	1	☐	CKSEL1	
	0	☒	CKSEL0	
High	7	☐	OCDEN	Enable On Chip Debug
	6	☒	JTAGEN	Enable JTAG
	5	☒	SPIEN	Enable Serial programming and Data Downloading
	4	☐	WDTON	Watchdog timer always on
	3	☐	EESAVE	EEPROM memory is preserved through chip erase
	2	☒	BOOTSZ1	Select Boot Size
	1	☒	BOOTSZ0	
	0	☐	BOOTRST	Select Reset Vector
Extended	7			
	6			
	5			
	4			
	3			
	2	☐	BODLEVEL2	
	1	☐	BODLEVEL1	Brown-out Detector trigger level
	0	☐	BODLEVEL0	

Notes: ☐ means unprogrammed (1); ☒ means programmed (0).
The top five bits of the extended fuse are undefined.

TABLE 4: AVR Alternate Pin Functions for ATMEGA164/324/644/1284 family

Port	Pin	Name	Description
A	PA7	ADC7	ADC input channel 7
	PA6	ADC6	ADC input channel 6
	PA5	ADC5	ADC input channel 5
	PA4	ADC4	ADC input channel 4
	PA3	ADC3	ADC input channel 3
	PA2	ADC2	ADC input channel 2
	PA1	ADC1	ADC input channel 1
	PA0	ADC0	ADC input channel 0
B	PB7	SCK	SPI Bus Master clock input
	PB6	MISO	SPI Bus Master Input/Slave Output
	PB5	MOSI	SPI Bus Master Output/Slave Input
	PB4	$\overline{SS}$	SPI Slave Select input
		OC0B	Timer/Conter 0 Output Compare Match B Output
	PB3	AIN1	Analog Comparator Negative Input
		OC0A	Timer/Conter 0 Output Compare Match A Output
	PB2	AIN0	Analog Comparator Positive Input
		INT2	External Interrupt 2 Input
	PB1	T1	Timer/Counter 1 External Counter Input
		CLKO	Divided System Clock Output
	PB0	T0	Timer/Counter 0 External Counter Input
		XCK0	USART0 External Clock Input/Output
C	PC7	TOSC2	Timer Oscillator pin 2
	PC6	TOSC1	Timer Oscillator pin 1
	PC5	TDI	JTAG Test Data Input
	PC4	TDO	JTAG Test Data Output
	PC3	TMS	JTAG Test Mode Select
	PC2	TCK	JTAG Test Clock
	PC1	SDA	2-wire Serial Bus Data Input/Output Line
	PC0	SCL	2-wire Serial Bus Clock Line
D	PD7	OC2A	Timer/Counter2 Output Compare Match A Output
	PD6	ICP1	Timer/Counter1 Input Capture Trigger
		OC2B	Timer/Counter2 Output Compare Match B Output
	PD5	OC1A	Timer/Counter1 Output Compare Match A Output
	PD4	OC1B	Timer/Counter1 Output Compare Match B Output
		XCK1	USART1 External Clock Input/Output
	PD3	INT1	External Interrupt1 Input
		TXD1	USART1 Transmit Pin
	PD2	INT0	External Interrupt0 Input
		RXD1	USART1 Receive Pin
	PD1	TXD0	USART0 Transmit Pin
	PD0	RXD0	USART0 Receive Pin

Notes: Each pin also has a pin change interrupt.
Each pin can also be configured as GPIO.

Appendix C Code Examples in C for the AVR

To compile the code examples given here, use the following command to invoke the `avr-gcc` cross-compiler.

```
avr-gcc -mmcu=atmega644p -DF_CPU=12000000 -Wall -Os prog.c -o prog.elf
```

The options specify that the target part is an ATMEGA644P, that it is running at a clock frequency of 12MHz, that all compiler warnings should be displayed and that code size optimisations should be turned on. The resulting output is in ELF (Executable and Linking Format) which must be converted into HEX (raw binary executable) file for download to the target by executing the following command.

```
avr-objcopy -O ihex prog.elf prog.hex
```

Provided the boot loader has been installed and the fuses correctly configured the HEX file can now be downloaded with `AVRDUDE` over the USB interface on `X1` using the following command.

```
avrdude -c usbasp -p m644p -U flash:w:prog.hex
```

Program 1 is a simple program to toggle pin `PB7` on and off, which will also flash the LED (`D1`) if the jumper (JP3) is in the `Usr` position.

PROGRAM 1: LEDTest.c

```
/*  Author: Steve Gunn
 * Licence: This work is licensed under the Creative Commons Attribution License.
 *           View this license at http://creativecommons.org/about/licenses/
 *   Notes: F_CPU must be defined to match the clock frequency
 */
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    /* set LED pin as an output */
    DDRB |= _BV(PINB7);
    /* forever loop (embedded programs never normally terminate) */
    for (;;)
    {
        /* Set pin B7 high */
        PORTB |= _BV(PINB7);
        _delay_ms(100);
        /* Set pin B7 low */
        PORTB &= ~_BV(PINB7);
        _delay_ms(900);
    }
}
```

Program 2 is a take on the first program which is often written to display 'Hello World' on the screen. Currently our embedded platform has no screen, so we improvise and use Morse code to signal the message with the LED (`D1`). The program first defines a lookup table to store the translation of characters to the dots and dashes of Morse code. The `encode` function considers each character of the message in turn and tries to find it in the lookup table (it treats lower and upper case letters identically by mapping all characters to uppercase with the `toupper` function). If successful it passes the appropriate dots and dashes to the `output` function, which does the work of generating the correct timing. The spaces in the message string are added to provide a delay before the message repeats.

PROGRAM 2: HelloWorld.c

```
/*  Author: Steve Gunn
 * Licence: This work is licensed under the Creative Commons Attribution License.
 *          View this license at http://creativecommons.org/about/licenses/
 *   Notes: F_CPU must be defined to match the clock frequency
 */
#include <inttypes.h>
#include <ctype.h>
#include <avr/io.h>
#include <util/delay.h>

#define UNIT_LENGTH_MS  200

static const struct
{
   const char character, *morse;
} code[] =
{
   { 'A', ".-"   },
   { 'B', "-..." },
   { 'C', "-.-." },
   { 'D', "-.."  },
   { 'E', "."    },
   { 'F', "..-." },
   { 'G', "--."  },
   { 'H', "...." },
   { 'I', ".."   },
   { 'J', ".---" },
   { 'K', ".-.-" },
   { 'L', ".-.." },
   { 'M', "--"   },
   { 'N', "-."   },
   { 'O', "---"  },
   { 'P', ".--." },
   { 'Q', "--.-" },
   { 'R', ".-."  },
   { 'S', "..."  },
   { 'T', "-"    },
   { 'U', "..-"  },
   { 'V', "...-" },
   { 'W', ".--"  },
   { 'X', "-..-" },
   { 'Y', "-.--" },
   { 'Z', "--.." },
   { ' ', " "    },
};
```

```
47 void LED_on(void)
48 {
49     PORTB |= _BV(PINB7);
50     _delay_ms(UNIT_LENGTH_MS);
51 }

53 void LED_off(void)
54 {
55     PORTB &= ~_BV(PINB7);
56     _delay_ms(UNIT_LENGTH_MS);
57 }

59 void output(const char* o)
60 {
61     /*  International Morse code is composed of five elements:
62      *       dot: one unit long
63      *       dash: three units long
64      *       inter-element (between character elements) : one unit long
65      *       short gap (between letters) : three units long
66      *       medium gap (between words) : seven units long
67      */
68     uint8_t n = 0;
69     while (o[n])
70         switch (o[n++])
71         {
72             case '.' : LED_on(); LED_off(); break;
73             case '-' : LED_on(); LED_on(); LED_on(); LED_off(); break;
74             case ' ' : LED_off(); LED_off(); break;
75         }
76     LED_off(); LED_off();
77 }

79 void encode(const char *message)
80 {
81     uint16_t i;
82     uint8_t j;
83     for (i=0; message[i]; i++)
84         for (j=0; j<sizeof(code)/sizeof(*code); j++)
85             if (toupper(message[i]) == code[j].character)
86                 output(code[j].morse);
87 }

89 int main(void)
90 {
91     /* set LED pin as an output */
92     DDRB |= _BV(PINB7);
93     /* forever loop */
94     for (;;) encode("Hello World              ");
95 }
```

Program 3 is a simple program to test the capabilities of the first UART interface. It sends an initialisation message to the host and then waits for the user to press a key on the host terminal. The embedded platform then echos back the key that was typed to the host terminal.

PROGRAM 3: UARTTest.c

```
/*  Author: Steve Gunn
 * Licence: This work is licensed under the Creative Commons Attribution License.
 *          View this license at http://creativecommons.org/about/licenses/
 *   Notes: F_CPU must be defined to match the clock frequency
 */
#include <inttypes.h>
#include <avr/io.h>

void init_uart0(void)
{
    /* Configure 9600 baud, 8-bit, no parity and one stop bit */
    const int baud_rate = 9600;
    UBRR0H = (F_CPU/(baud_rate*16L)-1) >> 8;
    UBRR0L = (F_CPU/(baud_rate*16L)-1);
    UCSR0B = _BV(RXEN0) | _BV(TXEN0);
    UCSR0C = _BV(UCSZ00) | _BV(UCSZ01);
}

char get_ch(void)
{
    while(!(UCSR0A & _BV(RXC0)));
    return UDR0;
}

void put_ch(char ch)
{
    while (!(UCSR0A & _BV(UDRE0)));
    UDR0 = ch;
}

void put_str(char *str)
{
    int i;
    for(i=0; str[i]; i++) put_ch(str[i]);
}

int main(void)
{
    char ch;
    init_uart0();
    put_str("Hello from Il Matto\n\r");
    /* forever loop */
    for (;;)
    {
        /* get character from UART */
        ch = get_ch();
        /* send message back to the host terminal */
        put_str("You sent the character '");
        put_ch(ch);
        put_str("'\n\r");
    }
}
```

Program 4 is a program to test the ports GPIO interface. The program performs two types of test:

`port_test` considers each port in turn and cycles through each of the 8-bits, configuring that bit as an output and the others as inputs. The output is driven low and the inputs are tested to see whether they are affected by this change. If they are, it indicates a fault, which is likely caused by a short circuit.

`io_test` considers each pair of ports in turn, one configured as an output and the other configured as an input. It expects two or more ports to be connected together and tests to see which ones are connected, by toggling the bits of the output port and testing to see whether the input port can recognise this change. Assuming ports have been connected together, the failure to detect any change indicates an open circuit in the relevant bits reported by the program.

The results are stored in the EEPROM of the AVR, and when the tests are complete a watchdog reset is issued, causing a software reset of the AVR and invocation of the boot loader. The results can then be downloaded from the non-volatile EEPROM as a text file using AVRDUDE. To aid with test identification the first 8 bytes of the EEPROM are used to store a counter, which is incremented each time the test program is executed. The execution time of the program is around 8 seconds for a ATMEGA644P and is largely determined by the relatively slow write speed to the EEPROM.

PROGRAM 4: GPIOTest.c

```
/*  Author: Steve Gunn & Rex Bannerman
 * Licence: This work is licensed under the Creative Commons Attribution License.
 *           View this license at http://creativecommons.org/about/licenses/
 *   Notes: To minimise unwanted interactions in the test:
 *              - Power from the external supply
 *              - Attach interface cable for download after tests complete
 *
 *          LED D1 does not interfere with the test and JP3 should be set to Usr.
 *          Test results are limited to 512 bytes to be compatible with the EEPROM
 *          for all devices in the ATmega*4PA family. The first 8 bytes are used
 *          to store a test identifier, which is incremented on each new run.
 */

#include <stdio.h>
#include <string.h>
#include <avr/io.h>
#include <avr/eeprom.h>
#include <avr/wdt.h>
#include <util/delay.h>

#define PIN(port)   _SFR_IO8(0x00 + (((port) - 'A') * 3))
#define DDR(port)   _SFR_IO8(0x01 + (((port) - 'A') * 3))
#define PORT(port)  _SFR_IO8(0x02 + (((port) - 'A') * 3))

uint8_t write_eeprom(char *message, uint16_t index)
{
    uint8_t length = strlen(message);
    eeprom_write_block(message, (void *)index, length);
    return length;
}
```

```
32 void fill_eeprom(char c, uint16_t start_index)
33 {
34     uint16_t index;
35     for(index=start_index; index<=E2END; index++)
36         eeprom_write_byte((uint8_t *)index, c);
37 }

39 void update_test_identifier(void)
40 {
41     char id[8];
42     uint8_t i=4;
43     eeprom_read_block(id, 0, 8);
44     do  /* Increment test counter */
45         id[i] = (id[i]=='9' ? '0' : id[i] + 1);
46     while(id[i--]=='0' && i>0);
47     eeprom_write_block((id[0]=='{') ? id : "{0000}\r\n", 0, 8);
48 }

50 void port_test(char port, char* result)
51 {
52     uint8_t pattern, pass=0x00;
53     /* Set 8-bits of out port as outputs */
54     DDR(port) = 0xFF;
55     /* Set test pattern as 0b10000000 and shift bit for each test */
56     for(pattern=0x80; pattern>0x00; pattern >>= 1)
57     {
58         /* Set output bit low and turn on pull-ups on input bits  */
59         PORT(port) = 0xFF - pattern;
60         /* Configure 7-bits of port as inputs and 1-bit as output */
61         DDR(port) = pattern;
62         /* Small delay for synchronisation                        */
63         asm("nop");
64         /* Are all the input bits still high? (PASS)              */
65         if (PIN(port)==0xFF-pattern)
66             pass+=pattern;
67     }
68     sprintf(result, "%c->%c: 0x%02X 0x%02X [%s]\r\n", port, port,
69             pass, 0xFF - pass, (pass==0xFF) ? "PASS" : "FAIL");
70 }

72 void io_test(char out, char in, char* result)
73 {
74     uint8_t fl, fh, hi, lo;
75     PORT(in)  = 0xFF;  /* Enable pull-ups on input port     */
76     DDR(in)   = 0x00;  /* Set 8-bits of in port as inputs   */
77     PORT(out) = 0xFF;  /* Set all output bits high          */
78     DDR(out)  = 0xFF;  /* Set 8-bits of out port as outputs */
79     asm("nop");        /* Small delay for synchronisation   */
80     hi = PIN(in);      /* Read inputs with pull-ups         */
81     PORT(out) = 0x00;  /* Set all output bits low           */
82     asm("nop");        /* Small delay for synchronisation   */
83     lo = PIN(in);      /* Read inputs with pull-ups         */
84     PORT(in)  = 0x00;  /* Disable pull-ups on input port    */
85     asm("nop");        /* Small delay for synchronisation   */
86     fl = PIN(in);      /* Read inputs with tri-state        */
87     PORT(out) = 0xFF;  /* Set all output bits high          */
88     asm("nop");        /* Small delay for synchronisation   */
89     fh = PIN(in);      /* Read inputs with tri-state        */
90     sprintf(result, "%c->%c: 0x%02X 0x%02X 0x%02X 0x%02X [%s]\r\n", out, in,
91             hi, lo, fh, fl, (hi==0xFF && lo==0x00) ? "PASS" : "FAIL");
```

```
92  }
94  int main(void)
95  {
96      char out, in, test_result[255];
97      uint16_t eeprom_index = 8;
98      /* Disable watchdog timer */
99      MCUSR &= ~_BV(WDRF);
100     wdt_disable();
101     /* Disable JTAG in software, so that it does not interfere with Port C        */
102     MCUCR |= _BV(JTD);
103     MCUCR |= _BV(JTD);
104     /* Perform tests and store the results in the EEPROM.                         */
105     update_test_identifier();
106     for(out='A'; out<='D'; out++)
107         for(in='A'; in<='D'; in++)
108         {
109             if (out==in)
110                 port_test(out, test_result);
111             else
112                 io_test(out, in, test_result);
113             eeprom_index += write_eeprom(test_result, eeprom_index);
114         }
115     fill_eeprom(' ', eeprom_index);
116     /* Enable watchdog timer and wait for it to generate a reset to enter the BL. */
117     wdt_enable(WDTO_15MS);
118     while(1);
119 }
```

Appendix D USB Boot Loader

The AVR flash memory is organised into two main sections, the Application section and the Boot Loader section (the size of these sections is determined by the BOOTSZ fuses) [5]. It is possible to install a program into the boot loader section which can then be used to load a new program into the application section. In practice, you can think of it as special code which tells the AVR how to download a program. At first this may seem confusing, but it enables software communication for protocols such as USB, which would normally require additional dedicated hardware to support program download. (The fact that you are able to connect the device to a USB port with the C232HM cable is because it provides dedicated hardware for USB communications.)

The boot loader described here is attractive because it enables the Il Matto board to be directly connected to a USB host without the need for any additional hardware. The code is based on the USBaspLoader software [10] which uses the V-USB driver [11]. A couple of small modifications are introduced to control the entry and exit from the boot loader.

> *The boot loader enables the device to communicate with a host PC for program download over a USB cable plugged into X1 (fuse configuration is not possible). Once installed and enabled it can be activated by pressing the reset button (S2), or from software by executing* `wdt_enable(WDTO_15MS); while(1);`.
> *To restart the main application without going through the boot loader you can power cycle the device with S1.*

An important decision when using a boot loader is to determine when the main application program should execute and when the boot loader should execute. Here, we make the decision based on the type of reset event, enabling the boot loader to be activated from both hardware (using an external reset) or software (using a watchdog reset). The reset circuitry for the microcontroller is shown in figure 8. After a reset event the microcontroller is able to determine which of the five reset events (JTAG, Watchdog, Brown-out, External, or Power-on) occurred by inspecting the MCU status register (MCUSR). Once the boot loader is installed and the fuses are correctly set, the boot loader code is run every time the device receives any reset event. The first thing the code does is to test for the required condition (an external or a watchdog reset) and if it is met, it jumps to main routine of the boot loader code to communicate with a USB host. It also clears the MCUSR so that the current reset event does not mask any future event and then it turns on LED (D1) to indicate the boot loader is active.

```
#define bootLoaderCondition()   (MCUSR & (_BV(EXTRF) | _BV(WDRF)))
```

If the condition is not met it jumps straight to the application section to start the main program.

The main communication routine in the boot loader uses the standard V-USB code to communicate with a USB host. To do this it must have precise timing and the inner functions of boot loader are written in assembly to provide this precision. A minimum frequency of 12 MHz is required to meet

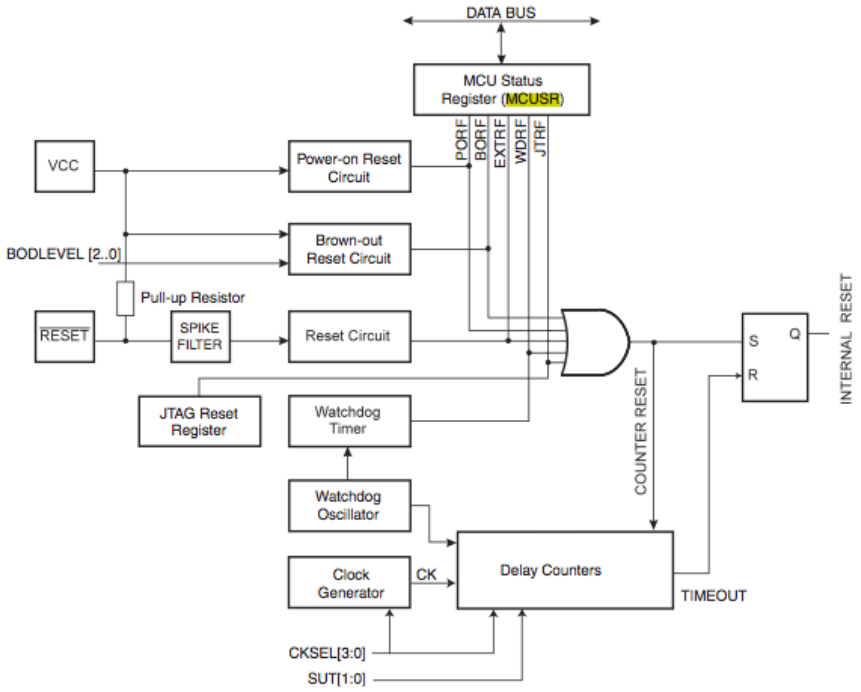


FIGURE 8: AVR reset circuit (reproduced from the ATMEGA644P datasheet [5])

the USB specification (which conveniently lies just inside the upper 13.3 MHz limit for an AVR running at 3.3V, such as `Il Matto`). It is necessary to use an external crystal oscillator to provide the timing precision (< 2000ppm) at this frequency.

A further requirement for the software to operate is that the USB `D+` line must be connected to a pin with an external interrupt, and the `D-` line must be connected to another pin on the same port. There are three external interrupt pins on this AVR and `INT2` on `PB2` is chosen for `D+` and `PB3` is chosen for `D-`.

When the device is plugged into a PC and the reset button (`S2`) is pressed it should now appear as a `USBasp` device. Windows requires the `libusb` driver to be installed, but Linux/Mac should work without a driver. It should now be possible to connect to the device using `AVRDUDE` with the command

```
avrdude -c usbasp -p m644p
```

When an `AVRDUDE` command is complete the boot loader will turn off LED (`D1`). The boot loader then exits and starts any program residing in the application section. If you wish to enter multiple `AVRDUDE` commands you will have to press `S2` after entering each one, or more conveniently you can exploit the *terminal mode* of `AVRDUDE` to avoid having to press `S2` more than once, by invoking `AVRDUDE` with the `-t` option.

```
avrdude -c usbasp -p m644p -t
```

Once in terminal mode type `help` to see the available commands and syntax.

Appendix E C232HM Cable

The C232HM is a multi-purpose cable produced by FTDI that simplifies the connection of electronic devices to a modern PC [4]. It provides all the necessary handshaking and royalty free software drivers for all common platforms for interfacing to a USB port. The device is based around the FT232H IC which is a single channel USB 2.0 Hi-Speed (480Mb/s) UART/FIFO that can also be configured to communicate in a variety of industry standard serial or parallel interfaces [3]. The cable is fitted with ten wires terminated in single pole connectors at the one end (table 5), which can be interfaced to a male header or have single stranded wire pushed into them. The other end of the 0.5m long cable has a moulded USB plug containing the circuitry (figure 9). The cable is powered from the USB host port and supports a data transfer rate up to 30Mbps in MPSSE mode. There are two variants of this cable

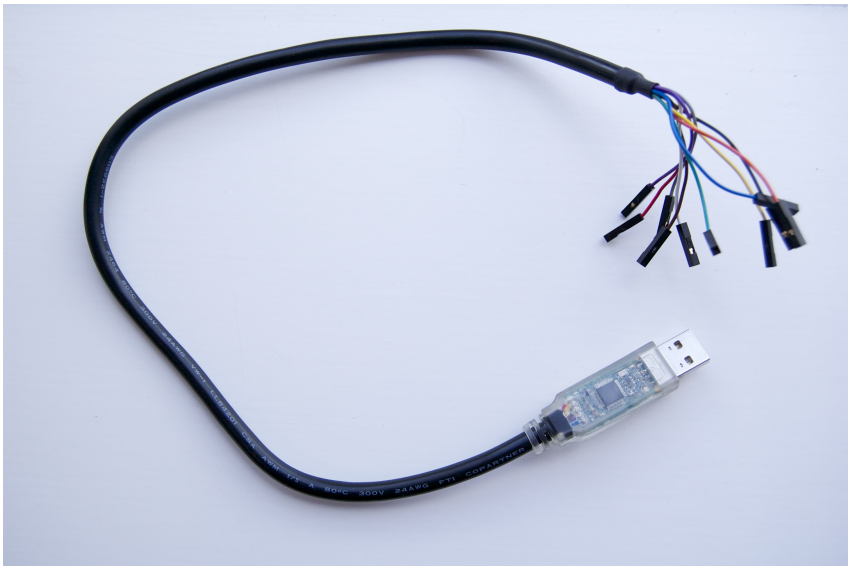


FIGURE 9: FTDI C232HM cable

which differ only in the value of VCC provided on pin 1. You have been supplied with the C232HM-DDHSL-0 variant which has a VCC of 3.3V and can supply 250mA of current. The cable signals are compliant with CMOS logic at 3.3 volts.

TABLE 5: FTDI C232HM cable pinout

Pin	Colour		JTAG	SPI	I^2C	ISP	UART
1	Red		VCC	VCC	VCC	VCC	VCC
2	Orange		TCK	SK	SCL	SCK	TXD
3	Yellow		TDI	DO	SDA	MOSI	RXD
4	Green		TDO	DI	SDA	MISO	$\overline{\text{RTS}}$
5	Brown		TMS	CS	—	RST	$\overline{\text{CTS}}$
6	Gray		GPIOL0	GPIOL0	GPIOL0	GPIOL0	$\overline{\text{DTR}}$
7	Purple		GPIOL1	GPIOL1	GPIOL1	GPIOL1	$\overline{\text{DSR}}$
8	White		GPIOL2	GPIOL2	GPIOL2	GPIOL2	$\overline{\text{DCD}}$
9	Blue		GPIOL3	GPIOL3	GPIOL3	GPIOL3	$\overline{\text{RI}}$
10	Black		GND	GND	GND	GND	GND

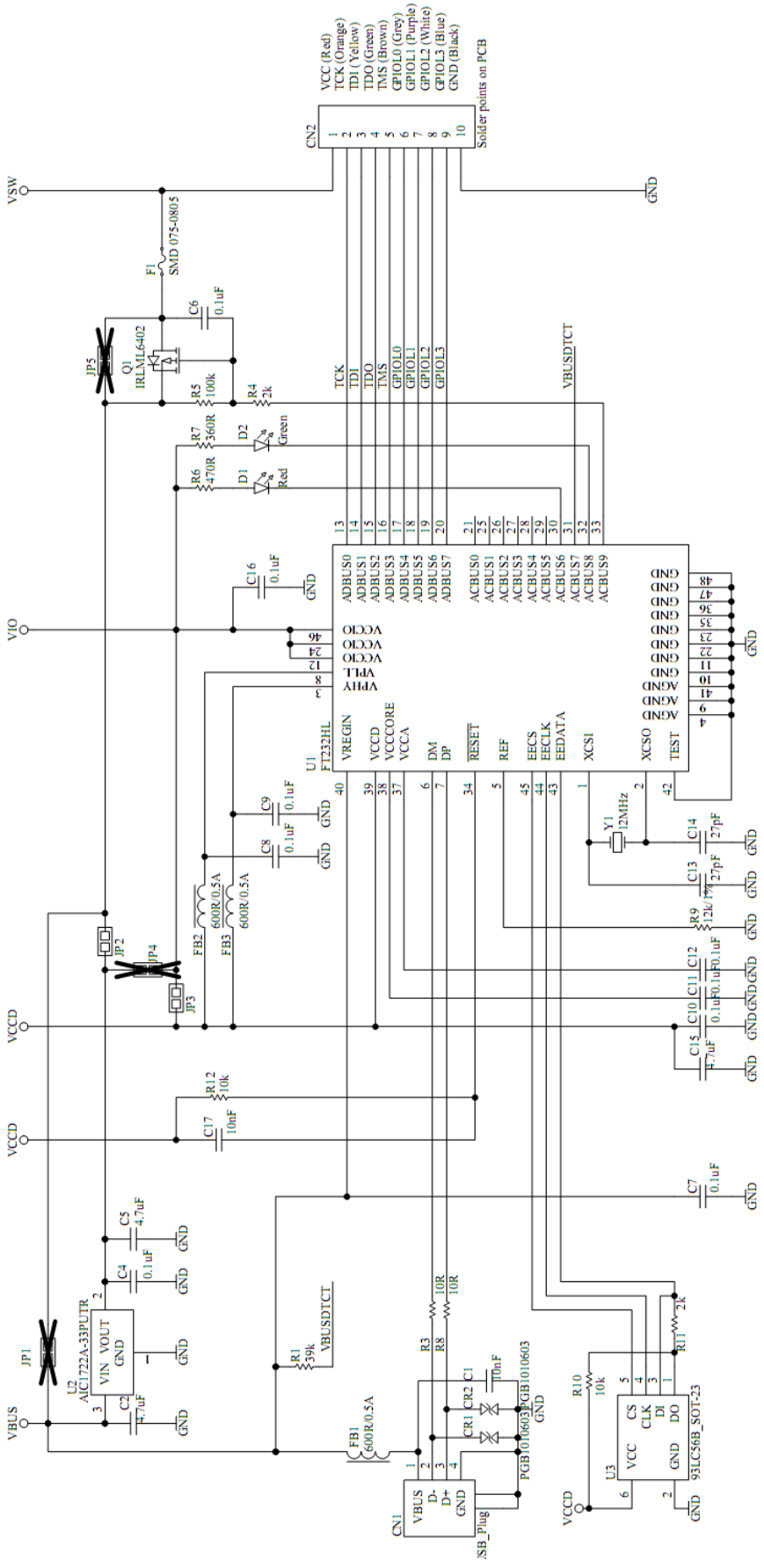


FIGURE 10: FTDI C232HM cable schematic [4]

Appendix F AVRDUDE Software

AVRDUDE is a program for downloading[8] (writing to) and uploading (reading from) the on-chip memory of an AVR microcontroller [6]. It supports a wide range of programming hardware and is able to program the Flash memory and EEPROM, and in some cases also the fuses and lock bits. It has been developed as an open source project and it is hosted at http://www.nongnu.org/avrdude. The intention of this appendix is to give a quick start guide to the most widely used commands, but for further details consult the manual [6].

List the supported programming hardware

```
avrdude -c ?
```

List the supported microcontrollers

```
avrdude -p ?
```

In general the program will be invoked with two parameters specifying the programmer (-c) and the device to be programmed (-p), followed by a number of actions to be performed on the memory or fuses (-U). Note that the options are case-sensitive. The following two examples show how to read and write the program memory of an ATMEGA644P microcontroller using the usbasp programmer.

Write Flash program memory

```
avrdude -c usbasp -p m644p -U flash:w:download.hex
```

Read Flash program memory and write to a file

```
avrdude -c usbasp -p m644p -U flash:r:upload.hex
```

To configure the fuses an alternative programmer with fuse programming capability is required, such as the c232hm[9].

Write Fuses from hex values specified on the command line

```
avrdude -c c232hm -p m644p -U lfuse:w:0xff:m -U hfuse:w:0x99:m -U efuse:w:0xff:m
```

Read Fuses and write the values to a file in hex format

```
avrdude -c c232hm -p m644p -U lfuse:r:lf.txt:h -U hfuse:r:hf.txt:h -U efuse:r:ef.txt:h
```

[8] This document adopts the uploading/downloading directions used in section 2.3 of the AVRDUDE documentation [6].
[9] Requires patched version of AVRDUDE (http://helix.air.net.au/index.php/avrdude-and-ftdi-232h/)

Glossary

Hardware

Breadboard	Solderless prototyping board with push in contacts
DIL	Dual-In-Line
EEPROM	Electrically Erasable Programmable Read Only Memory
GND	Ground
IC	Integrated Circuit
LDO	Low Drop-Out
LiPo	Lithium Polymer
LED	Light Emitting Diode
PCB	Printed Circuit Board
PTH	Plated Through Hole
PSU	Power Supply Unit
SMD	Surface Mount Device
SPDT	Single Pole Double Throw
THC	Through-Hole Component
VCC	Positive Supply Voltage[1]

Interfacing

ADC	Analogue to Digital Converter
FIFO	First In First Out
GPIO	General Purpose Input Output
JTAG	Joint Test Action Group (in-circuit programming and debugging)
I^2C	Inter-Integrated Circuit bus (serial communications)
ISP	In System Programming (in-circuit programming)
SPI	Serial Peripheral Interface (serial communications)
UART	Universal Asynchronous Receiver/Transmitter (serial communications)
USB	Universal Serial Bus (serial communications)

Software

`AVRDUDE`	AVR Downloader Uploader (programming AVRs)
`avr-gcc`	Cross compiler tools for AVR (building AVR programs)
`hyperterminal`	Terminal emulator (serial communications) [Windows]
`PuTTY`	Terminal emulator (serial communications) [Windows]
`screen`	Terminal emulator (serial communications) [Linux/Mac]
`UrJTAG`	Universal JTAG library, server and tools (JTAG programming)

[1] For a history see http://en.wikipedia.org/wiki/IC_power_supply_pin

References

[1] Amprobe. Professional Digital Multimeter (30XR-A). Users Manual 9/06, 2006. URL https://secure.ecs.soton.ac.uk/notes/ellabs/reference/equipment/std-bench/DMM%20-%20Amprobe%2030XR-A.pdf.

[2] R. Bannerman. An Installation and Setup Guide for Hosted and Embedded C Environments. 2012.

[3] FTDI Chip. FT232H Single Channel Hi-Speed USB to Multipurpose UART/FIFO IC. Datasheet FT_000288, Future Technology Devices International Ltd, 2012. URL http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232H.pdf.

[4] FTDI Chip. C232HM USB Hi-Speed to MPSSE Cable Datasheet. Datasheet FT_000401, Future Technology Devices International Ltd, 2012. URL http://www.ftdichip.com/Support/Documents/DataSheets/Cables/DS_C232HM_MPSSE_CABLE.PDF.

[5] Atmel Corporation. ATMEGA164PA/324PA/644PA/1284P Datasheet. Datasheet 8152G-AVR-11/09, 2011. URL http://www.atmel.com/Images/doc8152.pdf.

[6] Brian S. Dean. AVRDUDE A program for download/uploading AVR microcontroller flash and eeprom. Manual 5.11.1, 2011. URL http://download.savannah.gnu.org/releases/avrdude/avrdude-doc-5.11.1.pdf.

[7] Thurlby Thandar Instruments. Bench Power Supply (EL302P). Instruction Manual, 2009. URL https://secure.ecs.soton.ac.uk/notes/ellabs/reference/equipment/std-bench/PSU%20EL302P%20Instruction%20Manual%20-%20Iss%205.pdf.

[8] Curious Inventor. How to Solder, 2007. URL http://store.curiousinventor.com/guides/How_To_Solder/.

[9] Saleae. Logic Analyser (Logic). User's Guide 1.1.15, 2012. URL https://secure.ecs.soton.ac.uk/notes/ellabs/reference/equipment/std-bench/Logic%20Analyser-%20Saleae%20User%20Guide.pdf.

[10] Christian Starkjohann. USBaspLoader for AVR microcontrollers. Software, 2010. URL http://www.obdev.at/products/vusb/usbasploader.html.

[11] Christian Starkjohann. Virtual USB port for AVR microcontrollers. Software, 2012. URL http://www.obdev.at/products/vusb/.

[12] Tektronix. Digital Storage Oscilloscope (TDS1000C). User Manual Rev. A, 2006. URL https://secure.ecs.soton.ac.uk/notes/ellabs/reference/equipment/std-bench/Tektronix_TDS1000C_2000C_User_Manual.pdf.