

University of Southern Queensland  
Faculty of Engineering & Surveying

## **Embedded IP for Small Devices**

A dissertation submitted by

Simon Neil Brown

in fulfilment of the requirements of

**ENG4112 Research Project**

towards the degree of

**Bachelor of Engineering (Computer and Electronic)**

Submitted: October, 2005

# Abstract

Today technology utilising the Web is one of the most popular used computer technologies. It would be hard to imagine any computer user whom does not have a web browser or used a web browser. A web browser can view web-pages developed or located within any operating systems, whether that is a Windows, Linux or even iMac workstation. The beauty of this technology is that the web client software (Web Browser) can communicate with any web-server using the Hyper-text Transfer Protocol (HTTP). Also the pages displayed by these systems look identical even though they are generated by a variety of computer systems.

With embedded systems in mind, it would be silly not to utilise this technology for control and monitoring purposes. However, currently small embedded devices, those that are classified with less than 10 kB of ROM, have limited IP connectivity. This is because the current implementations occupy more than the device possesses.

The driver for this research project is the apparent lack of available IP implementations for small embedded devices. Typical embedded IP stacks range from 14kB up to and exceeding 500kB. For small devices, less than 10 kB, this puts this function out of reach.

However, this project aims to implement a subset of Internet Protocols to provide a means of control and monitoring for a small embedded device. It is envisaged that control and monitoring will be achieved with the use of a Web Browser, such as Microsoft "Internet Explorer". The project goals is to provide these services within a 2kB envelope.

University of Southern Queensland  
Faculty of Engineering and Surveying

<b>ENG4111/2 <i>Research Project</i></b>
--

### **Limitations of Use**

The Council of the University of Southern Queensland, its Faculty of Engineering and Surveying, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Engineering and Surveying or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled “Research Project” is to contribute to the overall education within the student’s chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

**Prof G Baker**

Dean

Faculty of Engineering and Surveying

# Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

SIMON NEIL BROWN

0050029170

---

Signature

---

Date

# Acknowledgments

To Natalie, Jacqueline and Samuel for their understanding and support during this worthwhile journey.

A special thanks to my supervisor Dr John Leis, who has provided guidance, a source of motivation and friendship over the past year.

SIMON NEIL BROWN

*University of Southern Queensland*

*October 2005*

# Contents

<b>Abstract</b>	<b>i</b>
<b>Acknowledgments</b>	<b>iv</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Tables</b>	<b>xi</b>
<b>Nomenclature</b>	<b>xii</b>
<b>Chapter 1 Introduction</b>	<b>1</b>
1.1 Overview of the Dissertation . . . . .	2
<b>Chapter 2 Existing System Evaluation</b>	<b>4</b>
2.1 Chapter Introduction . . . . .	4
2.2 Background Information . . . . .	4
2.3 Short-listed Implementations . . . . .	6
2.4 Code Size . . . . .	8

<b>CONTENTS</b>	<b>vi</b>
2.5 Functions . . . . .	10
2.6 Security . . . . .	11
2.7 Chapter Summary . . . . .	11
<b>Chapter 3 Embedded Platform and Development Tools</b>	<b>13</b>
3.1 Chapter Introduction . . . . .	13
3.2 Microprocessor Core . . . . .	13
3.3 Software and Tools . . . . .	14
3.4 Chosen System: Hardware . . . . .	15
3.5 Chosen System: Software . . . . .	17
3.6 Chapter Summary . . . . .	20
<b>Chapter 4 Software Development</b>	<b>21</b>
4.1 Chapter Introduction . . . . .	21
4.2 Software Overview . . . . .	21
4.3 High-level Data Link Control . . . . .	24
4.4 Point to Point Protocol . . . . .	25
4.4.1 Link Control Program . . . . .	26
4.4.2 Password Authentication Protocol . . . . .	27
4.4.3 Network Control Protocol . . . . .	28
4.5 Internet Protocol . . . . .	29

<b>CONTENTS</b>	<b>vii</b>
4.6 Transmission Control Protocol . . . . .	30
4.7 Hyper-text Transfer Protocol . . . . .	32
4.8 RAM Utilisation . . . . .	33
4.9 Code Optimisation . . . . .	34
4.10 Chapter Summary . . . . .	34
<b>Chapter 5 Testing and Security</b>	<b>35</b>
5.1 Chapter Introduction . . . . .	35
5.2 Testing So Far . . . . .	35
5.3 Security . . . . .	37
5.4 Environmental and Hardware . . . . .	38
5.5 Chapter Summary . . . . .	38
<b>Chapter 6 Conclusions and Further Work</b>	<b>39</b>
6.1 Achievement of Project Objectives . . . . .	39
6.2 Further Work . . . . .	40
6.3 Conclusion . . . . .	42
<b>References</b>	<b>43</b>
<b>Appendix A Project Specification</b>	<b>48</b>
<b>Appendix B Code Listing</b>	<b>50</b>



---

B.1 Source Code Listing . . . . .	51
<b>Appendix C Manufacturer Datasheets</b>	<b>86</b>
C.1 JED Micoprocessor Datasheets . . . . .	86
C.2 AVR ATmega128 Datasheet . . . . .	97
C.3 AVR ISP, In-System Programmer . . . . .	125
C.4 AVR033: Getting Started with the CodeVisionAVR C Compiler . . . .	128
C.5 LCD Display Datasheet . . . . .	145
C.6 RS232 Transceiver Datasheet . . . . .	150

# List of Figures

3.1	The AVR570 module from “JED Microprocessor”. Note the In-System Programmer interface (top right), reset switch in yellow (bottom left) and the 64 pin surface mounted microcontroller (centre). . . . .	16
3.2	The AVR572 development board from “JED Microprocessor” revealing the AVR570 module and the RS232 transceiver and DB-9 connector (middle left) . . . . .	17
3.3	A screen snapshot of the ‘AVRCodeVision’ IDE. File management facilities are revealed on the (left) sub-window with the main coding area (right). Convenient buttons are available for various functions including, saving, compiling, searching and target programming are located beneath the pull-down menus. A status message window is available for debugging and monitoring purposes (bottom) . . . . .	18
3.4	AVRCodeVision Device Programmer Interface. . . . .	19
4.1	Simplified Flow Diagram. . . . .	23
4.2	PPP General Frame Format, utilising HDLC. . . . .	24
4.3	Windows ME connection established window. . . . .	28
4.4	IP Frame, note the relative position of the checksum, which is within the header. . . . .	29

---

4.5	TCP segment data transfer. . . . .	31
4.6	Browser Display of Served Web-Page from the embedded device. . . . .	32
4.7	Plethora of packets received on establishment of Link. . . . .	33
5.1	Screen shot of Ethereal Packet Analyser revealing a malformed checksum calculation. . . . .	36

# List of Tables

2.1	Short-Listed IP Implementations and approximate cost . . . . .	7
2.2	Code Size . . . . .	9
2.3	Functions for the Short-listed Implementations . . . . .	10
4.1	LCP Configuration Options . . . . .	26

# Nomenclature

PPP	Point to Point Protocol
LCP	Link Control Protocol
PAP	Password Authentication Protocol
IPCP	Internet Protocol Control Protocol
IP	Internet Protocol
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
DHCP	Dynamic Host Configuration Protocol
HTTP	HyperText Transfer Protocol
ASCII	American Standard Code for Information Interchange
AT	Attention
CRC	Cyclic Redundancy Check
IC	Integrated Circuit
LED	Light Emitting Diode
UART	Universal Asynchronous Receiver / Transmitter
PC	Personal Computer
USQ	University of Southern Queensland
CPU	Central Processing Unit

RAM	Random Access Memory
ROM	Read Only Memory
IDE	Integrated Development Environment
ISP	In-System Programming
NCP	Network Control Protocol
JVM	Java Virtual Machine
AUD	Australian Dollars
SPI	Serial Peripheral Interface
RTC	Real Time Clock
LCD	Liquid Crystal Display
GPRS	General Packet Radio Service

# Chapter 1

## Introduction

Today technology utilising the Web is one of the most popular used computer technologies. It would be hard to imagine any computer user whom does not have a web browser or used a web browser. A web browser can view web-pages developed or located within any operating systems, whether that is a Windows, Linux or even iMac workstation. The beauty of this technology is that the web client software (Web Browser) can communicate with any web-server using the Hyper-text Transfer Protocol (HTTP). Also the pages displayed by these systems look identical even though they are generated by a variety of computer systems.

With embedded systems in mind, it would be silly not to utilise this technology for control and monitoring purposes. However, currently small embedded devices, those that are classified with less than 10 kB of ROM, have limited IP connectivity. This is because the current implementations occupy more than the device possesses.

The driver for this research project is the apparent lack of available IP implementations for small embedded devices. Typical embedded IP stacks range from 14kB up to and exceeding 500kB. For small devices, less than 10 kB, this puts this function out of reach.

However, this project aims to implement a subset of Internet Protocols to provide a means of control and monitoring for a small embedded device. It is envisaged that

control and monitoring will be achieved with the use of a Web Browser, such as Microsoft “Internet Explorer”. The project goal is to provide these services within a 2kB envelope.

The device will communicate via a PPP link, which would commonly be a serial link (EIA232) interfaced via a modem but just as easily it could just be a direct serial connection to a PC, in the form of EIA232 or USB. The PPP link, together with the IP, will provide the underlying communication path for the higher level protocols such as UDP, TCP and DHCP which in turn provide the baseline software platform to provide the application layer with end to end reliable transport.

The most challenging aspect of this project has been the goal to implement these services utilising only 2kB of text space. Currently the majority of these services have been implemented in 'C' and ported to the end target, with a current code envelope of approximately 8 kB. This dissertation provides a documented journey of this process.

As with all research projects further work is required to engineer the product for manufacture and commercial release and these issues are also briefly discussed.

## 1.1 Overview of the Dissertation

This dissertation is organized as follows:

**Chapter 2** presents and discusses the evaluation of existing commercial and open source implementations currently available and evaluates them in terms of cost, function and security.

**Chapter 3** discusses, evaluates and selects a suitable micro-controller core, a development environment and a suitable programming language for the development of a working prototype. The evaluation aims to select software tools and hardware to provide a portable system.

**Chapter 4** discusses the software development with particular attention to the issues faced with small embedded devices with limited resources.



**Chapter 5** discusses testing and security issues and considerations of embedded systems providing IP services.

**Chapter 6** concludes the dissertation and suggests further work in the area of “Embedded IP for Small Devices”.

## Chapter 2

# Existing System Evaluation

### 2.1 Chapter Introduction

This chapter presents and discusses the background research and evaluation of existing systems available. Details are revealed of the results of this search and then evaluated in terms of cost, security and function. Commercial and open source products are considered.

### 2.2 Background Information

In line with the project objectives, as revealed in Appendix A, the first task in the research project programme was the literature research of available systems and the evaluation and review of these systems. For reader ease the first two programme requirements are restated:

1. Research available systems.
2. Evaluate available systems in terms of cost, security and function, both open source and commercial

Before any review and evaluation studies begun it was important to gain an understanding of the technical requirements and technical nomenclature used in their description. The most concise documentation and literature found was the “Request For Comment” (RFC) documents.

The RFCs provide an open platform for the development of computer networking, focussing on the the “internet”. The following RFCs where found to be most useful in the initial understanding of the project technical requirements;

**RFC793:** Transmission Control Protocol

**RFC1547:** Requirements for an Internet Standard Point to Point Protocol

**RFC1661:** The Point to Point Protocol

**RFC2131:** Dynamic Host Configuration Protocol

**RFC0791:** Internet Protocol

**RFC2616:** Hypertext Transfer Protocol

However, it was found that although the RFCs provided a concise specification of the protocols and systems they generally lacked any real world implementation details. Two texts, “TCP/IP Lean” (Bentham 2003) and the ebook “The TCP/IP Guide” (Kozierok 2003-2005) were found to provide more practical and useful information, and were referenced often during the project.

The review process began with an exhaustive search via the internet for commercial and open source implementations. In addition to the “internet” search the resources of the University Library, both USQ and Monash, as well as trade journals (*Embedded Computer Design, Trade Journal* 2005, *PC/104 Embedded Solutions, Trade Journal* 2005) specialising in embedded technology and systems were consulted.

An initial large list of embedded implementations was produced and details pertaining to the products features where recorded on a spreadsheet. Where details were not clearly stated in marketing literature, details were requested, usually by email. In the

first iteration 37 products were identified, in various forms, including both commercial and open source products, but was reduced to around 25 in the second iteration.

The list of 25 products was further reduced. In a number of cases insufficient information was available for the evaluation process and consequently these were removed. In most cases it was due to lack of a response to an enquiry. The final list included 17 implementations and is discussed in the next section.

## 2.3 Short-listed Implementations

It was clear there are abundant commercial and open-source implementations available from the results of my searches. A table of the short-listed implementations was made, primarily filtered to reveal those which supported the services we were interested in, namely PPP, IP, TCP, UDP, DHCP and HTTP.

In an attempt to keep an even playing field, pricing was requested for a single product, complete for a small quantity of product of less than 500 units.

The pricing collected for each product is revealed in Table 2.1<sup>1</sup>. The commercial offerings came with varied and in some cases high price tags. From the literature gathered, the justification for the price tags were summarised as follows:

- a. Fully RFC Compliant
- b. Warranty or Guarantee of stability and function.
- c. Technical Support
- d. Implementations, in some cases, offered for a large range of processors.
- e. Available off the shelf.
- f. Technical Performance such as throughput or memory / resource usage.

---

<sup>1</sup>The majority of quotations were provided in US dollars or Euro and have been converted to Australian dollars. The exchange rate used was 0.77 and 0.597, which was current as of March 2005 for US dollars and Euro respectively.

The pricing did vary between products and it was hard to gauge to what effect the pricing offered reflected product quality, stability, performance or efficiency. Some companies offered demonstration code and differing licensing plans which seemed to indicate different products had differing markets and market focus.

Table 2.1: Short-Listed IP Implementations and approximate cost

<b>Product</b>	<b>Open Source or Commercial</b>	<b>Approximate Price (AUD)</b>
BLUNK Microsystems	Commercial	\$12,730
CMX Systems	Commercial	\$12,990
EBSnet	Commercial	\$22,100
EmINET Microsystems	Commercial	\$23,380
Ethernut	Open Source	\$0
InterNiche Technologies	Commercial	\$22,100
Iosoft Ltd	Commercial	\$2,300
Kadak Products	Commercial	\$27,530
MicriUm	Commercial	\$20,150
NexGen	Commercial	\$18,700
On Time	Commercial	\$11,730
Quadros	Commercial	\$20,130
Rabbit	Commercial	\$500
TINI Network Platform	Open Source	\$0
Tiny TCP	Open Source	\$0
uIP 0.9, Contiki, lwIP & Miniweb	Open Source	\$0
US Software	Commercial	\$25,300

Many of the products listed, are bound by licensing agreements. Licensing agreements are typically offered as single product licences, with and without royalties payable for each product sold, single site licences, for any development at one site or project and unlimited use licences. The unlimited licences usually included technical support for a limited period from purchase.

The Open Source implementations also varied in their nature. Three of the short-

listed implementations didn't include a PPP option, while the fourth, the TINI<sup>2</sup>, has a preloaded 64kB coded section which already contained a network stack. The TINI Dallas product supports full TCP/IP stack, including PPP, and was ready for development in assembly, C or JAVA languages.

## 2.4 Code Size

In line with the project goals of implementing a stack within a 2kB code envelope, information was gathered relating to the compiled size of a number of implementations. The results are shown in Table 2.2. From the results it was quite interesting to see a large variation in the compiled sizes, even within the same product. However it would be dangerous to make a judgement of these implementations without a detailed evaluation<sup>3</sup>. Nevertheless it is clear from the comparison between the two 'CMX' compilations that the selection of the end target can be important. The selection of differing word platform sizes reveals the efficiency of the compiler math routines, in the case of the two 'CMX' implementations, between 8 bit and 16 bit cores. Mathematics used within the IP and TCP protocols headers use identification and sequencing numbers with numbers ranging from 16 bits to 32 bits, which require large math routines for 8 bit devices and hence the large variations in size.

Characteristic effecting the compiled sizes were found to be:

1. Platform word size (8, 16, 32 bit ...).
2. Instruction word size.
3. Technical Performance such as throughput and memory / resource usage.
4. Functions offered.
5. Differences between compilers and optimisers used.

---

<sup>2</sup>It may be argued that this implementation should actually be label as "commercial", as the stack is within a purchased product.

<sup>3</sup>Which of course could take years to complete and is certainly beyond the scope of this research project.

6. RFC compliant.

7. Stability and security.

Table 2.2: Code Size

Product	Platform	Functions	Code Size(kB)
uIP	AVR (8-bit)	IP, ICMP & TCP	5.2
CMX-MicroNet	Freescall HCS12 (16-bit)	Modem, PPP, IP, UDP, TCP & HTTP server	14.2
CMX-MicroNet	Atmel AVR (8-bit)	Modem, PPP, IP, TCP & HTTP server	33
BLUNK Microsystems	32-bit Processors	Full protocol suite, PPP, IP, TCP, UDP, ICMP, FTP, DHCP etc ...	32 to 64
InterNiche Technologies	Philips 2100 (32-bit)	Modem, PPP, IP, TCP & HTTP	35
Iosoft, PWEB	Microchip PIC (8-bit)	Modem, SLIP, IP, TCP & HTTP	5.7
Dallas, TINI	DS80C400 (8-bit) confirm	PPP, IPv4/v6, TCP, UDP, IGMP, ICMP, DAD, SMTP, DHCP, FTP, HTTP, & TELNET	64

## 2.5 Functions

For completeness, the functions of the short-list are compared to the requirements of the project goals. A comparison is revealed in Table 2.3 of the implementations that supported the services required. It was interesting to note, from the previous table, Table 2.2, that the smallest implementation with the services of interest was 14.2 kB on a 16 bit platform and from 32kB for an 8 bit device.

Table 2.3: Functions for the Short-listed Implementations

Product	PPP, IP, UDP, TCP, DHCP & HTTP
BLUNK Microsystems	Yes
CMX Systems	Yes
EBSnet	Yes
EmINET Microsystems	Yes
Ethernut	Yes
InterNiche Technologies	Yes
Iosoft Ltd	No PPP, only SLIP
Kadak Products	Yes
MicriUm	Yes
NexGen	No DHCP
On Time	Yes
Quadros	Yes
Rabbit	Yes
TINI Network Platform	Yes
Tiny TCP	No PPP
uIP 0.9, Contiki, lwIP & Miniweb	No PPP, only SLIP
US Software	Yes



---

## 2.6 Security

An evaluation of security aspects of embedded system revealed that many of the implementations offer a version of IP called IPsec. This protocol provides encryption of IP data and is a mandatory option of the new IP protocol, IPv6. As with any additional feature additional resources are required to handle the additional overhead of these services.

Other security aspects are the authentication systems available. Of particular interest to PPP is that of the Password authentication Protocol (PAP) and the Challenge Handshake Authentication Protocol (CHAP). These protocols provide methods of authentication to which SLIP does not offer. Virtually all implementations offered PAP and CHAP, however there were a few whom only offered PAP.

Of all implementations evaluated the marketing literature did not seem to discuss issues relating malicious attacks, such as Denial of Service attacks. It seems that software companies do not wish to discuss these issues or possibly there is an inherent weakness in embedded systems?

## 2.7 Chapter Summary

Investigations have found that there wasn't an implementation that satisfied the project requirements for providing PPP, IP, UDP, TCP, DHCP and HTTP services within a 2kB envelope. However it was interesting to note that two of the open-source implementations, although not providing all the services required, were the smallest at around 6 kB. Generally all the commercial offerings boasted RFC compliant implementations and as a result, the compiled size were a few orders of magnitude greater than the project goal of 2kB, typically 15 to 64kB and varied greatly between 8, 16 & 32 bit implementations.

Further, the commercial offerings gave the impression that they focused on differing markets, and as such it was evident that perhaps there is no interest in development of a "micro" TCP-IP stack. Only time will tell if demand results in a product of this

nature entering the main-stream market.

## **Chapter 3**

# **Embedded Platform and Development Tools**

### **3.1 Chapter Introduction**

This chapter details the background project requirements for a selection and development tools for development of a working prototype. The working prototype was required to be selected and code developed to implement the IP services required, namely PPP, IP, UDP, TCP, DHCP and HTTP. The following selection describes the selection process of the end core and development tools.

### **3.2 Microprocessor Core**

A range of different cores were researched and it was found that a full year of research could be consumed in this process alone. On offer were numerous cores ranging from 8 to 32 bit with a great range of peripherals, including Analogue to Digital Converter (ADC), Timers, comparators, not to mention, flash RAM from kilobytes through to Mega-bytes.

Due to the overwhelming range of cores on offer the evaluation process converged to

a practical process which was further justified by the ever increasing consumption of time. This selection process was therefore judged on what was readily available, had solid tools for support, both for programming and code development, and was well supported by industry.

It is clear that the use of a flash RAM device would be beneficial during the development cycle as it could be programmed easily and often. The device therefore would need to have the capability of “In-System Programming” (ISP), where the micro-controller would not be required to be removed from the development circuit to be reprogrammed, further reducing the programming development cycle. Further, with many of the micro-controllers offered only as surface mounted devices, removal from the development board was not practical for reprogramming.

Two manufacturers were short listed based on their availability and wide industry use, the Microchip “PIC” (Peripheral Interface Controller) <http://www.microchip.com> and ATmels AVR core <http://www.atmel.com/products/avr/>. Both of these devices boast RISC architecture instruction sets and are 8 bit devices. These devices are available in a wide range of memory configurations and an equally wide range of on-device peripherals.

### 3.3 Software and Tools

To assist the software development, the use of higher level languages was required as the development of TCP/IP stack would be a challenging task if written in assembly language. Further, the use of assemble language would incur a further learning period to become familiar with the selected platform instructions and its idiosyncrasies. In-line with the project goals is the use of such a language that can provide software portability, as much as practicable in an embedded system. The use of assembly language would therefore diverge from the project goal of providing portability across varying manufactures.

The author has had experience in a number of programming languages during his studies, including Java, Basic, Delphi (Pascal) and 'C'. The 'C' language was chosen

due to its support for virtually all offered IP implementations together with its strong support within the embedded industry and the authors recent exposure within his studies. With these considerations in mind the chosen core would need to have a strong 'C' language compiler support and if possible an integrate development environment in which the core programming and software management functions could be contained as one interface, further reducing the "learning curve".

### **3.4 Chosen System: Hardware**

With the above characteristic in mind the AVR core was selected from the Atmel range of micro-controller devices. Due the large support of these devices, the range of available devices within the AVR range (memory, ADC, Timer etc) and support from numerous compilers, both commercial and open source, this device was an easy choice. Further, from the authors previous experience with the Microchip range of cores, the banked memory architecture has lead to difficulties with programming as the memory is split into banks which do not allow addressing of the complete ROM without setting a bank flag or switch. Microchip have recently improved the architectural design of the memory access problem but backward compatibility may be a problem for this design for portability reasons. The Atmel range, utilising the AVR core, has an identical architecture over devices ranging from 1 to 256 kB of ROM and 64 B to 8 kB of RAM, which allows a wide range from which to choose a hardware platform for the end product target.

As with both manufactures, the devices can be easily programmed in-circuit and the Atmel AVR core boasts more than 10,000 re-write/erasure cycles. This provides a very flexible development system which can easily be ported to a manufacturing environment with relative ease.

Due to project time constraints a simple development system was procured from "JED Microprocessor Systems" which satisfied the requirements of the previous evaluation requirements. The AVR570 ATmega128 CPU module and AVR572 prototype development board were purchased as well as an ISP (In-System Programmer).

The ATmega128 device boasts 128kB of FLASH ROM and 4kB of RAM, which is more than adequate for this development and was the only device offered in the convenient development platform. This was not considered a problem as the Atmel AVR range of 8 bit devices can easily be scaled onto a device with limited memory and peripherals, if required. The development hardware is revealed in Figure 3.1 and Figure 3.2.

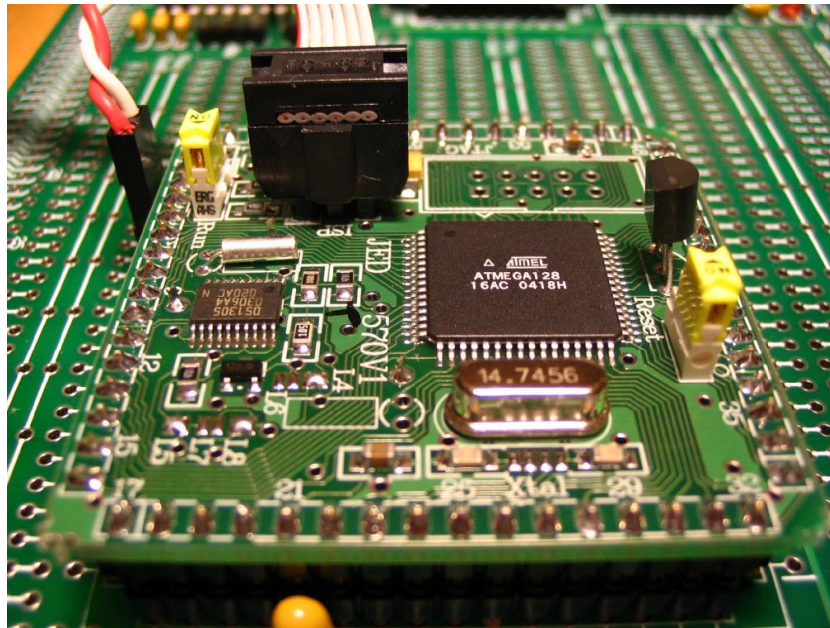


Figure 3.1: The AVR570 module from “JED Microprocessor”. Note the In-System Programmer interface (top right), reset switch in yellow (bottom left) and the 64 pin surface mounted microcontroller (centre).

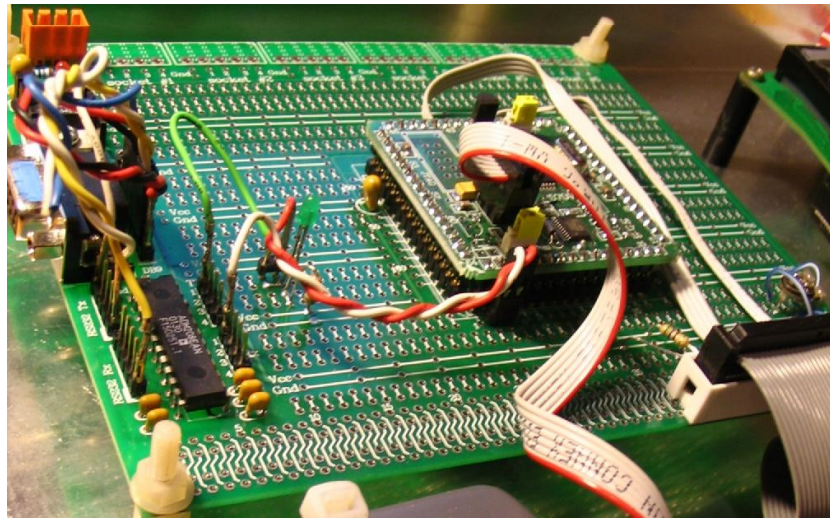


Figure 3.2: The AVR572 development board from “JED Microprocessor” revealing the AVR570 module and the RS232 transceiver and DB-9 connector (middle left)

### 3.5 Chosen System: Software

'CodeVisionAVR' by HP InfoTech, <http://www.hpinfotech.ro/>, was selected because of its wide industry use and strong user support. This compiler supports the 'C' language and provides a user friendly Integrated Development Environment (IDE). The IDE provides project management, including file management, as well as an interface to the Atmel In-System Programmer. Figure 3.3 reveals a screen snapshot of the main interface.

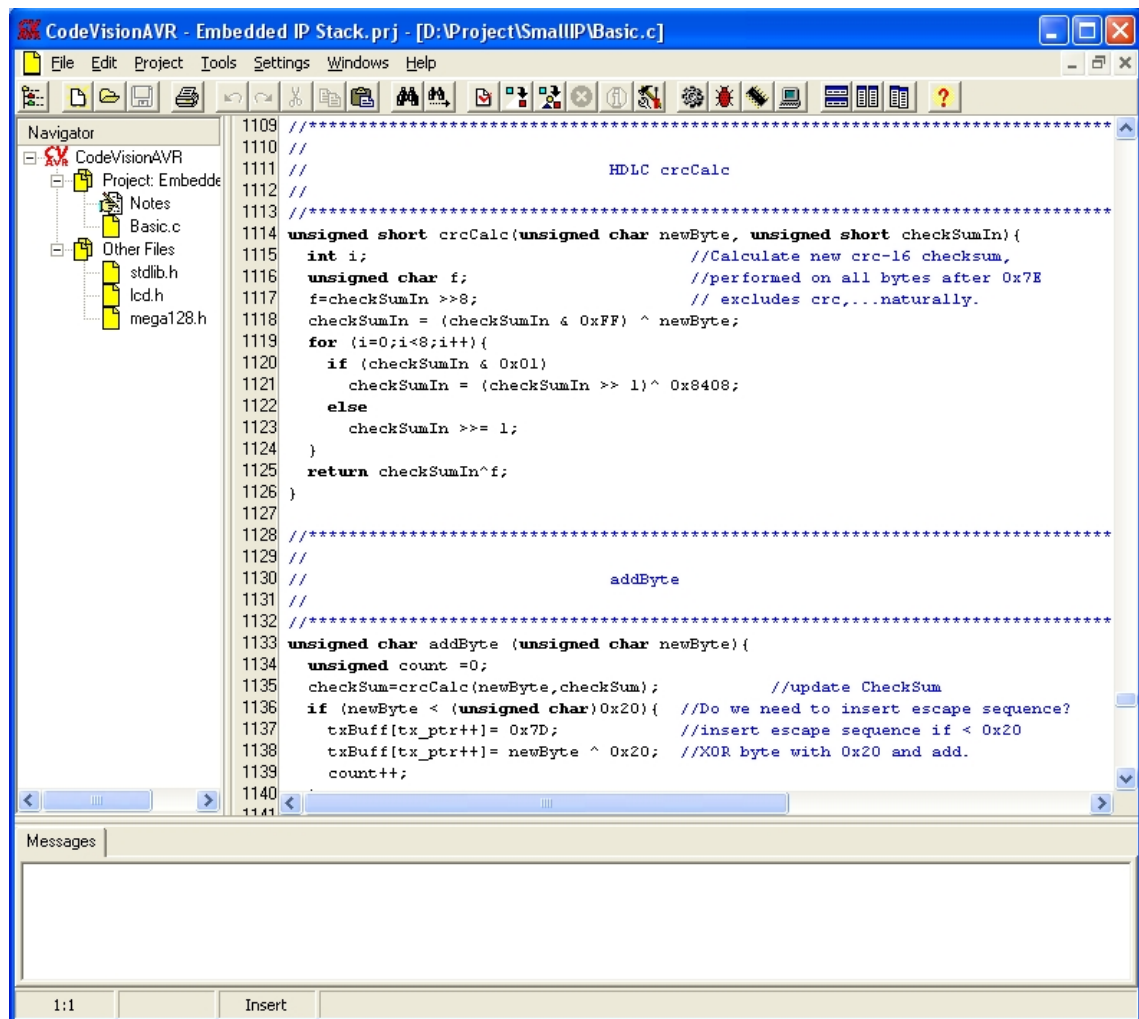


Figure 3.3: A screen snapshot of the ‘AVRCodeVision’ IDE. File management facilities are revealed on the (left) sub-window with the main coding area (right). Convenient buttons are available for various functions including, saving, compiling, searching and target programming are located beneath the pull-down menus. A status message window is available for debugging and monitoring purposes (bottom)

AVRCodeVision provides a range of features in which the development of embedded software is enhanced. The IDE provides many features including compiler optimisations, insertion of assembly within the ‘C’ code, supports the AVR core of the Atmel devices, built in code wizard and code completion tool, as well as supplementary libraries supporting external peripherals such as LCD (Liquid Crystal Displays), RTCs (Real Time Clocks), temperature sensors and various signalling protocols (I<sup>2</sup>C, Dallas



1 wire and SPI). The IDE also supports in-system programming with a simple 1 click compile and program function.

More detailed programming, including the configuration and set-up of the device can also be made via the programmer interface, revealed in Figure 3.4. The “Chip Programmer” provide a means to which the device’s contents may be read or written to a file, in addition the set-up of the security configuration can be made to “lock” the device to ensure the contents of the program can not be copied or overwritten.

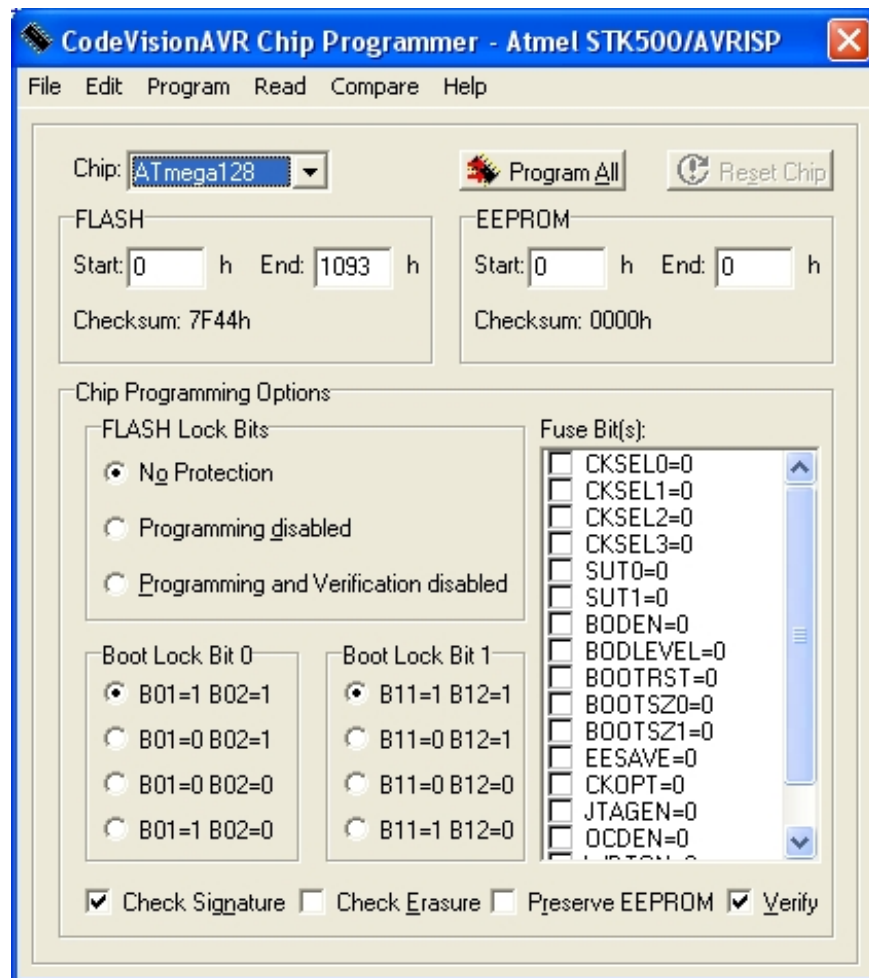


Figure 3.4: AVRCodeVision Device Programmer Interface.

---

## 3.6 Chapter Summary

This chapter has evaluated a range of hardware platforms and software development tools. This evaluation phase has established a baseline from which the development of a working prototype can be made. The process has successfully evaluated and selected an end core, namely the Atmel AVR, to which a portable software implementation can be made. The use of the AVRCodeVision IDE will provide the necessary software and hardware tools to foster the software development. The use of 'C' as a programming language will provide the portability due to its strong industry support, not only within the Atmel AVR range but also across a large range of end hardware platforms.

## Chapter 4

# Software Development

### 4.1 Chapter Introduction

In this chapter I will describe the software implementation of the Internet Protocols with an aim to provide the services within a 2kB envelope. This is certainly a challenging task to squeeze this into a small code size. It was discovered early in the project that to meet this objective certain compromises would need to be made to minimise the code and as such is main technical challenge of this Research Project.

The Internet Protocols implemented, namely PPP, IP, TCP and HTTP, will be described. Firstly an overview of the generic framework will be presented from which the IP software is built upon. The TCP/IP protocols required to serve a simple web-page will be discussed, beginning with the Host to Network layer (RS232 & PPP) through to the Application layer (HTTP).

### 4.2 Software Overview

The underlying software structure provides device initialisation, modem emulation, packet reception, packet transmission and link termination. The system was developed with Windows 98 PPP client, a terminal application Comlab (Vanstan 2003) and a

---

popular packet analyser Ethereal (Combs 2005).

After reset the device is initialised. The initialisation routine configures the ports and importantly the communication port to which the device communicates. The software then provides responses to modem commands issued by the PC as the device is directly connected to the PC's serial port. When the fundamental data link has been established the software waits for a data stream.

When a data stream is received and stored it is inspected. The inspection process determines what course of action it should take. If the packet is unrecognised or one of which it is not interested in it is silently discarded. If a response is required a packet is generated and sent via the communication port. A simplified flow diagram of the software structure is revealed in Figure 4.1.

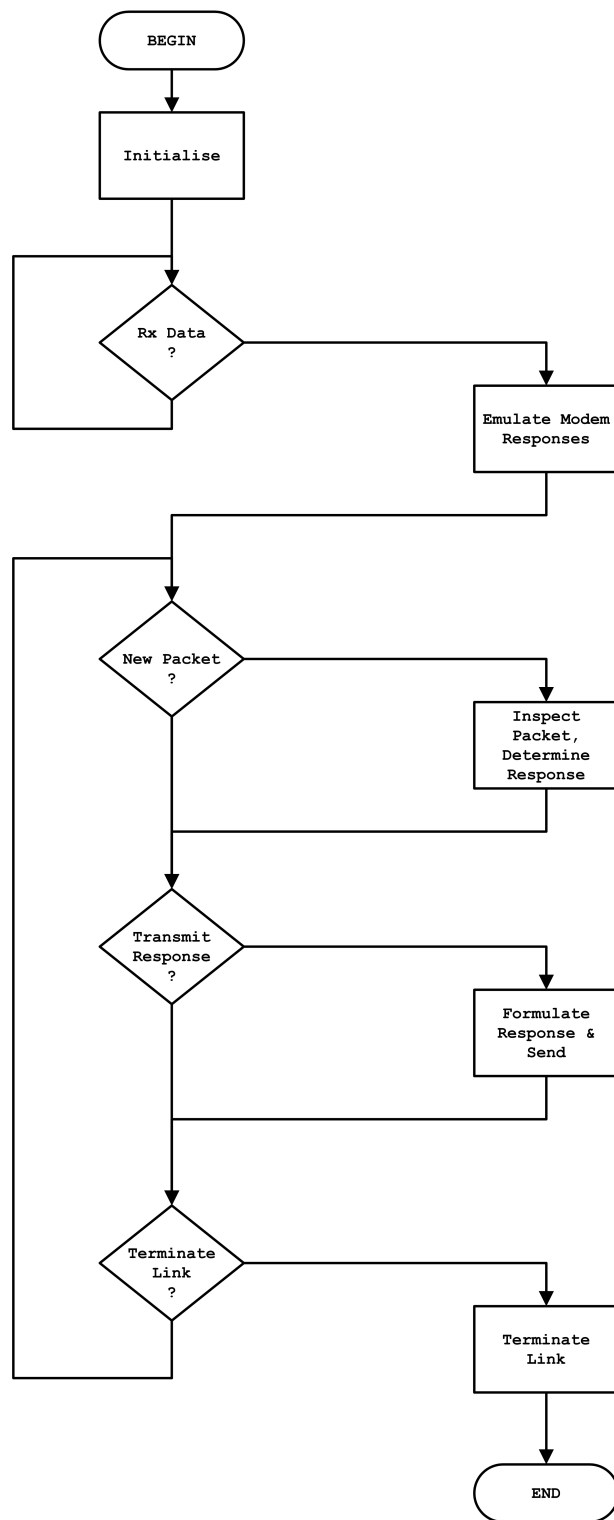


Figure 4.1: Simplified Flow Diagram.

A number of supporting functions are called during this process. They include reception and transmission routines, various checksum calculations, insertion of escape sequences and packet inspection functions.

### 4.3 High-level Data Link Control

The basis for all communication within the Point to Point Protocol is via the HDLC data link control protocol. In this implementation the HDLC is used to encapsulate the higher level protocols over an EIA232 link. In this case the HDLC protocol operates over an asynchronous link and includes error detection via a Frame Check Sequence. Figure 4.2 reveals the HDLC frame (Kozierok 2003-2005).

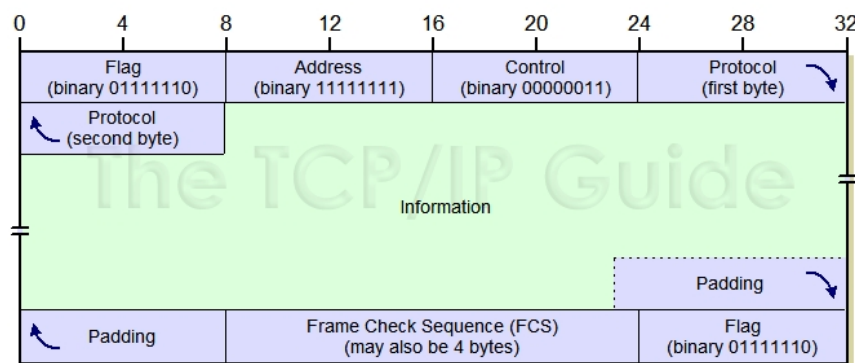


Figure 4.2: PPP General Frame Format, utilising HDLC.

With PPP the first three bytes are fixed in value. They are the Flag, Address and Control fields, and are fixed as the link only ever connects between two hosts. It is important to note that PPP doesn't conform to HDLC rules, it uses HDLC as the basis for its structure. The next field (protocol) specifies the protocol of the information payload. In this implementation the protocol fields of interest are LCP, PAP, IPCP, and IP. The higher level protocols, TCP and HTTP, are contained within the IP information payload.

The start and end of each packet is signified with 7E hexadecimal (01111110 Binary) and provides a means to detect the packet. As mentioned previously the HDLC packet utilises error detection using a FCS. This is calculated over the entire frame, excluding

the start and ending Flags, and is positioned at the end of the frame. The position of the FCS is convenient as it is calculated as the packet is being transmitted. The calculation of IP and TCP checksum is a little more tricky as the checksums are positioned within the headers, and as such the checksums are required to be calculated prior to passing them to the lower level protocols for subsequent transmission.

The PPP packet is then transmitted with the start Flag, Address, Control, protocol field, information payload, FCS and end Flag. However any byte within the frame that is less than 20 hexadecimal or is 7E or 7D hexadecimal is treated differently. If such a byte is found within the stream the byte is first exclusive-ORed with 20 hexadecimal and transmitted with a 7D hexadecimal followed by the exclusive-ORed byte.

In this implementation the PPP frame is generated in a buffer, the FCS calculated and then the buffer is sent to a transmission routine where the bytes are scanned to see if they match the above criteria (< 20, 7E or 7D Hexadecimal) and sent.

## 4.4 Point to Point Protocol

PPP is a protocol that enables links over a variety of different physical layer connections. In this implementation it is being utilised over EIA232, formally known as RS232, however it can also operate over other mediums, such as USB and ethernet. It can carry any type of network layer datagram, however we are only interested in the IP and UDP protocols in this implementation.

PPP has taken over from the Serial Line Internet Protocol (SLIP) mainly due to the former having security features such as authentication, the ability to support other protocols other than IP and ability to dynamically allocate IP addresses during the link establishment phase.

As we have briefly discussed in the preceding section, PPP operates using a HDLC-like data link control protocol and provides a means to which two hosts can communicate with Internet Protocols. However, before we can authenticate the user and exchange data over the link, we must configure it. This is achieved within this implementation

via the Link Control Protocol, the Password Authentication Protocol and the Network Control Protocol.

#### 4.4.1 Link Control Program

This protocol is used to establish, configure and test the data-link. Firstly the PPP link sends LCP packets to configure and test the data link. After each side of the link has agreed to its peers configuration the basic link is established.

The protocol uses a system of requests, acknowledgements, negative-acknowledgements to negotiate the options. However there are also many other configuration controls, reference should be made to RFC1661 (Simpson 1994b) for full implementation details. This implementation provides the minimum negotiations required to establish the protocol and is far from RFC compliance. This is wholly justified by the fact that the project goal is to provide a very small IP implementation. The implementation of an RFC compliant design would therefore consume a large envelope of code diverging from the project goal.

There are a number of configuration items that can be negotiated. In this implementation configuration of the minimum number of options to establish a link was made. The minimum options for the peer are 2, 5, 7 and 8, whereas for the host (embedded device) options 3, 5, 7 and 8 were required. The options of interest are revealed in the Table 4.1. It was found that the link could not be established, with the Windows 98 client, with any less than the options listed. Any attempt to reduce the options resulted in the peer not agreeing and hence terminating the link.

Table 4.1: LCP Configuration Options

Option	Description
2	Maximum Receive Unit
3	Authentication-Protocol.
5	Magic Number
7	Protocol-Field-Compression
8	Address-and-Control-Field-Compression.



The LCP configuration option 2 signifies that the host can receive larger packets, or to request that the peer send smaller packets.

The default value is 1500 octets which seems to indicate it matches the maximum size of an ethernet frame. However it was noticed during development that even if this option was reduced, to say 200 bytes, the peer ignored the request and continued to send larger packets. Inspection of RFC1661 revealed the following “If smaller packets are requested, an implementation MUST still be able to receive the full 1500 octet information field in case link synchronisation is lost.”, however it didn’t seem to have any bearing if synchronisation was not lost.

Option 3 of the LCP configuration specifies the authentication protocol. This may be either the Password Authentication Protocol (PAP) or Challenge Handshake Authentication Protocol (CHAP). The PAP authentication type was selected and negotiated as it was simple to implement, whereas the CHAP requires a considerable overhead in terms of software relative to PAP. PAP simply provides a mechanism to which a UserID and Password are sent, in plain text, from the peer requesting the authentication. The host then authenticates the requester by sending an acknowledgement. Conversely if the UserID or Password are incorrect the host sends a Rejection.

The last two options 7 and 8 provide a means to which information within the HDLC frame may be omitted to improve the through-put of the link. The first three bytes of the HDLC frame, after the initiating Flag (7E), are always fixed. When this compression is in use these bytes are simply not transmitted. However, each peer may still transmit these bytes if it so desires. In this implementation these bytes are always transmitted as it reduces the code envelope.

#### 4.4.2 Password Authentication Protocol

Once the initial Link is established the authentication process begins with the negotiated protocol PAP (option 3 in the LCP negotiation). The UserID and Password pair is sent to the authenticator and if authenticated a Acknowledgement is sent.

In this implementation the software looks for a PAP packet with a REQ (Request) from

the peer and acknowledges that request. Currently the software provides no password or user ID checking.

#### 4.4.3 Network Control Protocol

The Network Control Protocol is a used to configure each network protocol which intends on utilising the data link by use of network protocol configurations. For the case of IP, the configuration protocol used is the Internet Protocol Control Protocol (IPCP). With this protocol the IP configuration is negotiated and established. Such things as, DNS, Gateway, netmasks and IP addresses can be configured with this protocol. In this implementation the IP address for both the host and the peer is made. Again, the minimum configuration was negotiated, that being the IP addressing of the host and client only.

After the negotiations have been made and agreed by both hosts the Host to Network layer is established and the link is available to accept IP traffic. In Windows the clients signifies this by popping up a window confirming you have made a connection to your host, as in Figure 4.3.



Figure 4.3: Windows ME connection established window.

## 4.5 Internet Protocol

At this stage we have successfully negotiated and established a PPP link which carry our IP datagrams, which in turn carry the higher level protocols TCP and HTTP.

The implementation of the IP packet was relatively trouble-free as once the PPP link is established it is just a matter of filling up the transmit buffer for the lower layer to send. The only difficulty was the implementation of the checksum as it is located within the header which is at the front of the datagram. Recall the checksum for the HDLC frame is at the end of the data stream and was easy to calculate as bytes were transmitted. In the IP case this involved the use of another temporary buffer in which the checksum could be calculated prior to passing it to the lower layer.

Figure 4.4 reveals the datagram format detailing the position of the checksum within the datagram.

ver	len	type	Total Length	
ident			f	offset
TTL		Protocol	Header Checksum	
Source Address				
Destination Address				
Data				

Figure 4.4: IP Frame, note the relative position of the checksum, which is within the header.

IP provides a means in getting datagrams from the source to their destination. This is achieved with addressing the datagram. As can be seen in Figure 4.4, the IP header contains fields for the IP source and destination addresses. In this implementation they are statically set however, using DHCP these could be dynamically made.

---

## 4.6 Transmission Control Protocol

The Transmission Control Protocol (TCP) proved to be the most challenging aspect of the software development. As TCP is a connection orientated protocol it provides mechanisms to ensure that data sent over the link reaches its destination. This involves the establishment of a logical connection, the passing of data and an acknowledgement that the data is received. In addition each unique data transfer is handled in this set-up and acknowledge method separately.

TCP first requires a connection to be established and in the case of a client requesting a web-page the client first sends a SYN to begin the process. The server in this case also sends a SYN and an ACK to acknowledge the SYN. TCP provides a means to which ACKs can be piggybacked to reduce traffic, as in the case of a SYN and ACK in one response. Once this initial handshaking has taken place the client requests a web-page with the HTTP GET command and also piggybacks the ACK to the previous server response. The server then replies with a HTTP response, in this implementation in the form of a HTTP/1.0 ... "Hello World", with an ACK to the previous segment to signify it has received its request. Again the client sends an ACK to acknowledge the reception of the webpage, and the server sends another ACK to acknowledge the clients ACK. At this stage the web-page can be displayed. The webpage is only displayed when the client knows the transfer has been received in order and that it is all it should receive. Figure 4.5 (Bentham 2003) reveals this process described, note the use of piggy-backing to reduce network traffic.

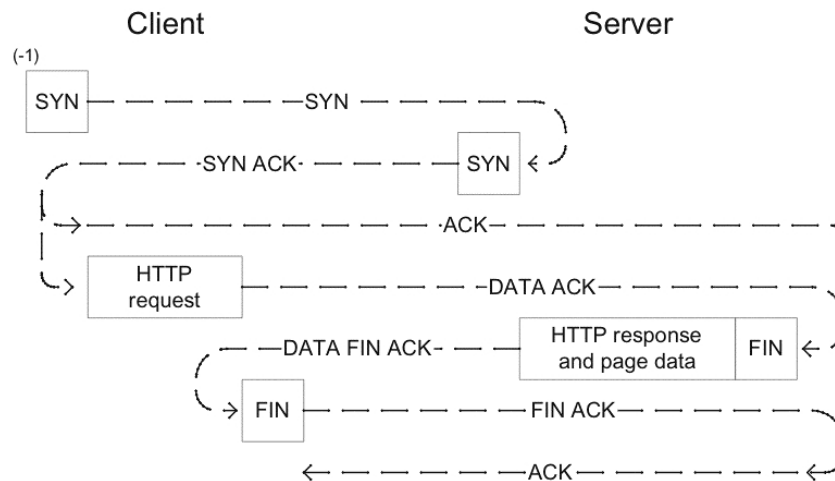


Figure 4.5: TCP segment data transfer.

In addition to the handshaking taking place TCP provides a mechanism to ensure that the segments transmitted are sequenced. As packets can be received out of order each segment transmits a 32 bit sequence number which identifies the start of each segment. When the peer acknowledges the segment it also acknowledges the sequence number which is incremented by the number of bytes received in the TCP datagram. Subsequent segments transmitted increment the sequence number by the size of payload so that the receiver can determine the order in which the data should be placed in. In this implementation the segment size has been kept small, less than 500 bytes, to reduce the software management if datagrams where split over multiple packets.

TCP implements the sliding window method of data transfer to improve link efficiency which is also another function of the sequence numbering. As the embedded device has limited RAM, the window size has been limited to ensure that it only ever needs to keep track of one segment (one packet). This has the secondary advantage of reducing the software complexity and hence code size.

The second challenge of TCP was the calculation of the checksum. In TCP the checksum is not only is calculated over its header and payload, as a conventional checksum is calculated, but also included is information from the IP layer. This provides additional measures to ensure the TCP segment is directed to its intended recipient. It was interesting to note that this violates the architectural layering principles of the OSI model

in which each layer is a self contained unit.

The TCP checksum is calculated with a “Pseudo Header” and included within the Pseudo Header is IP source and destination addresses, the IP protocol field (from the lower IP layer) and the computed TCP length.

This presented some difficulties with the calculation as the TCP and IP layers were closely bound. The software implementation therefore, was closely bound in the way it operated as such we had two checksum to calculate prior to passing this down to the PPP link for transmission.

## 4.7 Hyper-text Transfer Protocol

The application level protocol used to send and receive web-pages is the Hyper-text Transfer Protocol. It provides a simple method of requesting web-pages from servers and the format of the data from the server. In this implementation the device looks for the “GET” ASCII byte sequence within the TCP payload. The device then reply to this request with a simple response, in this case a static web-page is displayed. Figure 4.6 reveals the first web-page served on the embedded device.

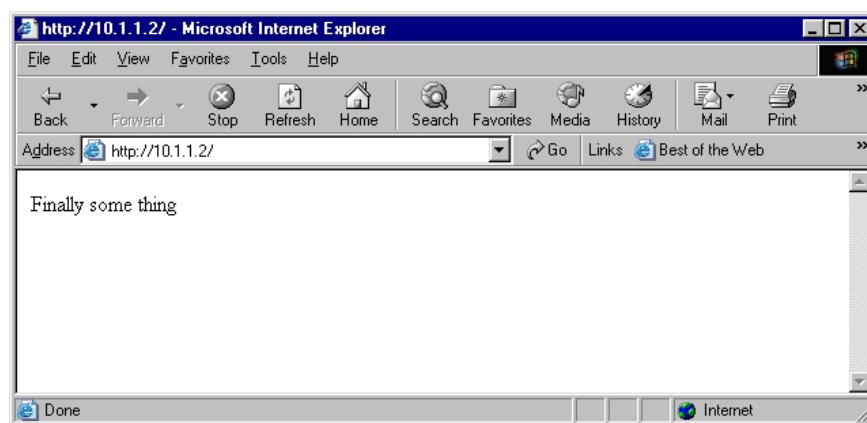


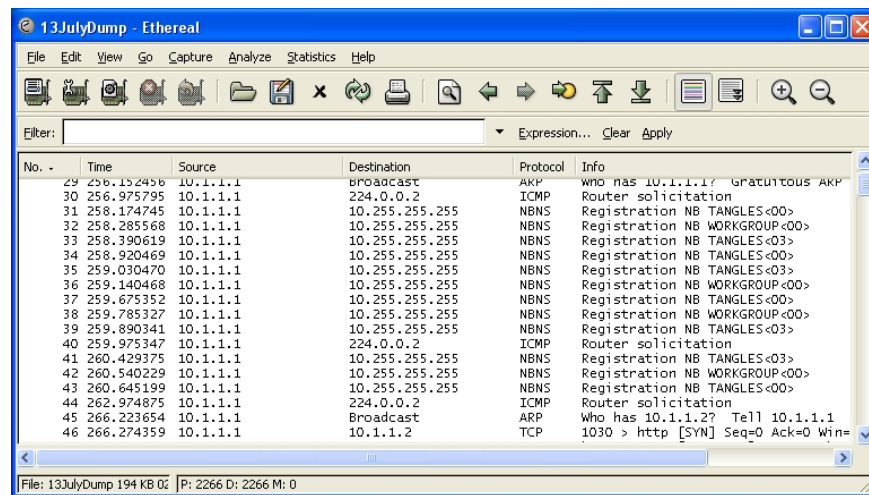
Figure 4.6: Browser Display of Served Web-Page from the embedded device.

The implementation of this protocol was straightforward and only required the testing of the TCP datagram, for a GET and the simple formulation of a static web-page which

was passed down to the TCP layer.

## 4.8 RAM Utilisation

Currently the software utilises approximately 3kB of RAM. This is made up of 2 buffers of 1kB for transmission and reception, in addition to a temporary buffer of 768 Bytes plus the stack of approximately 240 Bytes. The transmit and temporary can be reduced in size to around 150 Bytes. However the receive buffer needs to be larger as it was found that some packets can cause a buffer over-run. This was caused by the plethora of packets received once the link was established, and also as it is idle, and is revealed in Figure 4.7.



No.	Time	Source	Destination	Protocol	Info
29	256.152456	10.1.1.1	Broadcast	ARP	who has 10.1.1.1? Gratuitous ARP
30	256.975795	10.1.1.1	224.0.0.2	ICMP	Router solicitation
31	258.174745	10.1.1.1	10.255.255.255	NBNS	Registration NB TANGLES<00>
32	258.285568	10.1.1.1	10.255.255.255	NBNS	Registration NB WORKGROUP<00>
33	258.390619	10.1.1.1	10.255.255.255	NBNS	Registration NB TANGLES<03>
34	258.920469	10.1.1.1	10.255.255.255	NBNS	Registration NB TANGLES<00>
35	259.030470	10.1.1.1	10.255.255.255	NBNS	Registration NB TANGLES<03>
36	259.140468	10.1.1.1	10.255.255.255	NBNS	Registration NB WORKGROUP<00>
37	259.675352	10.1.1.1	10.255.255.255	NBNS	Registration NB TANGLES<00>
38	259.785327	10.1.1.1	10.255.255.255	NBNS	Registration NB WORKGROUP<00>
39	259.890341	10.1.1.1	10.255.255.255	NBNS	Registration NB TANGLES<03>
40	259.975347	10.1.1.1	224.0.0.2	ICMP	Router solicitation
41	260.429375	10.1.1.1	10.255.255.255	NBNS	Registration NB TANGLES<03>
42	260.540229	10.1.1.1	10.255.255.255	NBNS	Registration NB WORKGROUP<00>
43	260.645199	10.1.1.1	10.255.255.255	NBNS	Registration NB TANGLES<00>
44	262.974875	10.1.1.1	224.0.0.2	ICMP	Router solicitation
45	266.223654	10.1.1.1	Broadcast	ARP	Who has 10.1.1.2? Tell 10.1.1.1
46	266.274359	10.1.1.1	10.1.1.2	TCP	1030 > http [SYN] Seq=0 Ack=0 Win=

Figure 4.7: Plethora of packets received on establishment of Link.

It is expected that the RAM utilisation can be easily reduced to around 1 kB (150 B for transmit and temporary buffers, 500 B for receive buffer and 200 B for the stack) which would be better suited to smaller devices.

---

## 4.9 Code Optimisation

Currently the prototype systems occupies a code envelope of 8 kB. This is discussed in more detail, including further work, in Chapter 6.

## 4.10 Chapter Summary

This chapter has discussed the challenges encountered during this research project and some of the difficulties working with an embedded system utilising the Internet Protocols. It was interesting to discover that the Network (IP) and Transport layers (TCP) were closely bound and how it presented implementation issues with the Pseudo Header checksum calculation.



## Chapter 5

# Testing and Security

### 5.1 Chapter Introduction

This chapter introduces the system testing activities performed so far. The security measures implemented and security considerations which are important to embedded devices are also discussed. As the project has been primarily been focused on the software implementation some hardware testing is discussed.

### 5.2 Testing So Far

The testing of this system has evolved through-out the project in line with the services which were required for each project steps. The project began with the initial set-up and testing of the basic EIA232 communication, the PPP link, then the IP and finally the TCP layers to provide a working prototype.

The first step was the establishment of the basic EIA232 communication. This was a simple task as 'C' provided functions to send and receive bytes through the USART. The AVRCodeVision IDE code wizard was used to set-up the communication parameters of the port and a simple program was written to echo the characters send from a PC.

The first challenge of the project was the implementation of the HDLC packet which was fundamental to the operation of the TCP/IP stack as it provides the data link between systems. The formulation of the HDLC was relatively easy however the calculation of the Checksum provided some challenges. The packet analyser 'Ethereal' was found to be invaluable in the debugging process to provide feedback that the correct HDLC packet was formed and sent with the correct checksum. Various packets were formulated and sent to and from the embedded system to test the operation. This testing, with the assistance of 'Ethereal', proved that the software algorithms were functional correctly including the reception and transmission of the correct escape sequences (7E, 7D, and anything below 20 hex).

The process of testing with ethereal was extended to testing of the IP and TCP protocols. It is interesting to note that the way in which the TCP/IP suite operates. If you do not have the correct structure in your packets it just ignores them and 'Ethereal' again was used to debug malformed packets. For interest Figure 5.1 reveals a screen shot of 'Ethereal' capturing a malformed TCP checksum.

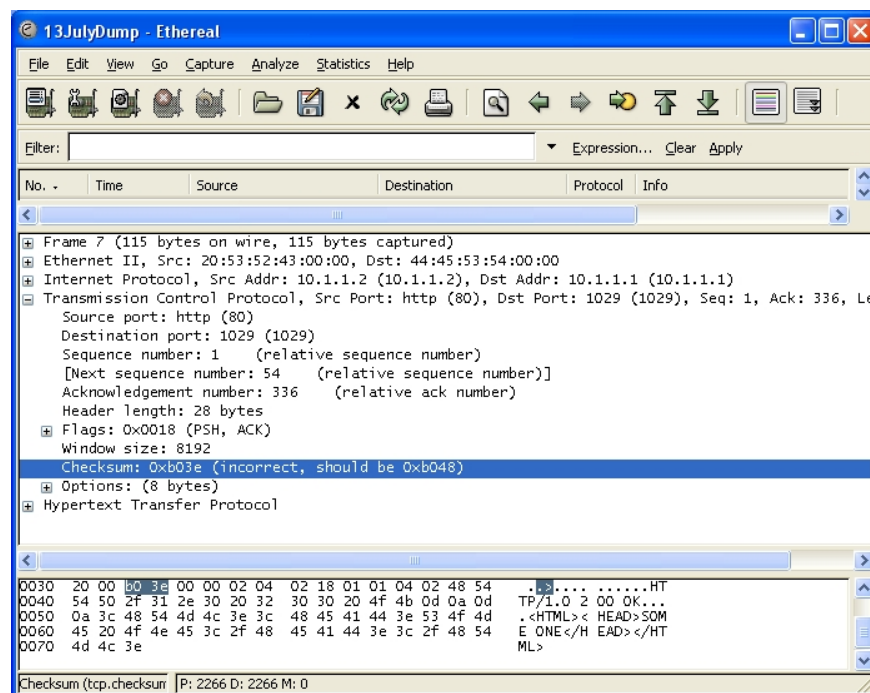


Figure 5.1: Screen shot of Ethereal Packet Analyser revealing a malformed checksum calculation.

Final testing of the system was performed with the system in its expected software environment. Windows 98 was used as the PPP client and Microsoft Internet Explorer as the browser. Further testing is required to prove its compatibility with differing PPP clients and web-browsers. However, as HTTP is platform independent no issues are expected. Further testing with Windows XP, and also a Linux client will be required to provide further confidence.

The EIA232 interface was initially selected to operate at 9600 baud as a GSM radio modem was available. However, it is necessary to test the system at higher baud speeds to test the softwares handling of communications.

### 5.3 Security

The prototype system utilises Password Authentication Protocol as its method of authentication. However the implementation so far does not provide any user or password checking and simply allows any user to establish a connection. This is obviously unacceptable and hence will require implementation of user and password checking. The other consideration is where that user may input executable code into both the userID and password fields in an attempt to create a buffer overrun. This potentially could corrupt the system stack if the buffer spanned into that area. No testing if this has yet be provisioned.

Other aspects of this system in terms of security is that of Denial of Service attacks. These attacks are where the device is continually bombarded with IP traffic, whether it is HTTP requests, PING requests or general traffic overload. As the system only responds to a very narrow and specific type of request the only significant problem is that of TCP traffic. Currently the system is wide open in this respect. It is however unlikely that this may happen unless a virus of some form was initiating the traffic without the users knowledge. It would be unlikely that the user of the system would be responsible for this attack.

---

## 5.4 Environmental and Hardware

As this device may be used potentially in a wide variety of harsh environments testing in these environments would be required. This would include aspects such as temperature extreme testing and testing within noisy electrical environments. At this stage as the project has primarily been focused on the software implementation these hardware considerations have not been fully investigated and hence tested.

## 5.5 Chapter Summary

This chapter has discussed the various aspects of testing and how the software testing was closely bound with the software development. The layered approach to the Internet Protocols has forced the software development not to continue until the underlying protocols were functional. Some areas of testing, other than software, were discussed and highlighted the need for an engineering approach to be taken to all aspects of the system. Although this research project was wholly focused on software development considerable work is therefore still required to produce a commercial product.

## Chapter 6

# Conclusions and Further Work

### 6.1 Achievement of Project Objectives

The following objectives have been addressed:

**Research and Evaluation of Existing Systems** Chapter 2 presented and discussed the research and evaluation of existing systems available today. Details were presented of the available system and where evaluated in terms of cost, security and function. Commercial and open source products were considered in the evaluation. Investigations found that there wasn't an implementation, both commercial and open source, that satisfied the project requirements for providing PPP, IP, UDP, TCP, DHCP and HTTP services within a 2kB envelope.

The commercial offerings boasted RFC compliant implementations and as a result, the compiled size were a few orders of magnitude greater than the project goal of 2kB, whereas the open source implements, although smaller, didn't provide a PPP option. The evaluation also highlighted that the selection of an end target was critical in relation to the compiled code size. It was concluded the code size was a function of word size ( 8, 16 and 32 bit devices).

**Embedded Platform and Software Development Tools** Chapter 3 evaluated a range of hardware platforms and software development tools. The evaluation

phase has established a baseline from which the development of a working prototype was made. The successful selection of an end core and programming language, namely the Atmel AVR and 'C' respectively, to which a portable software implementation was made. The use of the selected platform provided portability between the Atmel range of AVR core devices and also others manufactures due to the widely used industry supported 'C' programming language.

**Construction of a Working Prototype** Chapter 4 detailed the software implementation of the Internet Protocols. The protocols PPP, IP, TCP and HTTP were successfully implemented and provided a working prototype. However the UDP and DHCP protocols were not implemented as the it was found that within the Network Configuration Protocol the allocation of IP address was made. This does not indicate that these protocols are not required, only that they were not required for this implementation utilising PPP. Implementation of these protocols would certainly be useful if the device were interfaced via another medium such as Ethernet or within a GPRS network for self negotiation and allocation of network addresses.

## 6.2 Further Work

The majority of the objectives of this project have been made however further work is required to tidy up the software, optimise and enhance its data structures and RAM usage and to improve its security.

The first and most obvious work required is to optimise the software structures to reduce the code envelope. During the early parts of this research project it was understood that some of the algorithms would need to be optimised in assembly to reduce the code envelope. However as the project has matured and time has been available to reflect back on the initial objective I feel that it is more important, in both a commercial and technical sense, to ensure the system is portable.

As was discovered in the evaluation phase the selection of the end core had larger repercussions for the end code envelope size. If for example a larger device, such a

16bit type, was selected this would potentially made the prototype code smaller. Now, if the software was then ported to a 8 bit device the compiled code size would increase, as expected and revealed in the evaluation phase. This leads me to conclude that software portability is more important than striving for the 2 kB goal. Yes, I believe I could come close to this goal if I optimised the system in assembly but the system will be inflexible and platform dependant.

The testing of the system at higher baud rates to assess the simple communication method used is also of importance. To enable portability I did not utilise any interrupt driven communications which certainly may reduce the devices top end communication speed. Empirical testing of the device to establish this limitation is still required.

Another important consideration for further work is that of security. Security takes many forms which all need consideration. For example the process of Password Authentication provides a level of security for access to the system, but at the same time does this pose a security risk if the input buffer is over-run with executable code that corrupts the stack? Other considerations that need further research is malicious attacks from external parties in the form of Denial of Service attacks and what measures can be taken to combat this. Consideration must also be made with respect to what access the device provides if a users wishes to use the device as a stepping stone for attacks of another computer systems. Currently the risk is considered very low as this implementation provides a “skeleton” TCP/IP stack. However, further development of the system will warrant these careful considerations, especially in a commercial environment.

The implementation of UDP and DHCP protocols for self configuration within a network would certainly prove to be an excellent feature. This feature would be very convenient for integration into ethernet networks for monitoring and control purposes. I can also see a trend to use the GPRS as the transport mechanisms for remote devices where monitoring is required without the hassles of dialing a device to access it <sup>1</sup>.

Other areas of development for this project may extend to some datalogging functions. It would be convenient to utilise the File Transfer Protocol (FTP) for this function

---

<sup>1</sup>The authors knowledge in this area is limited however, it would seem that the functions of DHCP would also be advantageous to provide a “Plug-n-Play” monitoring/control device

and was the motivation for including it as a “As time permits” option in the Project Specification.

The addition of dynamic web-pages, including such things as colour, buttons or even graphics, would make the interface more informative and in-line with User expectations. Another area of research could be the addition of a small mail client to send email providing monitoring and status. Further research into this area would be required to understand the mechanisms that govern small embedded web-servers, be that memory (RAM and ROM limitation) or processing power. It also leads the question of whether the device would need an operating system to manage all these added features <sup>2</sup>.

## 6.3 Conclusion

Technology utilising the Internet protocols is an exciting and fast developing area of Engineering. This project has explored the available technology and found it lacking in the area of small embedded software. It has captured the flexible nature of this technology, via the simple to use web-browser tools, which will enable its ease of use from a users point of view.

This product has a real opportunity to fill a market and the uses are limited only by your imagination. For example, this technology can be incorporated into existing networking hardware for monitoring, deployed via radio link technology in the field or even in a smart home application.

The project has excited me personally in the possibilities and opportunities still available within the Engineering field, both in Electronic and Software Engineering.

---

<sup>2</sup>Hello, any sponsors out there!



# References

B. Lloyd, W. S. (1992), *RFC 1334 - PPP Authentication Protocols*, Network Working Group.

Bentham, J. (2003), *TCP/IP Lean*, Vol. 2, second edn, CMP Books, Lawrence, Kansas.  
<http://www.iosoft.co.uk>.

Campbell, J. (1993), *C Programmers Guide to Serial Communications*, second edn, Sams Pub, Indianapolis, Ind.

*ChipWeb* (collected March 2005), Iosoft, Cambridge, UK.  
<http://www.iosoft.co.uk>.

*CMX MicroNet* (collected Febuary 2005), CMX Systems, Jacksonville FL, USA 32223.  
<http://www.cmx.com>.

Combs, G. (2005), *Ethereal*, Open Source.  
<http://www.ethereal.com>.

Droms, R. (1997), *RFC 2131 - Dynamic Host Configuration Protocol*, Network Working Group.

*Dynamic C with TCP/IP Libraries* (collected March 2005), Rabbit Semiconductor.  
<http://www.rabbitsemiconductor.com>.

*Embedded Computer Design, Trade Journal* (2005), John Black, Michael Hopper, Wayne Kristoff, 13253 La Montana, Suite 207 Fountain Hills, AZ 85268.  
<http://www.embedded-computing.com>.

*Embedded TCP/IP Stack* (collected March 2005), Unicoi Systems, Atlanta, GA 30040 USA.

<http://www.unicoi.com>.

*ESF TCP/IP* (collected March 2005), EmIment micosystems, Portland Oregon, USA 97202.

<http://www.eminentmicro.com>.

*Ethernut* (collected Febuary 2005), Egnite Software GmbH, 44575 Castrop-Rauxel, Germany.

<http://www.egnite.de>.

*Fusion RTOS, Fusion Net and Fusion HTTP* (collected March 2005), Clarinox Technologies, Sandringham Melbourne, Australia.

<http://www.clarinox.com>.

Gadre, D. V. (2000), *Programming and customizing the AVR microcontroller*, McGraw-Hill, CMP Books, New York.

*INterNiche Lite TCP/IP and Portable PPP* (collected March 2005), Intertniche Technologies, Cambell, CA 95008 USA.

<http://www.iniche.com>.

*IPv6 Stack* (collected March 2005), Acmet Technologies Private Limited, Boxborough, MA 01719.

<http://www.acmet.com>.

Kozierok, C. M. (2003-2005), *The TCP/IP Guide*.

<http://www.tcpiptide.com>.

*KwikNet* (collected March 2005), Kadak Products, Vancouver, BC, Canada.

<http://www.kadak.com>.

Loewen, M. (2002), *Using PICmicro MCUs to Connect to Internet via PPP*, Microchip Technology Inc, 2355 West Chandler Blvd, Chandler, AZ 85224-6199.

<http://www.microchip.com>.

McGregor, G. (1992), *RFC 1332 - The PPP Internet Protocol Control Protocol (IPCP)*, Network Working Group.

*Nexgen IP and WEB* (collected March 2005), Nexgen, Charville, France.

<http://www.nexgen-software.com>.

*PC/104 Embedded Solutions, Trade Journal* (2005), John Black, Michael Hopper, Wayne Kristoff, 13253 La Montana, Suite 207 Fountain Hills, AZ 85268.

<http://www.PC104online.com>.

Perkins, D. (1993), *RFC 1547 - Requirements for an Internet Standard Point-to-Point Protocol*, Network Working Group.

R. Fielding, J. Gettys, J. M. H. F. L. M. P. L. T. B.-L. (1999), *RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1*, Network Working Group.

*RFC0793: Transmission Control Protocol* (1981), University of Southern California.

*RFC 791 - Internet Protocol* (1981), University of Southern California.

Richard Barnett, Larry OCull, S. C. (2003), *Embedded C programming and the Atmel AVR*, Delmar, Clifton Park, NY.

*RomPager* (collected March 2005), Allero Software Development Corporation, Boxborough, MA, USA 01719.

<http://www.allegrosoft.com>.

*RTIP* (collected March 2005), Segger Microcontroller Systems, Hinrich- Herzt, Germany.

<http://www.segger.com>.

*RTXC Quadnet TCP/IP v4/v6* (collected March 2005), Quadros Systems, Huston TX USA.

<http://www.quadros.com>.

Simpson, W. (1993), *RFC 1549 - PPP in HDLC Framing*, Network Working Group.

Simpson, W. (1994a), *RFC 1570 - PPP LCP Extensions*, Network Working Group.

Simpson, W. (1994b), *RFC 1661 - The Point-to-Point Protocol (PPP)*, Network Working Group.

- Target TCP* (collected March 2005), Blunk Microsystems, San Jose, CA, USA 95120-4558.  
<http://www.blunkmicro.com>.
- TCP-IP Stack* (collected March 2005), EBSnet, Groton, MA, USA 01450.  
<http://www.ebsnetinc.com>.
- TCP/IP Suite* (collected March 2005a), Interpeak AB, Stockholm, Sweden.  
<http://www.interpeak.com>.
- TCP/IP Suite* (collected March 2005b), On Time Software, Groton, MA USA.  
<http://www.on-time.com>.
- TCP/IP Suite* (collected March 2005c), QNX Software Systems, Ontario, Canada.  
<http://www.qnx.com>.
- TINI- Tiny InterNet Interfaces* (collected March 2005), Maxim, Sunnydale, CA USA.  
<http://www.maxim-ic.com>.
- Tiny TCP* (collected March 2005), Unusual research.  
<http://www.unusualresearch.com>.
- Treck IPv4/v6 Dual Stack* (collected March 2005), Treck Inc, Ohio, USA.  
<http://www.treck.com>.
- uC/TCP-IP* (collected March 2005), Micrium, Weston, FL USA.  
<http://www.micrium.com>.
- uIP* (collected March 2005), Adam Dunkels, Swedish Institute of Computer Science, Kista, Sweden.  
<http://www.sics.se/~adam/>.
- USNET* (collected March 2005), Micro Digital, Costa Mesa CA, USA.  
<http://www.smxinfo.com>.
- Vanstan, G. (2003), *COMLAB*, Control Gadgets, Unit 1/20 Loweana St, Southport, QLD, 4215, [geoff@controlgadgets.com.au](mailto:geoff@controlgadgets.com.au).  
<http://www.controlgadgets.com.au>.

---

*Web Server* (collected March 2005), Green Hills Software, Santa Babra CA 93101 USA.

<http://www.ghs.com>.

*XPort Embedded Web Server* (collected March 2005), Lantronix, Irvine CA, USA.

<http://www.lantronix.com>.

## Appendix A

# Project Specification

University of Southern Queensland  
Faculty of Engineering and Surveying

## ENG 4111/2 Research Project PROJECT SPECIFICATION

FOR: Simon BROWN  
TOPIC: Embedded IP for Small Devices  
SUPERVISOR: Dr John Leis

### Project AIM:

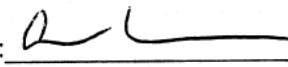

To develop a useful subset of the IP protocol stack for use in embedded devices. Implement PPP (point-to-point) protocol, the UDP (User Datagram Protocol, the TCP (Transmission Control Protocol), and finally a subset of application protocols such as DHCP (Dynamic Host Control Protocol and HTTP for transferring web pages. Ultimately, this will allow embedded devices to act as miniature web servers, simplifying the gathering of information from them.

**PROGRAMME:** Issue A, 3rd Feb 2005

1. Research available systems.
2. Evaluate available systems in terms of cost, security and function, both open source and commercial.
3. Research / Investigate range of embedded platforms (Micro-controllers) & software tools with a final goal of providing software portability, where practical.
4. Select target system hardware & programming language based on the previous results.
5. Write code to implement PPP, UDP, TCP, DHCP & HTTP with aim to restrict text size below 2 kB on chosen target system.
6. Construct working prototype of system.

*As time permits*

7. Implement FTP services for transferring of data files from embedded system.

AGREED:  (student)  (Supervisor)  
3/2/05 11/2/05

ID 0050029170

3/02/2005

## Appendix B

### Code Listing



## B.1 Source Code Listing

Listing B.1: The Prototype Embedded IP Stack Source Code

```
#include <mega128.h>
#asm.equ _lcd_port = 0x15;
PORTC #endasm
#include <lcd.h>
#include <stdlib.h>
#include <delay.h>
#include <string.h>

// Standard Input/Output functions
#include <stdio.h>

#define NULL      0
#define LCP       1
#define PAP       2
#define IPCP      3
#define IP        4
#define TCP       5

#define REQ       1
#define ACK       2
#define NAK       3
#define REJ       4
#define TERM      5

#define LCPState  1
#define PAPState  2
#define NCPState  3
#define IPState   4

// Declare your global variables here
unsigned char txBuff[1023],
    rxBuff[1023];
unsigned short tx_ptr, rx_ptr, maxRx;
unsigned short checksum;

unsigned char clientIP1;
unsigned char clientIP2;
unsigned char clientIP3;
unsigned char clientIP4;

unsigned char sourcePortH;
unsigned char sourcePortL;
unsigned char destinationPortH;
unsigned char destinationPortL;

unsigned char sequenceNumber1;
unsigned char sequenceNumber2;
unsigned char sequenceNumber3;
unsigned char sequenceNumber4;

unsigned char ackNumber1;
unsigned char ackNumber2;
unsigned char ackNumber3;
unsigned char ackNumber4;
unsigned char TCPFlags;
```

---

```

unsigned char    identIP = 0x00;
char            tempChar[768];
unsigned long int sequenceNumber;

// Declare some of my procedures
void            sendPacket(unsigned short length);
unsigned char    rxPacket();
unsigned short   crcCalc(unsigned char newByte,
                        unsigned short checkSumIn);
unsigned char    addByte(unsigned char newByte);
void            dummyModemCommand();
void            initialise();
unsigned char    rxProtocolType();
unsigned short   rxPacketOptions();

unsigned char    rxIdentNumber();
unsigned char    generatePacket(unsigned char protocol,
                                unsigned char code,
                                unsigned char identification,
                                unsigned char *tx_str);

void            storeIpInfo();
void            storeTCPInfo();

void            checkSumIP2();
void            checkSumTCP(unsigned char length);

// Main program structure .....

// -----
//  MAIN PROGRAM
// -----
void main(void)
{
    //
    //  Declare your local variables
    //  here
    //
    unsigned char state, txCount,
        rxPacketID, rxNew, txNew, i;

    initialise();
    lcd_init(20);

    while (!(UCSR0A & 0x80))
    {
        // wait
        ///for
        ///rxData
        ///

    }

    dummyModemCommand();

    // send
    ///some
    ///responses
    ///to
    ///simulate
    ///modem
    ///control
    ///

```

```

rxPacketID = 1;
txNew = 0;
rxNew = 0;
state = 0;

while(1)
{
    // Something in the RX Buffer?
    if((UCSR0A & 0x80))
    {
        lcd_clear();

        // lcd_putchar('B');
        if(rxPacket())
        {
            rxNew = 0x01;
            lcd_putchar('C');
            // Valid
            // packet
            // type?
            //

        }
    }

    if(rxNew)
    {
        // lcd_putchar('Y');
        switch(rxProtocolType())
        {
            case LCP:
                // lcd_putchar
                // ('1');

                if(rxBuff[4] == REQ)
                {
                    // if
                    // acceptable
                    // tx
                    // packet
                    //

                    if(rxPacketOptions() == 0x00D2)
                    {
                        // 2,5,7,8
                        // options
                        //
                        // move
                        // to
                        // next
                        // state,
                        // PAP
                        //

                        state = 11;
                    }
                }
            else if(rxBuff[4] == ACK)
            {
                state = 1;
            }
            else if(rxBuff[4] == TERM)

```

---

```

    {
        //
        // send term &
        // close connection //
        // Terminate connection ....
        //
        state = 14;
    }
    break;

case PAP:                                     // lcd_putchar
    ('2');                                     ///

    if (rxBuff[4] == REQ)
    {
        // if
        /// acceptable
        /// tx
        /// packet
        ///
        state = 21;
        // send
        /// ACK,
        /// ignore
        /// password
        /// sent.
        ///

    }
    else
    {
        //
        // send term &
        // close connection //
        // Terminate connection ....
        //
        state = 14;
    }
    break;

case IPCP:                                    // lcd_putchar
    ('3');                                    ///

    if (rxBuff[4] == REQ)
    {
        // if
        /// acceptable
        /// tx
        /// packet
        ///

        if ((rxBuff[7] < 0x0b) &&
            (rxBuff[10] == 0x00))
        {
            // less
            /// than
            /// 10
            /// bytes
            ///

            // send nak with address
            state = 31;
        }
    }

```

---

```

        else if (rxBuff[10] == 0x0a)
        {
            //
            // send ACK as ip has
            // been set
            //
            state = 32;
            break;
        }
    }
    else if (rxBuff[4] == ACK)
    {
        // Peer
        // accepts
        // configuration
        //
        state = 40;
        // move
        // to
        // PAP
        // state.
        //

    }
    else if (rxBuff[4] == NAK)
    {
        // I
        // don't
        // accept
        // configuration
        //
        state = 30;
        // can
        // we
        // negotiate
        //

    }
    else if (rxBuff[4] == REJ)
    {
        // Does
        // not
        // agree
        // with
        // proposed
        // config
        //

        state = 30;
    }
    else
    {
        //
        // send term &
        // close connection //
        // Terminate connection....
        //
        state = 14;
    }
}
break;
case IP:

```

---

```

    // lcd_putchar('P');
    storeIpInfo();

    if(rxBuff[13] == 0x06)
    {
        // TCP
        //
        // lcd_putchar('T');
        if(rxBuff[37] == 0x02)
        {
            // SYN
            // Sent
            // from
            // Client
            //

            storeTCPInfo();
            state = 50;
            break;
        }
        else if((rxBuff[37] == 0x18) &&
                (rxBuff[44] == 0x47))
        {
            // ack
            // and
            // GET
            // HTTP
            //

            storeTCPInfo();
            state = 51;
            lcd_putchar('A');
            break;
        }
        else if((rxBuff[37] == 0x10) &&
                (state == 55))
        {
            storeTCPInfo();
            state = 53;
            lcd_putchar('A');
            break;
        }
        state = 1;
    }

    break;

case NILL:
    lcd_putchar('5');
    state = 1;

    break;
    // must
    // be
    // no
    // received
    // valid
    // packet .....
    //

default:
    // lcd_putchar('6');
    state = 1;

```

```

        break;

    }

    // lcd_putchar('7');
    txNew = 1;
}

if(state == 1)
{
    txNew = 0;
}
rxNew = 0;
if(txNew)
{
    switch(state)
    {
        case 0:

            state = 0;
            rxPacketID++;
            tempChar[0] = 0x15;

            tempChar[1] = 0x02;

            tempChar[2] = 0x06;

            tempChar[3] = 0x00;
            tempChar[4] = 0x0A;
            tempChar[5] = 0x00;
            tempChar[6] = 0x00;
            tempChar[7] = 0x03;

            tempChar[8] = 0x04;

            tempChar[9] = 0xC0;

            tempChar[10] = 0x23;

            tempChar[11] = 0x05;
            tempChar[12] = 0x06;
            tempChar[13] = 0x01;
            tempChar[14] = 0x02;
            tempChar[15] = 0x03;
            tempChar[16] = 0x04;
            tempChar[17] = 0x07;

// end
// of
// switch
//

// end
// of
// if
//

// LCP
// HOST
// send
// REQ
//

// length
//
// ACCM
//
// length
//

// PAP
//
// length
//
// Authentication
// type
//
// cont
//

// Compression

```

```

tempChar[18] = 0x02;          ///
                               /// length
tempChar[19] = 0x08;          ///
                               /// Compression
tempChar[20] = 0x02;          ///
                               /// length
                               ///
txCount = generatePacket(LCP,
                        REQ,
                        rxPacketID,
                        &tempChar[0]);

sendPacket(txCount);
break;

case 1:
    state = 1;
    break;

case 11:
                               /// send
                               /// ACK
                               ///
tempChar[0] = 0x11;           ///
                               /// length
tempChar[1] = 0x02;           ///
                               /// ACCM
tempChar[2] = 0x06;           ///
                               /// length
                               ///
tempChar[3] = 0x00;
tempChar[4] = 0x0A;
tempChar[5] = 0x00;
tempChar[6] = 0x00;
tempChar[7] = 0x05;
tempChar[8] = 0x06;
tempChar[9] = rxBuff[16];
tempChar[10] = rxBuff[17];
tempChar[11] = rxBuff[18];
tempChar[12] = rxBuff[19];
tempChar[13] = 0x07;          /// Compression
                               ///
tempChar[14] = 0x02;          ///
                               /// length
tempChar[15] = 0x08;          ///
                               /// Compression
tempChar[16] = 0x02;          ///
                               /// length
                               ///
txCount = generatePacket(LCP,
                        ACK,
                        rxIdentNumber(),
                        &tempChar[0]);

sendPacket(txCount);
state = 20;
break;

case 14:
                               /// send
                               /// Term
                               ///
tempChar[0] = 0x00;           ///
                               /// length
                               ///

```



---

```

    txCount = generatePacket(LCP,
                             TERM,
                             rxIdentNumber,
                             &tempChar[0]);
    sendPacket(txCount);
                                     // send
                                     /// packet
                                     /// immediately
                                     /// to
                                     /// terminate
                                     /// connection
                                     ///

    dummyModemCommand();
    break;

case 20:
    state = 20;
    break;

case 21:
                                     // send
                                     /// send
                                     /// ACK
                                     ///
                                     // length,
                                     /// zero
                                     /// for
                                     /// PAP
                                     /// Ack
                                     ///

    txCount = generatePacket(PAP,
                             ACK,
                             rxIdentNumber(),
                             &tempChar[0]);

    sendPacket(txCount);
    state = 30;
    break;

case 30:
                                     // IPCP
                                     /// send
                                     /// send
                                     /// REQ
                                     ///

    state = 30;
    break;

case 31:
                                     // send
                                     /// send
                                     /// NAK
                                     ///
                                     // length
                                     ///
                                     // IP
                                     /// Address
                                     ///
                                     // length
                                     ///

    tempChar[0] = 0x07;
    tempChar[1] = 0x03;
    tempChar[2] = 0x06;
    tempChar[3] = 0x0a;
    tempChar[4] = 0x01;
    tempChar[5] = 0x01;

```



```

break;
case 50:

tempChar[1] = 0x45;
tempChar[2] = 0x00;
tempChar[3] = 0x00;

tempChar[4] = 0x30;

tempChar[5] = 0x00;
identIP++;
tempChar[6] = 1;

tempChar[7] = 0x40;
tempChar[8] = 0x00;

tempChar[9] = 0x80;
tempChar[10] = 0x06;

tempChar[11] = 0x00;
tempChar[12] = 0x00;

tempChar[13] = 0x0a;

tempChar[14] = 0x01;

```

```

///
// TCP
///Packet
///rxed
///SYN
///from
///Client
///send
///SYN
///ACK
///
// version
///
// DSF
///
// IP
///TCP
///frame
///length
///
// IP
///TCP
///frame
///length
///
// Ident
///
// LSByte
///plus
///increment.
///
// Flags
///
// Fragment
///offset
///
// TTL
///
// Protocol
///6
///TCP
///
// checksum
///
// checksum
///
// IP
///Source
///
// IP
///Source
///

```

---

```

tempChar[15] = 0x01;          /// IP
                               /// Source
                               /// 
tempChar[16] = 0x02;          /// IP
                               /// Source
                               /// 
tempChar[17] = 10;            /// IP
                               /// Destination
                               /// 
tempChar[18] = 01;            /// IP
                               /// Destination
                               /// 
tempChar[19] = 01;            /// IP
                               /// Destination
                               /// 
tempChar[20] = 01;            /// IP
                               /// Destination
                               /// 

checkSumIP2();

/// TCP Part
ackNumber4++;
sequenceNumber4++;
tempChar[21] = destinationPortH; /// Our
                               /// Source
                               /// 
tempChar[22] = destinationPortL;
tempChar[23] = sourcePortH;    /// Our
                               /// Destination
                               /// 
tempChar[24] = sourcePortL;
tempChar[25] = ackNumber1;     /// Our
                               /// Sequence
                               /// Number
                               /// 

tempChar[26] = ackNumber2;
tempChar[27] = ackNumber3;
tempChar[28] = ackNumber4;

tempChar[29] = sequenceNumber1;
tempChar[30] = sequenceNumber2;
tempChar[31] = sequenceNumber3;
tempChar[32] = sequenceNumber4;

tempChar[33] = 0x70;           /// header
                               /// length
                               /// n*32 bit
                               /// words
                               /// 
tempChar[34] = 0x12;           /// SYN
                               /// ACK
                               /// 

tempChar[35] = 0x20;           /// Window
                               /// size

```

```

tempChar[36] = 0x00;
tempChar[37] = 0x00;
tempChar[38] = 0x00;
tempChar[39] = 0x00;
tempChar[40] = 0x00;

tempChar[41] = 0x02;
tempChar[42] = 0x04;
tempChar[43] = 0x02;
tempChar[44] = 0x18;

tempChar[45] = 0x01;
tempChar[46] = 0x01;
tempChar[47] = 0x04;
tempChar[48] = 0x02;

checksumTCP(28);

tempChar[0] = 0x31;

txCount = generatePacket(IP,
                          ACK,
                          rxIdentNumber(),
                          &tempChar[0]);

sendPacket(txCount);
state = 1;
break;

case 51:

tempChar[01] = 0x45;
tempChar[02] = 0x00;

```

```

///high
///byte
///
/// window
///size
///128
///bytes
///
/// CheckSum
///
/// Urgent
///pointer
///
/// Urgent
///pointer
///
/// length
///of
///total
///packet
///
/// TCP
///Packet
///rxed
///ACK
///HTTP
///GET,
///send
///ACK
///
/// version
///
/// DSF
///

```

---

```

tempChar[03] = 0x00;      /// IP
                          /// TCP
                          /// frame
                          /// length
                          /// 
tempChar[04] = 48;        /// IP
                          /// TCP
                          /// frame
                          /// length
                          /// 30
                          /// hex
                          /// 48
                          /// 
tempChar[05] = 0x00;      /// Ident
                          /// 
tempChar[6] = 2;          /// LSByte
                          /// plus
                          /// increment.
                          /// 
tempChar[7] = 0x40;        /// Flags
                          /// 
tempChar[8] = 0x00;        /// Fragment
                          /// offset
                          /// 
tempChar[9] = 0x80;        /// TTL
                          /// 
tempChar[10] = 0x06;       /// Protocol
                          /// 6
                          /// TCP
                          /// 
tempChar[11] = 0x00;       /// checksum
                          /// 
tempChar[12] = 0x00;       /// checksum
                          /// 
tempChar[13] = 0x0a;        /// IP
                          /// Source
                          /// 
tempChar[14] = 0x01;        /// IP
                          /// Source
                          /// 
tempChar[15] = 0x01;        /// IP
                          /// Source
                          /// 
tempChar[16] = 0x02;        /// IP
                          /// Source
                          /// 
tempChar[17] = 0x0a;        /// IP
                          /// Destination
                          /// 
tempChar[18] = 0x01;        /// IP
                          /// Destination
                          /// 
tempChar[19] = 0x01;        /// IP
                          /// Destination

```

```

tempChar[20] = 0x01;          ///
                               /// IP
                               /// Destination
                               ///

checkSumIP2();

// TCP Part
tempChar[21] = destinationPortH; /// Our
                               /// Source
                               ///

tempChar[22] = destinationPortL;
tempChar[23] = sourcePortH;    /// Our
                               /// Destination
                               ///

tempChar[24] = sourcePortL;
tempChar[25] = ackNumber1;     /// Our
                               /// Sequence
                               /// Number
                               ///

tempChar[26] = ackNumber2;
tempChar[27] = ackNumber3;

// ackNumber4++;
tempChar[28] = ackNumber4;

tempChar[29] = sequenceNumber1;
tempChar[30] = sequenceNumber2;

sequenceNumber++;

sequenceNumber =
    sequenceNumber +
    rxBuff[7] -
    40;
if(rxBuff[6])
{
    sequenceNumber += 255;
}

sequenceNumber4 = (unsigned char)sequenceNumber;

tempChar[32] = sequenceNumber4;

sequenceNumber3 =
    (unsigned char)
    (sequenceNumber >> 8);

tempChar[31] = sequenceNumber3;

tempChar[33] = 0x70;          /// header
                               /// length
                               /// n*32 bit
                               /// words
                               ///

tempChar[34] = 0x10;          /// SYN
                               /// ACK
                               ///

tempChar[35] = 0x20;          /// Window

```

```

tempChar[36] = 0x00;
tempChar[39] = 0x00;
tempChar[40] = 0x00;

tempChar[37] = 0x00;
tempChar[38] = 0x00;

tempChar[41] = 0x01;
tempChar[42] = 0x01;
tempChar[43] = 0x01;
tempChar[44] = 0x01;
tempChar[45] = 0x01;
tempChar[46] = 0x01;
tempChar[47] = 0x01;
tempChar[48] = 0x01;

checkSumTCP(28);
tempChar[0] = 49;

txCount = generatePacket(IP,
                          ACK,
                          rxIdentNumber(),
                          &tempChar[0]);

sendPacket(txCount);

// delay_ms(1000);
// SEND HTTP Page
tempChar[04] = 128;

tempChar[6] = 3;

```

```

// size
// high
// byte
//
// window
// size
// 128
// bytes
//
// Urgent
// pointer
//
// Urgent
// pointer
//
// length
// of
// total
// packet
// was
// 49
//
// IP
// TCP
// frame
// length
// 80
// hex
// 128
//
// LSByte
// plus
// increment.
//

```



---

```

tempChar[11] = 0x00;           // Clear
                               ///Checksum
                               ///

tempChar[12] = 0x00;           // Clear
                               ///Checksum
                               ///

checksumIP2();

// TCP Part
tempChar[34] = 0x18;           // ACK
                               ///PSH
                               ///

tempChar[37] = 0x00;           // Clear
                               ///Checksum
                               ///

tempChar[38] = 0x00;           // Clear
                               ///Checksum
                               ///

// HTTP Data
i = 49;
tempChar[i++] = 'H';
tempChar[i++] = 'T';
tempChar[i++] = 'T';
tempChar[i++] = 'P';
tempChar[i++] = '/';
tempChar[i++] = '1';
tempChar[i++] = '.';
tempChar[i++] = '0';
tempChar[i++] = '_';
tempChar[i++] = '2';
tempChar[i++] = '0';
tempChar[i++] = '0';
tempChar[i++] = '_';
tempChar[i++] = 'O';
tempChar[i++] = 'K';
tempChar[i++] = 0x0d;
tempChar[i++] = 0x0a;
tempChar[i++] = 'C';
tempChar[i++] = 'o';
tempChar[i++] = 'n';
tempChar[i++] = 't';
tempChar[i++] = 'e';
tempChar[i++] = 'n';
tempChar[i++] = 't';
tempChar[i++] = '-';
tempChar[i++] = 'T';
tempChar[i++] = 'y';
tempChar[i++] = 'p';
tempChar[i++] = 'e';
tempChar[i++] = ':';
tempChar[i++] = '_';
tempChar[i++] = 't';
tempChar[i++] = 'e';
tempChar[i++] = 'x';
tempChar[i++] = 't';

```

```

tempChar[i++] = '/';
tempChar[i++] = 'h';
tempChar[i++] = 't';
tempChar[i++] = 'm';
tempChar[i++] = 'l';
tempChar[i++] = 0x0d;
tempChar[i++] = 0x0a;
tempChar[i++] = 0x0d;
tempChar[i++] = 0x0a;
tempChar[i++] = '<';
tempChar[i++] = 'b';
tempChar[i++] = 'o';
tempChar[i++] = 'd';
tempChar[i++] = 'y';
tempChar[i++] = '>';
tempChar[i++] = '_';
tempChar[i++] = 'F';
tempChar[i++] = 'i';
tempChar[i++] = 'n';
tempChar[i++] = 'a';
tempChar[i++] = 'l';
tempChar[i++] = 'l';
tempChar[i++] = 'y';
tempChar[i++] = '_';
tempChar[i++] = 's';
tempChar[i++] = 'o';
tempChar[i++] = 'm';
tempChar[i++] = 'e';
tempChar[i++] = '_';
tempChar[i++] = 't';
tempChar[i++] = 'h';
tempChar[i++] = 'i';
tempChar[i++] = 'n';
tempChar[i++] = 'g';
tempChar[i++] = '_';
tempChar[i++] = '_';
tempChar[i++] = '_';
tempChar[i++] = '<';
tempChar[i++] = '/';
tempChar[i++] = 'b';
tempChar[i++] = 'o';
tempChar[i++] = 'd';
tempChar[i++] = 'y';
tempChar[i++] = '>';
tempChar[i++] = 0x0a;

checkSumTCP(108);

//
// tempChar[38] =
// tempChar[38] + 1;
//
tempChar[0] = 129;

// was
///108
///

// length
///of

```

```

                                                                    ///total
                                                                    ///packet
                                                                    ///was
                                                                    ///128
                                                                    ///
txCount = generatePacket(IP,
                          ACK,
                          1,
                          &tempChar[0]);

sendPacket(txCount);

// SEND HTTP Page
tempChar[04] = 48;
                                                                    /// IP
                                                                    ///TCP
                                                                    ///frame
                                                                    ///length
                                                                    ///80
                                                                    ///hex
                                                                    ///128
                                                                    ///

tempChar[6] = 4;
                                                                    /// LSByte
                                                                    ///plus
                                                                    ///increment.
                                                                    ///

tempChar[11] = 0x00;
                                                                    /// Clear
                                                                    ///Checksum
                                                                    ///

tempChar[12] = 0x00;
                                                                    /// Clear
                                                                    ///Checksum
                                                                    ///

checksumIP2();

// TCP Part
tempChar[28] = 82;
tempChar[34] = 0x11;
                                                                    /// ACK
                                                                    ///FIN
                                                                    ///

tempChar[37] = 0x00;
                                                                    /// Clear
                                                                    ///Checksum
                                                                    ///

tempChar[38] = 0x00;
                                                                    /// Clear
                                                                    ///Checksum
                                                                    ///

checksumTCP(28);

tempChar[0] = 49;
                                                                    /// length
                                                                    ///of
                                                                    ///total
                                                                    ///packet
                                                                    ///was
                                                                    ///49
                                                                    ///

txCount = generatePacket(IP,
                          ACK,
                          rxIdentNumber(),
                          &tempChar[0]);

```

```

        sendPacket ( txCount );
        state = 55;
        break;
    case 52:
        /// TCP
        /// Packet
        /// rxd
        /// ACK
        /// from
        /// Client
        /// ACK
        /// and
        /// send
        /// HTTP
        /// response
        ///

        break;
    case 53:
        /// Send
        /// Final
        /// ACK
        /// from
        /// Webpage
        /// serve
        ///
        tempChar [01] = 0x45;
        tempChar [02] = 0x00;
        tempChar [03] = 0x00;
        tempChar [04] = 48;
        tempChar [05] = 0x00;
        tempChar [6] = 5;
        tempChar [7] = 0x40;
        tempChar [8] = 0x00;
        tempChar [9] = 0x80;
        /// version
        /// DSF
        ///
        /// IP
        /// TCP
        /// frame
        /// length
        ///
        /// IP
        /// TCP
        /// frame
        /// length
        /// 30
        /// hex
        /// 48
        ///
        /// Ident
        ///
        /// LSByte
        /// plus
        /// increment.
        ///
        /// Flags
        ///
        /// Fragment
        /// offset
        ///
        /// TTL

```

```

tempChar[10] = 0x06;          ///
                              /// Protocol
                              /// 6
                              /// TCP
tempChar[11] = 0x00;          ///
                              /// checksum
tempChar[12] = 0x00;          ///
                              /// checksum
                              ///
tempChar[13] = 0x0a;          ///
                              /// IP
                              /// Source
tempChar[14] = 0x01;          ///
                              /// IP
                              /// Source
tempChar[15] = 0x01;          ///
                              /// IP
                              /// Source
tempChar[16] = 0x02;          ///
                              /// IP
                              /// Source
tempChar[17] = 0x0a;          ///
                              /// IP
                              /// Destination
tempChar[18] = 0x01;          ///
                              /// IP
                              /// Destination
tempChar[19] = 0x01;          ///
                              /// IP
                              /// Destination
tempChar[20] = 0x01;          ///
                              /// IP
                              /// Destination
                              ///

checksumIP2();
// TCP Part
tempChar[21] = destinationPortH; ///
                              /// Our
                              /// Source
                              ///
tempChar[22] = destinationPortL;
tempChar[23] = sourcePortH;    ///
                              /// Our
                              /// Destination
                              ///
tempChar[24] = sourcePortL;
tempChar[25] = ackNumber1;     ///
                              /// Our
                              /// Sequence
                              /// Number
                              ///
tempChar[26] = ackNumber2;
tempChar[27] = ackNumber3;
// ackNumber4++;
tempChar[28] = ackNumber4;
tempChar[29] = sequenceNumber1;

```

---

```

tempChar[30] = sequenceNumber2;
sequenceNumber++;
sequenceNumber =
    sequenceNumber +
    rxBuff[7] -
    40;
if(rxBuff[6])
{
    sequenceNumber += 255;
}

sequenceNumber4 = (unsigned char)sequenceNumber;
tempChar[32] = sequenceNumber4;
sequenceNumber3 =
    (unsigned char)
    (sequenceNumber >> 8);
tempChar[31] = sequenceNumber3;
tempChar[33] = 0x70;                                     /// header
                                                         ///length
                                                         ///n*32bit
                                                         ///words
                                                         ///
tempChar[34] = 0x10;                                     /// SYN
                                                         ///ACK
                                                         ///
tempChar[35] = 0x20;                                     /// Window
                                                         ///size
                                                         ///high
                                                         ///byte
                                                         ///
tempChar[36] = 0x00;                                     /// window
                                                         ///size
                                                         ///128
                                                         ///bytes
                                                         ///
tempChar[39] = 0x00;                                     /// Urgent
                                                         ///pointer
                                                         ///
tempChar[40] = 0x00;                                     /// Urgent
                                                         ///pointer
                                                         ///

tempChar[37] = 0x00;
tempChar[38] = 0x00;

tempChar[41] = 0x01;
tempChar[42] = 0x01;
tempChar[43] = 0x01;
tempChar[44] = 0x01;
tempChar[45] = 0x01;
tempChar[46] = 0x01;
tempChar[47] = 0x01;
tempChar[48] = 0x01;

```

---

```

        checksumTCP(28);
        tempChar[0] = 49;                                     // length
                                                             // of
                                                             // total
                                                             // packet
                                                             // was
                                                             // 49
                                                             //

        txCount = generatePacket(IP,
                                   ACK,
                                   rxIdentNumber(),
                                   &tempChar[0]);

        sendPacket(txCount);
        state = 1;

        break;

    case 55:                                                  // TCP
                                                             // Packet
                                                             // rxed
                                                             // ACK
                                                             // from
                                                             // Client
                                                             // ACK
                                                             // and
                                                             // send
                                                             // HTTP
                                                             // response
                                                             //

        break;

    default:
        state = 14;
        break;
    }

    txNew = 0;
}

}

} // end of main

// PROCEDURES

// _____
//   storeIpInfo
//   _____
void storeIpInfo()
{
    clientIP1 = rxBuff[16];
    clientIP2 = rxBuff[17];
    clientIP3 = rxBuff[18];
}

```

```

    clientIP4 = rxBuff[19];
}

// -----
//   storeTCPInfo
// -----
void storeTCPInfo()
{
    sourcePortH = rxBuff[24];
    sourcePortL = rxBuff[25];

    destinationPortH = rxBuff[26];
    destinationPortL = rxBuff[27];

    sequenceNumber1 = rxBuff[28];
    sequenceNumber2 = rxBuff[29];
    sequenceNumber3 = rxBuff[30];
    sequenceNumber4 = rxBuff[31];

    sequenceNumber =
        ((unsigned long)sequenceNumber1 <<
         24;
    sequenceNumber = sequenceNumber +
        (
            ((unsigned long)sequenceNumber2 <<
             16
            );
    sequenceNumber = sequenceNumber + (((unsigned long)
        sequenceNumber3 << 8);
    sequenceNumber = sequenceNumber + sequenceNumber4;

    ackNumber1 = rxBuff[32];
    ackNumber2 = rxBuff[33];
    ackNumber3 = rxBuff[34];
    ackNumber4 = rxBuff[35];
    TCPFlags = rxBuff[37];
}

// -----
//   16 bit CheckSum
// -----
void checkSumIP2()
{
    unsigned short    tempShort = 0;
    unsigned long int tempLong2,
        tempLong = 0;
    unsigned char     i;

    for(i = 1; i < 21; i++)
    {
        tempShort = tempChar[i];
        tempShort = tempShort << 8;
        tempShort = tempShort & 0xFF00;
        tempShort = tempShort + tempChar[++i];
        tempLong = tempLong + ((unsigned long)tempShort;
    }

    tempLong2 = tempLong >> 16;
    tempLong = tempLong & 0xFFFF;
    tempLong = tempLong + tempLong2;

```



```

tempLong = tempLong ^ 0xFFFF;

tempChar[12] = (unsigned char)tempLong;
tempLong = tempLong >> 8;
tempChar[11] = (unsigned char)tempLong;
}

// -----
// 16 bit CheckSum
// -----
void checkSumTCP(unsigned char length)
{
    unsigned short    tempShort = 0;
    unsigned long int tempLong2;
    unsigned long     tempLong = 0x160b;
    unsigned char     i = 21;

    tempLong = tempLong + length;
    length = length + 20;

    for(i = 1; i < length; i++)
    {
        tempShort = tempChar[i];
        tempShort = tempShort << 8;
        tempShort = tempShort & 0xFF00;
        tempShort = tempShort + tempChar[++i];
        tempLong = tempLong + (unsigned long)tempShort;
    }

    tempLong2 = tempLong >> 16;
    tempLong = tempLong & 0xFFFF;
    tempLong = tempLong + tempLong2;

    tempLong = tempLong ^ 0xFFFF;

    tempChar[38] = (unsigned char)tempLong;
    tempLong = tempLong >> 8;
    tempChar[37] = (unsigned char)tempLong;
}

// -----
// rxProtocolType
// -----
unsigned char rxProtocolType()
{
    if((rxBuff[2] == (unsigned char)0xC0
        ) &&
        (rxBuff[3] == (unsigned char)0x21))
        return LCP;
    else if((rxBuff[2] == (unsigned char)0xC0) &&
        (rxBuff[3] == (unsigned char)0x23))
        return PAP;
    else if((rxBuff[2] == (unsigned char)0x80) &&
        (rxBuff[3] == (unsigned char)0x21))
        return IPCP;
    else if((rxBuff[2] == (unsigned char)0x00) &&
        (rxBuff[3] == (unsigned char)0x21))

```

```

    return IP;

    // lcd_putchar('E');
    return NILL;
}

// -----
// rxPacketOptions
// -----
unsigned short rxPacketOptions()
{
    unsigned char    packet_ptr = 8; // Position
                                   // of
                                   // first
                                   // option
                                   //
    unsigned short    options = 0;
    unsigned short    temp = 1;
    lcd_putchar('9');
    while(packet_ptr < maxRx)
    {
        temp = temp << (rxBuff[packet_ptr] - 1);
        options = options | temp;
        packet_ptr++;
        packet_ptr = packet_ptr + (rxBuff[packet_ptr] - 1);
        temp = 1;
    }

    lcd_putchar('9');
    return options;
}

// -----
// rxIdentNumber
// -----
unsigned char rxIdentNumber()
{ // Identification number
    return rxBuff[5];
}

// -----
// sendPacket
// -----
void sendPacket(unsigned short length)
{
    int                i = 0;
    unsigned short    tChecksum;
    putchar(0x7E);

    while(length)
    {
        putchar(txBuff[i]);
    }
}

```

```

// Add
// start
// sequence
//
// Roll
// through
// array ,
//
// until
// done...

```

```

        i++;                                     /// Increment
                                                /// through
                                                /// array
                                                ///
        length--;                               /// Decrement
                                                /// array
                                                /// count...
                                                ///
    }
    checksum = checksum ^ 0xFFFF;               /// invert
                                                /// checksum,
                                                /// ones
                                                /// complement.
                                                ///
    tChecksum = checksum & 0xFF;
    if((unsigned char)tChecksum <
        (unsigned char)0x20)
    {
        putchar(0x7D);
        tChecksum ^= 0x20;
        putchar((unsigned char)tChecksum);
    }
    else if((unsigned char)tChecksum == 0x7e)
    {
        putchar(0x7D);
        putchar(0x5e);
    }
    else if((unsigned char)tChecksum == 0x7d)
    {
        putchar(0x7D);
        putchar(0x5D);
    }
    else
    {
        putchar((unsigned char)tChecksum);
    }
    tChecksum = checksum >> 8;
    if((unsigned char)tChecksum <
        (unsigned char)0x20)
    {
        putchar(0x7D);
        tChecksum ^= 0x20;
        putchar((unsigned char)tChecksum);    /// insert
                                                /// escape
                                                /// sequence
                                                /// if
                                                /// <
                                                /// 0x20
                                                ///
    }
    else if((unsigned char)tChecksum == 0x7e)

```

```

    {
        putchar(0x7D);
        putchar(0x5e);
    }
    else if((unsigned char)tChecksum == 0x7d)
    {
        putchar(0x7D);
        putchar(0x5D);
    }
    else
    {
        putchar((unsigned char)tChecksum);
    }
    putchar(0x7E);
}

// -----
//   rxPacket
// -----
unsigned char rxPacket()
{
    unsigned char c;
    // lcd_putchar('I');
    rx_ptr = 0;
    // Rest rx_ptr index
    if((c = getchar()) != 0x7E)
    {
        lcd_putchar('0');
        return 0;
    }
    else
    {
        while((c = getchar()) != 0x7E)
        { // test second byte for frame end
            if(c == 0x7D)
            {
                rxBuff[rx_ptr] = 0x20 ^ (getchar()); // xor
                                                         // byte
                                                         // with
                                                         // 0x20
                                                         // if
                                                         // so
                                                         //
            }
            else if((c == 0x21) && (rx_ptr == 0))
            {
                rxBuff[0] = 0xFF;
                rxBuff[1] = 0x03;
                // only
                // interested
                // in
                // IP
                // header....
            }
        }
    }
}

```

```

        rxBuff[2] = 0x00;
        rxBuff[3] = c;
        rx_ptr++;
        rx_ptr++;
        rx_ptr++;
    }
    else if((c != 0xFF) && (rx_ptr == 0))
    {
        rxBuff[0] = 0xFF;

        rxBuff[1] = 0x03;
        rxBuff[2] = c;
        rx_ptr++;
        rx_ptr++;
    }
    else if((rx_ptr == 1) && (c != 0x7D))
    {
        rxBuff[1] = 0x03;
    }
    else
    {
        rxBuff[rx_ptr] = c;

        rx_ptr++;

    }

    rx_ptr++;

}

// lcd_putchar('F');
maxRx = rx_ptr;
return 0x01;
}

// _____
// HDLC crcCalc
// _____
unsigned short crcCalc(unsigned char newByte,
                        unsigned short checksumIn)
{
    int i; // Calculate
           //new crc-16
           //checksum,
    unsigned char f; // performed on
                    //all bytes

```



---

```

        count++;
    }
    else
        txBuff[tx_ptr++] = newByte;

        count++;
    return count;
}

// -----
//  dummyModemCommand
// -----
void dummyModemCommand()
{
    printf("OK\n");

    // lcd_putchar('M');
    delay_ms(750);
    printf("OK\n");

    // lcd_putchar('M');
    delay_ms(750);
    printf("OK\n");

    // lcd_putchar('M');
    delay_ms(100);
    printf("OK\n");

    // lcd_putchar('M');
}

// -----
//  generate Packet
// -----
unsigned char generatePacket(unsigned char protocol,
                             unsigned char code,
                             unsigned char identification,
                             unsigned char *tx_str)
{
    unsigned char length;

    // length
    // of
    // information
    // field
    // (
    // no
    // PPP
    // header
    // or
    // CRC
    // or
    // $7E)

```

```

unsigned char count = 0;          ///reset
                                   ///byte
                                   ///count,
                                   ///includes
                                   ///all
                                   ///escape
                                   ///sequences
                                   ///reset
tx_ptr = 0;                       ///buffer
                                   ///pointer
                                   ///to
                                   ///zero
                                   ///reset

checksum = 0xFFFF;

count += addByte(0xFF);
count += addByte(0x03);

if(protocol == LCP)
{
    count += addByte(0xC0);
    count += addByte(0x21);
    count += addByte(code);
    count += addByte(identification);
    length = tx_str[0];
    count += addByte(0x00);
    count += addByte(length + 3); ///must
                                   ///include
                                   ///code
                                   ///of
                                   ///code,
                                   ///ID
                                   ///'
                                   ///length
                                   ///fields.
                                   ///reset
}

else if(protocol == PAP)
{
    count += addByte(0xC0);
    count += addByte(0x23);
    count += addByte(code);
    count += addByte(identification);
    length = tx_str[0];
    count += addByte(0x00);
    count += addByte(length + 4); ///must
                                   ///include
                                   ///code
                                   ///of
                                   ///code,
                                   ///ID
                                   ///'
                                   ///length
                                   ///fields.

```



```

}
else if(protocol == IPCP)
{
    count += addByte(0x80);
    count += addByte(0x21);
    count += addByte(code);
    count += addByte(identification);
    length = tx_str[0];
    count += addByte(0x00);
    count += addByte(length + 3); // must
                                //include
                                //code
                                //of
                                //code,
                                //ID
                                //£
                                //length
                                //fields.
                                //
}
else if(protocol == IP)
{
    count += addByte(0x00);
    count += addByte(0x21);
    length = tx_str[0];
}

if(length)
{
    length--;
    tx_str = &tx_str[1];
    while(length)
    {
        length--;
        count += addByte(*tx_str);
        tx_str++;
    }
}

return count;
}
// end
//of
//packet
//thingy
//

// -----
// Intialise Chip
// -----
void initialise()
{
    PORTA = 0x00;
    DDRA = 0x00;
    PORTB = 0x00;
    DDRB = 0x00;
    PORTC = 0x00;

```

```

DDRC = 0x00;
PORTD = 0x00;
DDRD = 0x00;
PORTE = 0x00;
DDRE = 0x02;
PORTF = 0x00;
DDRF = 0x00;
PORTG = 0x00;
DDRG = 0x00;
ASSR = 0x00;
TCCR0 = 0x00;
TCNT0 = 0x00;
OCR0 = 0x00;
TCCR2 = 0x00;
TCNT2 = 0x00;
OCR2 = 0x00;
TCCR3A = 0x00;
TCCR3B = 0x00;
TCNT3H = 0x00;
TCNT3L = 0x00;
ICR3H = 0x00;
ICR3L = 0x00;
OCR3AH = 0x00;
OCR3AL = 0x00;
OCR3BH = 0x00;
OCR3BL = 0x00;
OCR3CH = 0x00;
OCR3CL = 0x00;
EICRA = 0x00;
EICRB = 0x00;
EIMSK = 0x00;
TIMSK = 0x00;    // was 04 enables
                  ///interrupt on
                  ///Timer/Counter 1,
                  ///Overflow Interrupt
                  ///Enable

ETIMSK = 0x00;
UCSR0A = 0x00;

// Usart setup
UCSR0B = 0x18;
UCSR0C = 0x06;
UBRR0H = 0x00;
UBRR0L = 0x5F;
ACSR = 0x80;
SFIOR = 0x00;

// Timer
TCCR1A = 0x00;
TCCR1B = 0x00;    // was 02 timer
                  ///control register
                  ///02 = clk/64

TCNT1H = 0x00;
TCNT1L = 0x00;
ICR1H = 0x00;
ICR1L = 0x00;
OCR1AH = 0x00;
OCR1AL = 0x00;
OCR1BH = 0x00;
OCR1BL = 0x00;
OCR1CH = 0x00;
OCR1CL = 0x00;

```

```
}
```

## Appendix C

# Manufacturer Datasheets

### C.1 JED Micoprocessor Datasheets



# JED MICROPROCESSORS PTY LTD

173 Boronia Rd, Boronia, (PO Box 30), Victoria, 3155, Australia  
Phone: +61 3 9762 3588 Fax: +61 3 9762 5499  
<http://www.jedmicro.com.au> email: [jed@jedmicro.com.au](mailto:jed@jedmicro.com.au)

## AVR570 plug-in Atmel RISC CPU module

The AVR570 is a 47mm square module providing potential users of the ATmega128 with a method of using the most powerful CPU in the 8-bit Atmel range without needing expensive surface mount design, masking, pick-and-place machine programming and production runs to justify it.

Instead, users can simply lay out a square pad layout for four, 16-pin 0.1" spaced strip connectors (with 0.025" square pins, i.e. 0.9mm holes) and have a plug-in module which is mass-produced by JED and programmed by users using the on-board ISP (In System Programming) connector either stand-alone in a test jig or installed in the final board.

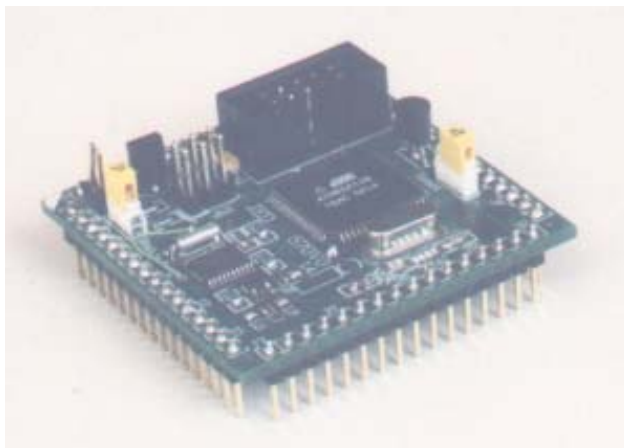
During the debug phase, users can connect the low-cost Atmel JTAG ICE (In Circuit Emulator) to an optional 10-pin socket and debug systems in their final environment, with all I/O active.

## Standard functions of the AVR570

The AVR570 has the close CPU functions provided, i.e. the crystal for the CPU clock and the power-fail detector/reset generator.

The **crystal** is on the module, but the pins for the oscillator are still available for external connection off-module, but via Links 2 and 3 which must be bridged before external use via pins 23 and 24. Normally a 3.6864 MHz crystal is installed, but special requests can be accommodated. (The ATmega128 can run at up to 16 MHz.)

While external connection of the clock is possible, the longer lines for the crystal oscillator pins down to the base board would be a probable source of radio frequency radiation and is NOT recommended practice. An external clock could be fed to the CPU via these pins to XTAL1 via Pin 24. The CPU oscillator output can also be used for off-



## AVR570 CPU module, ATmega128-based

- High speed RISC CPU with 128K high-security FLASH program memory, 4K Byte of SRAM, 4K Byte of EEPROM;
- Module is 47mm square, and uses standard 0.1" spaced connectors on four sides to connect to user-designed base-board or JED AVR572 prototype or AVR573 base boards;
- Two UART interfaces, available on pins at CMOS levels, available for RS232, RS485, GPS or GSM/GPRS;
- On-board reset/power fail detector;
- On-board optional 5 volt regulator;
- On-board 6-pin ISP programming port and 10-pin JTAG ICE connector;
- On-board RESET switch and on-board RUN/PROGRAM switch;
- Physically and electrically compatible with ATmega103 CPU module from Olimex Ltd, but uses newer, more powerful ATmega128 with improved 133 instruction set, dual UARTs, hardware I<sup>2</sup>C system and ICE debug system;
- Optional Real Time Clock (DS1305) via SPI using port G pins for interface, with on-board lithium battery behind board.



Designed and manufactured in Australia by an Australian-owned company.

module functions, but if this is done, the oscillator power level should be set to CKOPT ON. If the signal is not used externally, the oscillator should be set to CKOPT OFF to reduce RF radiation from the module.

There is a DS1233 CPU Reset generator on the AVR570 to detect power supply drops and reset the CPU. It also delays CPU start-up until after the oscillator starts and the CPU stabilises. A small slider switch (SW2) across the RESET line can be used to reset the CPU when desired during testing.

### In-System Programming Port

Connector J2 is a 6-pin male ISP header on the board for programming the FLASH memory and the EEPROM memory in the ATmega128 CPU. This connector is compatible with the Atmel STK500 prototype board, connector J200.

It is also compatible with programming devices from PC parallel and serial ports available from JED, Kanda and Atmel. (A simple cable scrambler can convert the earlier 10-pin standard to this 6-pin standard. The signals are electrically identical.)

A small slide switch (SW1) is closed in the RUN mode, but opened in programming mode, to isolate any base-board serial devices from the RX0 line into the CPU.

### Option: JTAG In Circuit Emulation (ICE)

This function is provided via 10-pin connector J3.

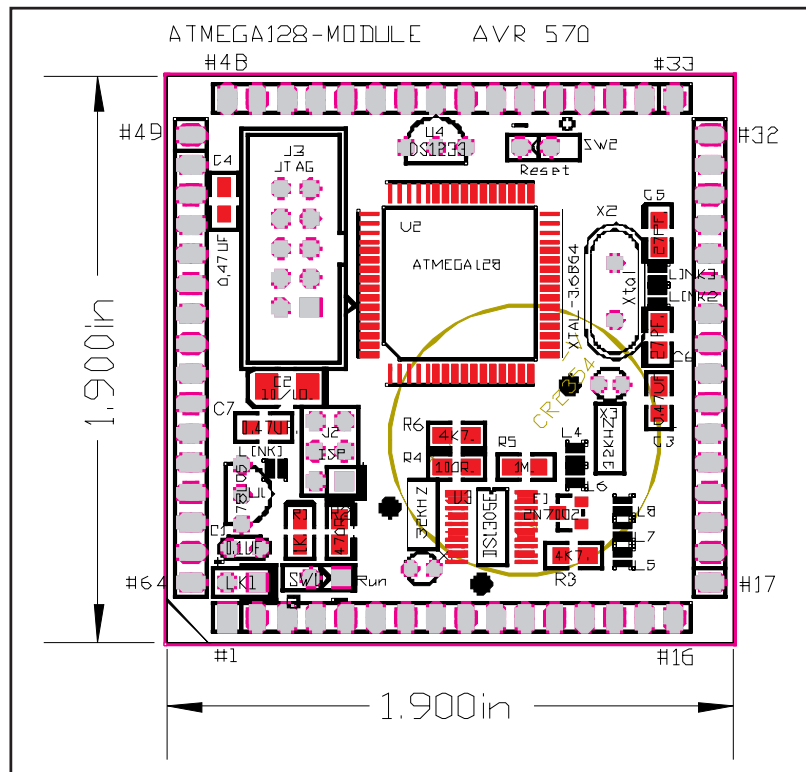
A cable from here connects to the Atmel JTAG ICE. The JTAG interface is a 4-wire Test Access Port (TAP) using the top 4 analog input pins ADC4 ... ADC7 and provides full device FLASH and EEPROM memory programming and on-chip debugging support, allowing real-time emulation of the micro controller while running in the target system. It gives the user complete control of the internal resources of the AVR CPU.

Its use reduces the number of analog channels available during testing. This connector is across the four lines mentioned, and even if the connector is installed, the analog lines are available on the edge connectors pins 54 ... 57. User's should not drive these analog data lines while JTAG ICE is in use. A convenient way might be to have series resistors which are removed during debug.

### Option: DS1305 Real-Time-Clock

An optional RTC can be supplied, installed on the AVR570, along with its associated 32,768 Hz crystal. The alarm output of the RTC can be connected (via link 5) to pin edge pin 18, and then on the base board, an inversion FET can drive the gate of series power P- FET. This is to enable systems to be built which automatically power up under control of the alarm system in the RTC.

The RTC uses the Serial Peripheral Interface (SPI) port on the ATmega128 CPU port pins PB1, PB2 and PB3, and if the RTC is used, users should reserve these port pins for



AVR570 ATmega128 CPU module with optional Real Time Clock

SPI functions on their base board. User hardware should enable the SPI functions for SPI peripherals as needed and disable the CE or CE\* pins on their SPI devices when not in use so the RTC can use the SPI port when it needs to. The CE pin (high true) enable for the RTC to use the SPI port is actually pin 18 of the CPU, which is Port pin PG3. Link 8 must be linked to connect PG3 to the RTC CE. (The RTC CE pin is pulled low by a resistor so the RTC communications is inactive during device programming and before the port G is enabled as an output and set low.)

A FET is installed with this option in parallel with the alarm output of the RTC, and this is driven from CPU pin 19, Port pin PG4. This allows the CPU to force the Alarm\* output low (active) on edge pin 18 and so allow alarms to be set for some future time while the CPU is kept active, and power is then turned OFF by taking Port pin PG4 LOW. This FET's gate is also pulled low by a resistor when PG4 is not initialised.

### Option: 5 volt power supply regulator

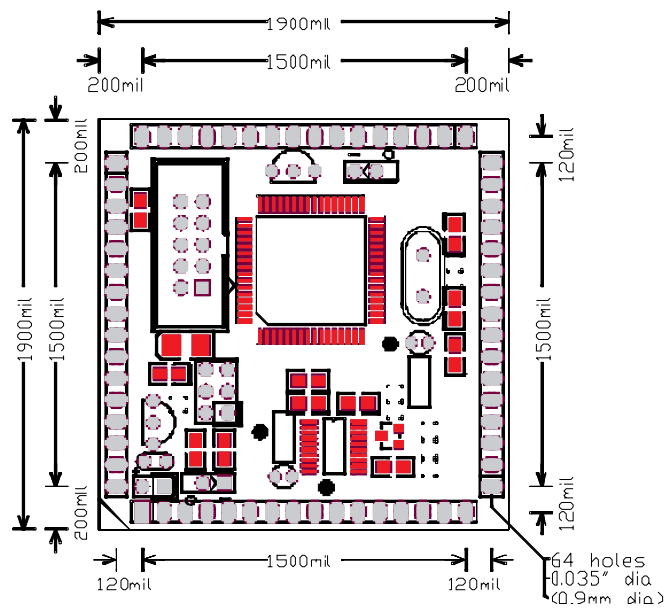
To maintain compatibility with the Olimex ATmega103 card, there is provision for an optional 5 volt regulator on the AVR570. This is not recommended unless a very small system is being built, as the off-board logic should be run from the same 5-volt power supply as the CPU, and the TO92 packaged device has only limited dissipation. A 78L05 or a low dropout device can be used.

If used, Link 1 should be connected, and Vcc is available to the base board via pins 21 and 52.

If not used, Vcc is fed to the module via pins 21 and 52.

Pin Number	Pin Name
Pin 1	CPU-Pin1: PEN*
Pin 2	CPU-Pin 2: PEO via SW1/RXD0
Pin 3	CPU-Pin 3: PE1/TXD0
Pin 4	CPU-Pin 4: PE2/XCK0
Pin 5	CPU-Pin 5: PE3/OC3A
Pin 6	CPU-Pin 6: PE4/OC3B
Pin 7	CPU-Pin 7: PE5/OC3C
Pin 8	CPU-Pin 8: PE6/T3/INT6
Pin 9	CPU-Pin 9: PE7/IC3/INT7
Pin 10	CPU-Pin 10: PBO/SS*
Pin 11	CPU-Pin 11: PB1/SCK
Pin 12	CPU-Pin 12: PB2/MOSI
Pin 13	CPU-Pin 13: PB3/MISO
Pin 14	CPU-Pin 14: PB4/OC0
Pin 15	CPU-Pin 15: PB5/OC1A
Pin 16	CPU-Pin 16: PB6/OC1B
Pin 17	CPU-Pin 17: PB7/OC1C
Pin 18	CPU-Pin 18:PG3/RTC Int0 out for alarm
Pin 19	CPU-Pin 19: PG4/Batt.
Pin 20	CPU-Pin 20: RESET*
Pin 21	CPU-Pin 21: Vcc
Pin 22	CPU-Pin 22: Ground
Pin 23	CPU-Pin 23: X2/nc
Pin 24	CPU-Pin 24: X1/nc
Pin 25	CPU-Pin 25: PD0/SCL/INT0*
Pin 26	CPU-Pin 26: PD1/SDA/INT1*
Pin 27	CPU-Pin 27: PD2/RXD1/INT2*
Pin 28	CPU-Pin 28: PD3/TXD1/INT3*
Pin 29	CPU-Pin 29: PD4/IC1
Pin 30	CPU-Pin 30: PD5/XCK1
Pin 31	CPU-Pin 31: PD6/T1
Pin 32	CPU-Pin 32: PD7/T2

Pin Number	Pin Name
Pin 33	CPU-Pin 33: WR*/PG0
Pin 34	CPU-Pin 34: RD*/PG1
Pin 35	CPU-Pin 35: PC0/A8
Pin 36	CPU-Pin 36: PC1/A9
Pin 37	CPU-Pin 37: PC2/A10
Pin 38	CPU-Pin 38: PC3/A11
Pin 39	CPU-Pin 39: PC4/A12
Pin 40	CPU-Pin 40: PC5/A13
Pin 41	CPU-Pin 41: PC6/A14
Pin 42	CPU-Pin 42: PC7/A15
Pin 43	CPU-Pin 43: ALE/PG2
Pin 44	CPU-Pin 44: PA7/AD7
Pin 45	CPU-Pin 45: PA6/AD6
Pin 46	CPU-Pin 46: PA5/AD5
Pin 47	CPU-Pin 47: PA4/AD4
Pin 48	CPU-Pin 48: PA3/AD3
Pin 49	CPU-Pin 49: PA2/AD2
Pin 50	CPU-Pin 50: PA1/AD1
Pin 51	CPU-Pin 51: PA0/AD0
Pin 52	CPU-Pin 52: VCC
Pin 53	CPU-Pin 53: GND
Pin 54	CPU-Pin 54: PF7/ADC7
Pin 55	CPU-Pin 55: PF6/ADC6
Pin 56	CPU-Pin 56: PF5/ADC5
Pin 57	CPU-Pin 57: PF4/ADC4
Pin 58	CPU-Pin 58: PF3/ADC3
Pin 59	CPU-Pin 59: PF2/ADC2
Pin 60	CPU-Pin 60: PF1/ADC1
Pin 61	CPU-Pin 61: PF0/ADC0
Pin 62	CPU-Pin 62: AREF
Pin 63	CPU-Pin 63: AGND
Pin 64	CPU-Pin 64: AVCC



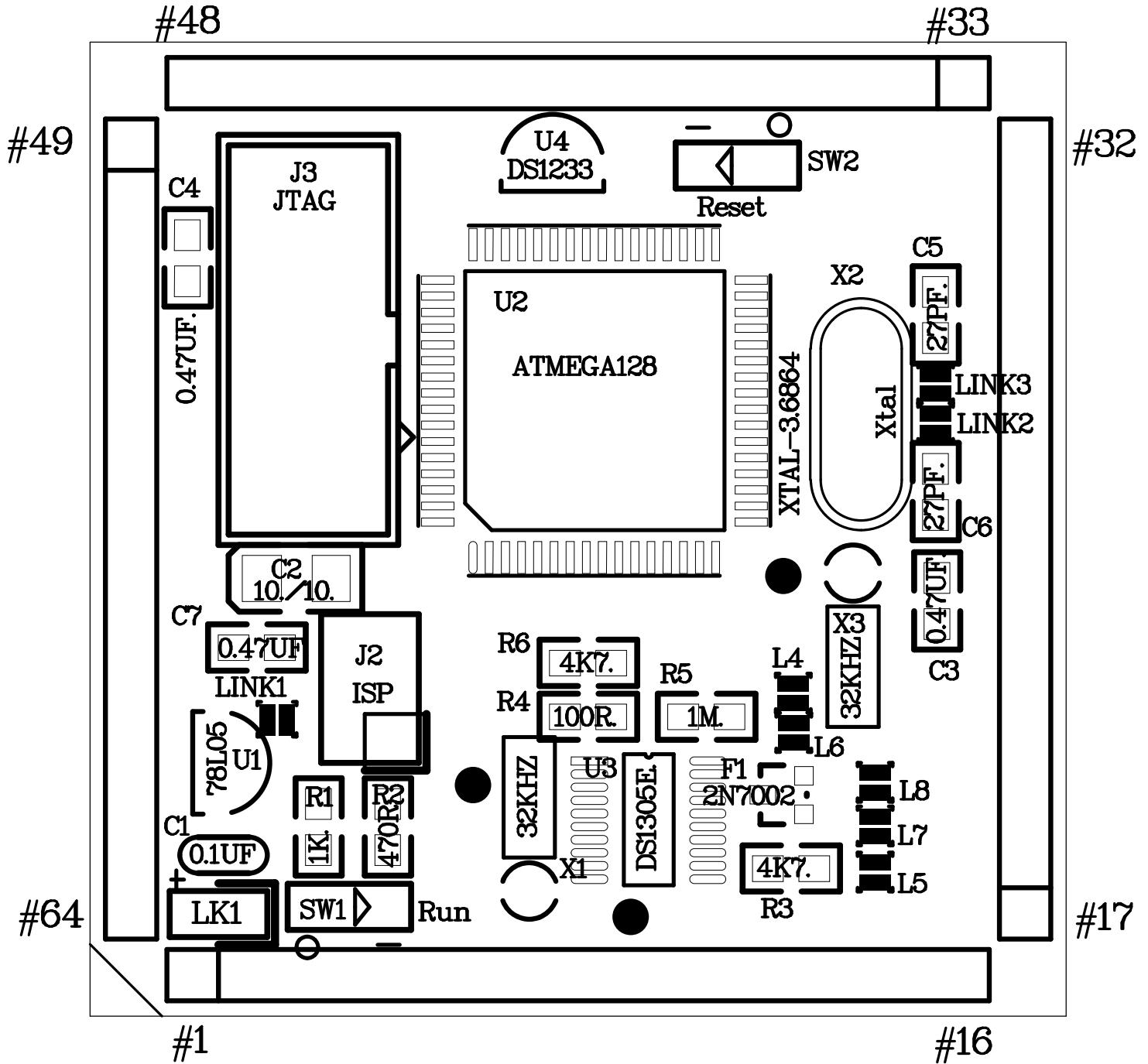
570 dimensions  
Clearance & hole positions

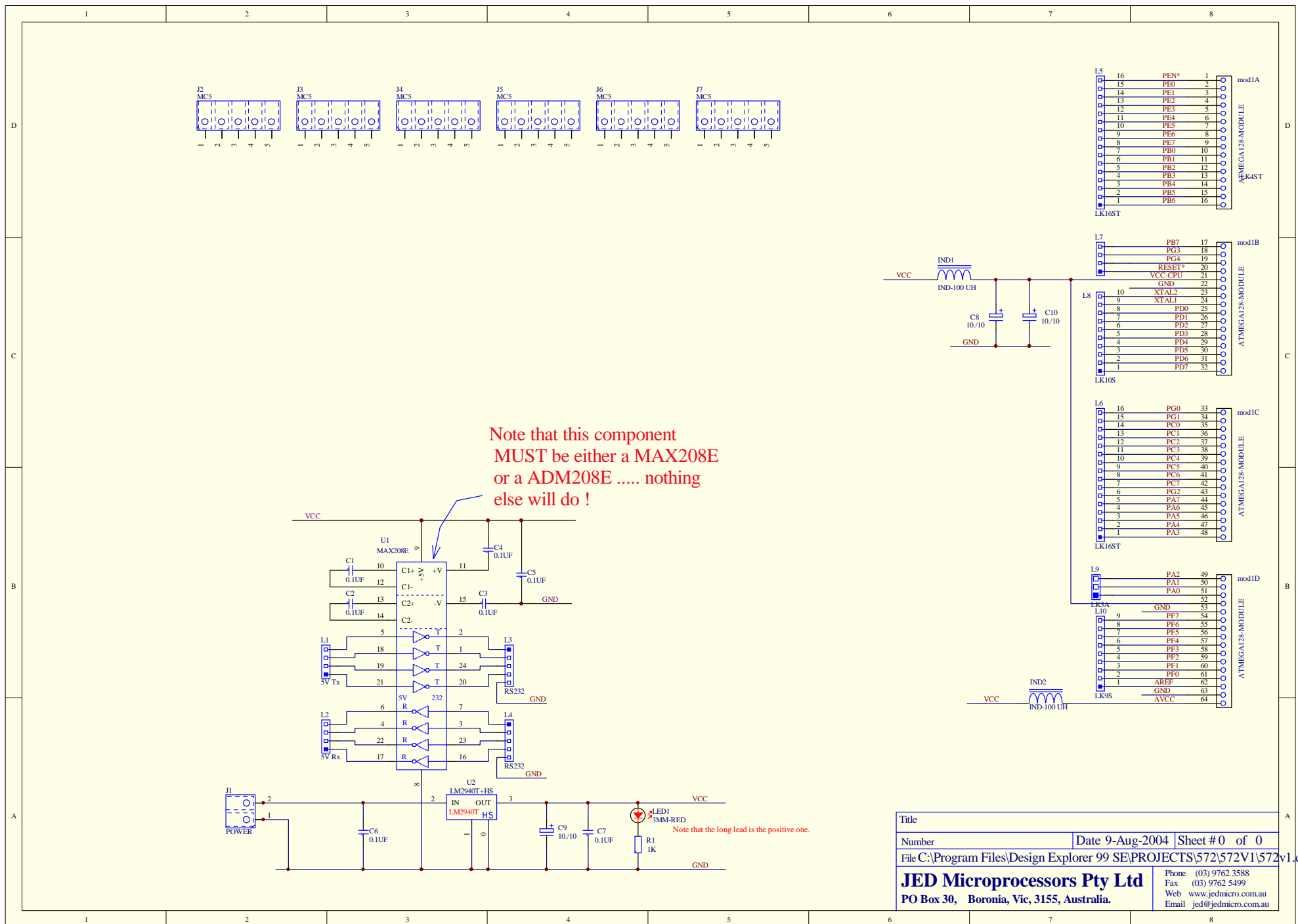


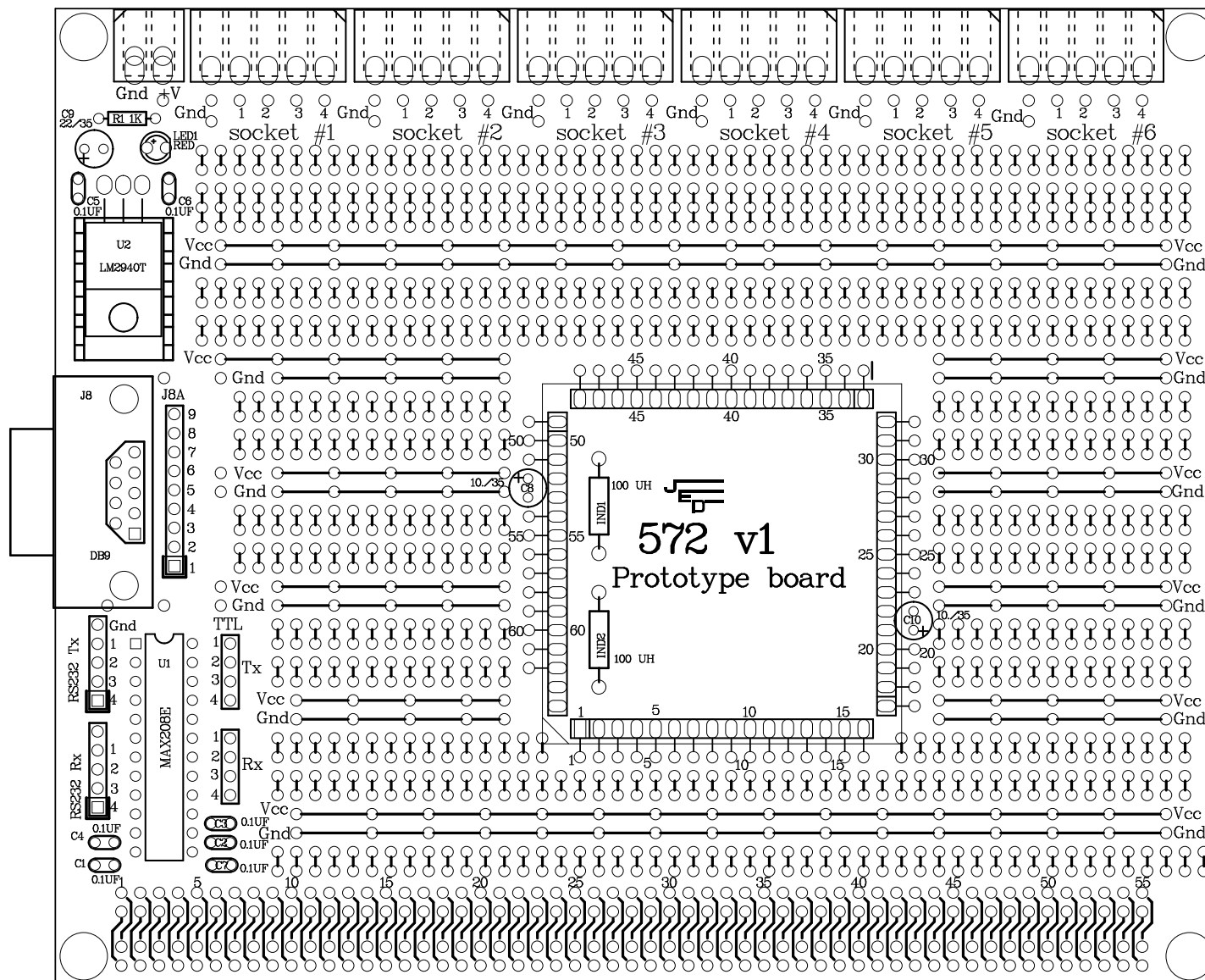


# ATMEGA128-MODULE

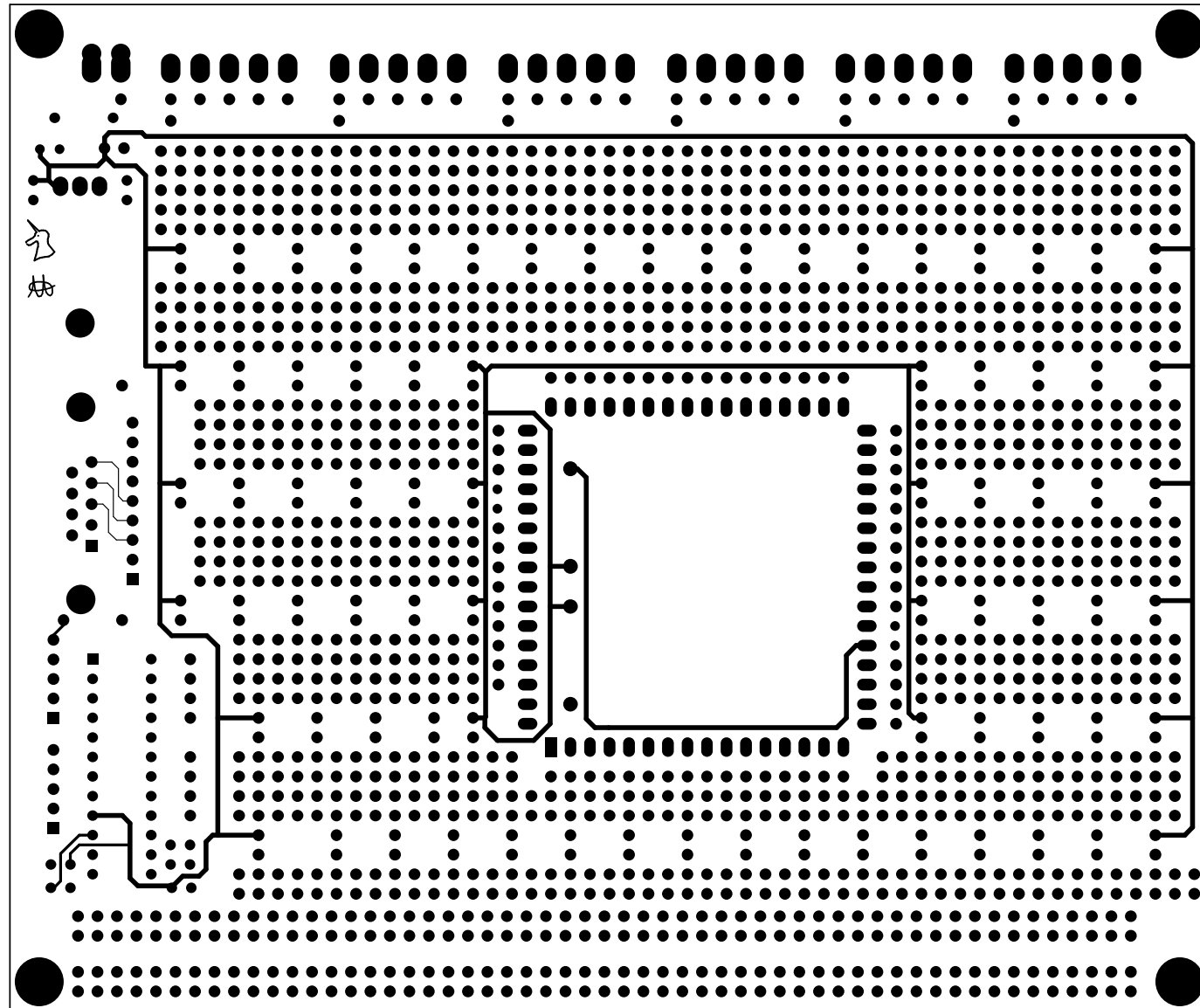
AVR 570



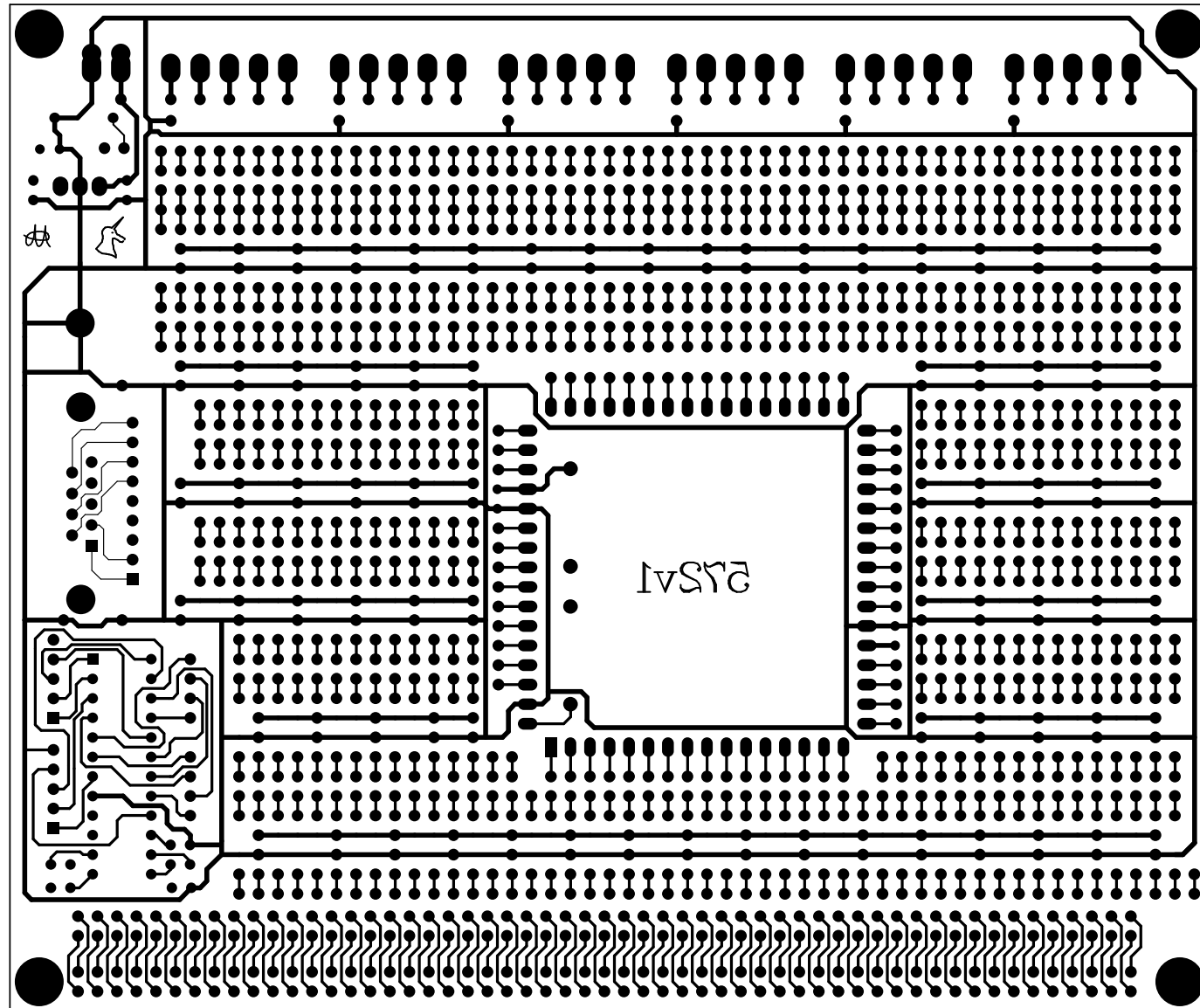




JED MICROPROCESSORS PTY LTD  
 BORONIA VICTORIA 3155  
 Board title C:\Program Files\Design Explorer 99 SE\I  
 Layer Mechanical Layer 1  
 Date 9-Aug-2004



JED MICROPROCESSORS PTY LTD  
BORONIA VICTORIA 3155  
Board title C:\Program Files\Design Explorer 99 SE\I  
Layer Mechanical Layer 1  
Date 9-Aug-2004



JED MICROPROCESSORS PTY LTD  
 BORONIA VICTORIA 3155  
 Board title C:\Program Files\Design Explorer 99 SE\I  
 Layer Mechanical Layer 1  
 Date 9-Aug-2004

---

## C.2 AVR ATmega128 Datasheet

## Features

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 133 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers + Peripheral Control Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-chip 2-cycle Multiplier
- Nonvolatile Program and Data Memories
  - 128K Bytes of In-System Reprogrammable Flash  
Endurance: 10,000 Write/Erase Cycles
  - Optional Boot Code Section with Independent Lock Bits  
In-System Programming by On-chip Boot Program  
True Read-While-Write Operation
  - 4K Bytes EEPROM  
Endurance: 100,000 Write/Erase Cycles
  - 4K Bytes Internal SRAM
  - Up to 64K Bytes Optional External Memory Space
  - Programming Lock for Software Security
  - SPI Interface for In-System Programming
- JTAG (IEEE std. 1149.1 Compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  - Two Expanded 16-bit Timer/Counters with Separate Prescaler, Compare Mode and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Two 8-bit PWM Channels
  - 6 PWM Channels with Programmable Resolution from 2 to 16 Bits
  - Output Compare Modulator
  - 8-channel, 10-bit ADC
    - 8 Single-ended Channels
    - 7 Differential Channels
    - 2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
  - Byte-oriented Two-wire Serial Interface
  - Dual Programmable Serial USARTs
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
  - Software Selectable Clock Frequency
  - ATmega103 Compatibility Mode Selected by a Fuse
  - Global Pull-up Disable
- I/O and Packages
  - 53 Programmable I/O Lines
  - 64-lead TQFP and 64-pad MLF
- Operating Voltages
  - 2.7 - 5.5V for ATmega128L
  - 4.5 - 5.5V for ATmega128
- Speed Grades
  - 0 - 8 MHz for ATmega128L
  - 0 - 16 MHz for ATmega128



## 8-bit AVR® Microcontroller with 128K Bytes In-System Programmable Flash

ATmega128  
ATmega128L

## Summary

Rev. 2467MS-AVR-11/04

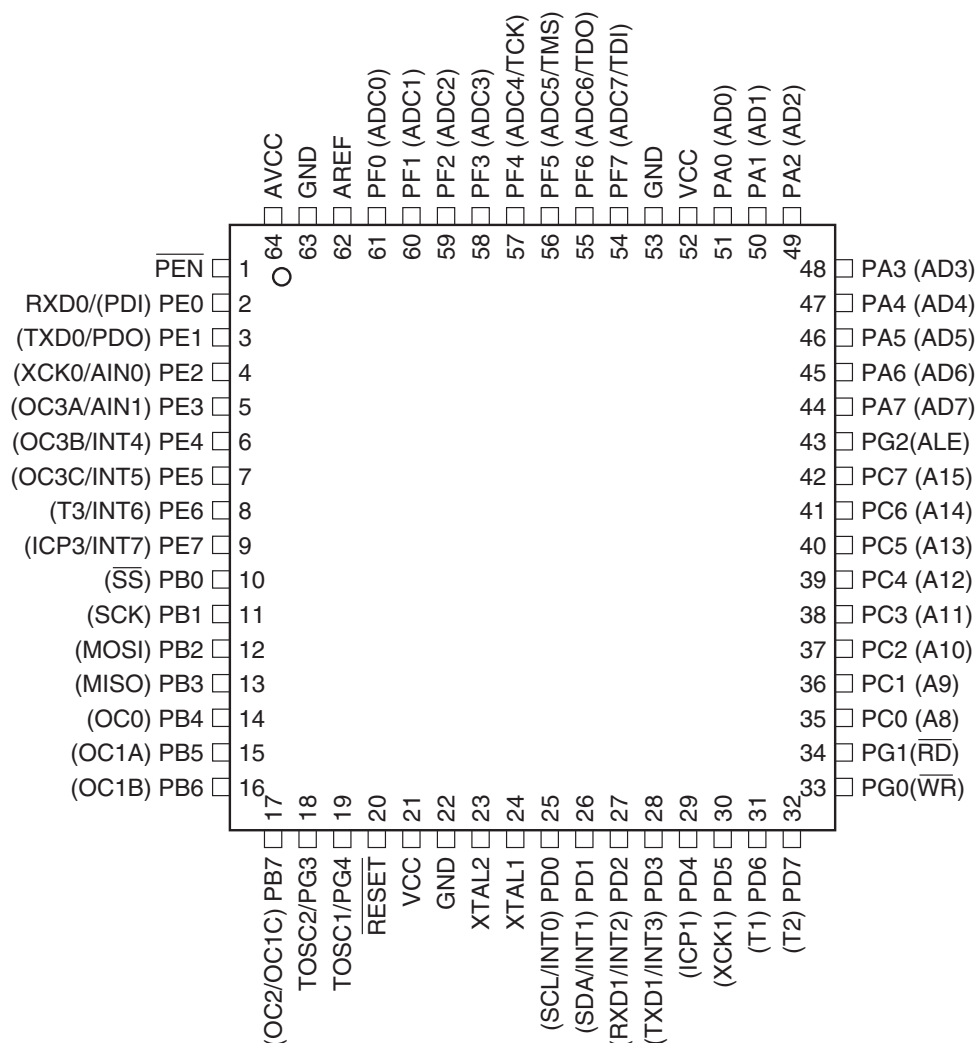


Note: This is a summary document. A complete document is available on our Web site at [www.atmel.com](http://www.atmel.com).



## Pin Configurations

Figure 1. Pinout ATmega128

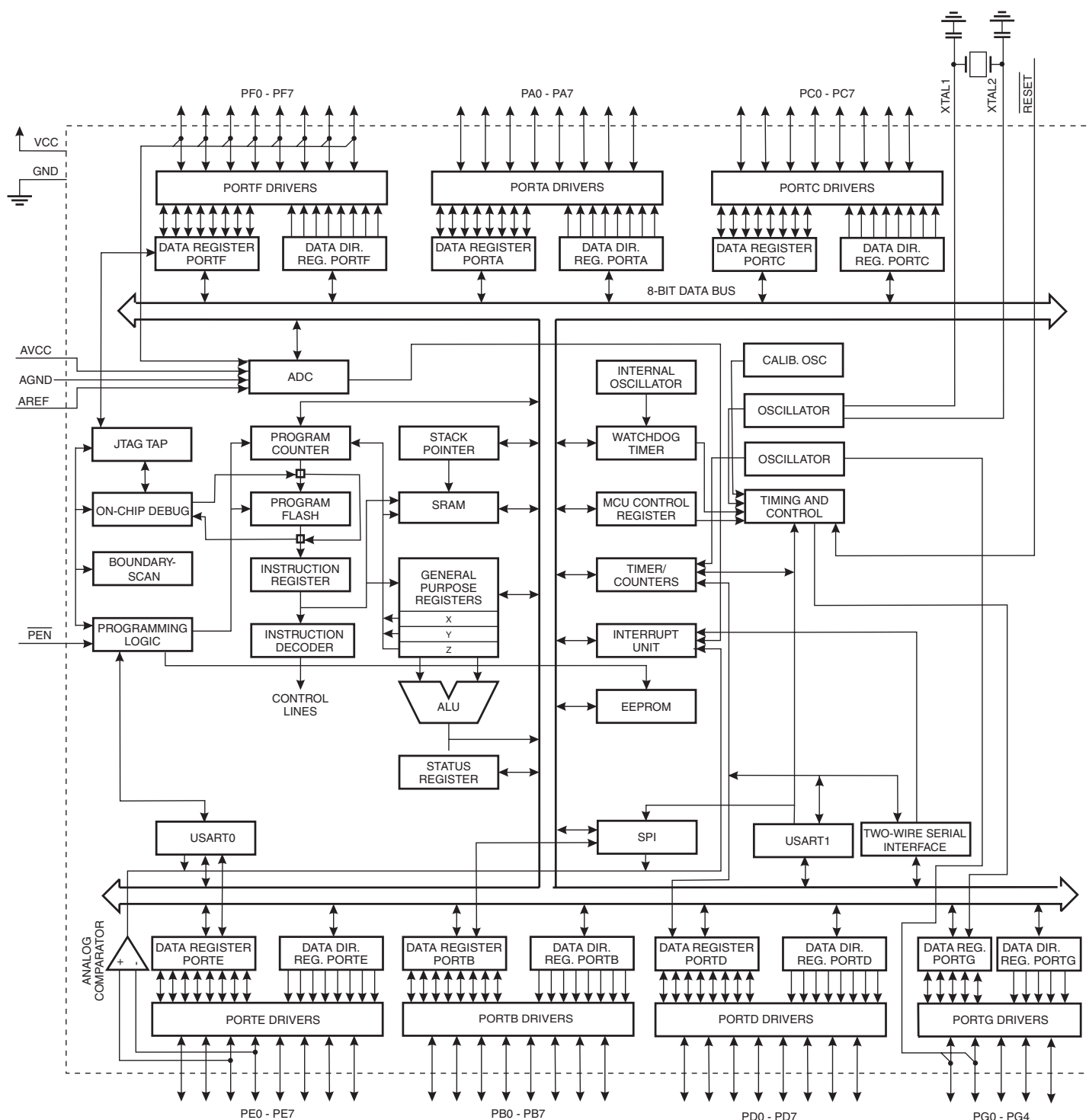


Note: The bottom pad under the MLF package should be soldered to ground.

## Overview

The ATmega128 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega128 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

**Figure 2. Block Diagram**





The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega128 provides the following features: 128K bytes of In-System Programmable Flash with Read-While-Write capabilities, 4K bytes EEPROM, 4K bytes SRAM, 53 general purpose I/O lines, 32 general purpose working registers, Real Time Counter (RTC), four flexible Timer/Counters with compare modes and PWM, 2 USARTs, a byte oriented Two-wire Serial Interface, an 8-channel, 10-bit ADC with optional differential input stage with programmable gain, programmable Watchdog Timer with Internal Oscillator, an SPI serial port, IEEE std. 1149.1 compliant JTAG test interface, also used for accessing the On-chip Debug system and programming and six software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or Hardware Reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except Asynchronous Timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the Crystal/Resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption. In Extended Standby mode, both the main Oscillator and the Asynchronous Timer continue to run.

The device is manufactured using Atmel's high-density nonvolatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed in-system through an SPI serial interface, by a conventional nonvolatile memory programmer, or by an On-chip Boot program running on the AVR core. The boot program can use any interface to download the application program in the application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega128 is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The ATmega128 AVR is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators, in-circuit emulators, and evaluation kits.

## **ATmega103 and ATmega128 Compatibility**

The ATmega128 is a highly complex microcontroller where the number of I/O locations supersedes the 64 I/O locations reserved in the AVR instruction set. To ensure backward compatibility with the ATmega103, all I/O locations present in ATmega103 have the same location in ATmega128. Most additional I/O locations are added in an Extended I/O space starting from \$60 to \$FF, (i.e., in the ATmega103 internal RAM space). These locations can be reached by using LD/LDS/LDD and ST/STS/STD instructions only, not by using IN and OUT instructions. The relocation of the internal RAM space may still be a problem for ATmega103 users. Also, the increased number of interrupt vectors might be a problem if the code uses absolute addresses. To solve these problems, an ATmega103 compatibility mode can be selected by programming the fuse M103C. In this mode, none of the functions in the Extended I/O space are in use, so the internal RAM is located as in ATmega103. Also, the Extended Interrupt vectors are removed.

The ATmega128 is 100% pin compatible with ATmega103, and can replace the ATmega103 on current Printed Circuit Boards. The application note “Replacing ATmega103 by ATmega128” describes what the user should be aware of replacing the ATmega103 by an ATmega128.

#### **ATmega103 Compatibility Mode**

By programming the M103C fuse, the ATmega128 will be compatible with the ATmega103 regards to RAM, I/O pins and interrupt vectors as described above. However, some new features in ATmega128 are not available in this compatibility mode, these features are listed below:

- One USART instead of two, Asynchronous mode only. Only the eight least significant bits of the Baud Rate Register is available.
- One 16 bits Timer/Counter with two compare registers instead of two 16-bit Timer/Counters with three compare registers.
- Two-wire serial interface is not supported.
- Port C is output only.
- Port G serves alternate functions only (not a general I/O port).
- Port F serves as digital input only in addition to analog input to the ADC.
- Boot Loader capabilities is not supported.
- It is not possible to adjust the frequency of the internal calibrated RC Oscillator.
- The External Memory Interface can not release any Address pins for general I/O, neither configure different wait-states to different External Memory Address sections.

In addition, there are some other minor differences to make it more compatible to ATmega103:

- Only EXTRF and PORF exists in MCUCSR.
- Timed sequence not required for Watchdog Time-out change.
- External Interrupt pins 3 - 0 serve as level interrupt only.
- USART has no FIFO buffer, so data overrun comes earlier.

Unused I/O bits in ATmega103 should be written to 0 to ensure same operation in ATmega128.

#### **Pin Descriptions**

<b>VCC</b>	Digital supply voltage.
<b>GND</b>	Ground.
<b>Port A (PA7..PA0)</b>	<p>Port A is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port A output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port A pins that are externally pulled low will source current if the pull-up resistors are activated. The Port A pins are tri-stated when a reset condition becomes active, even if the clock is not running.</p> <p>Port A also serves the functions of various special features of the ATmega128 as listed on page 70.</p>
<b>Port B (PB7..PB0)</b>	<p>Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source</p>

current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port B also serves the functions of various special features of the ATmega128 as listed on page 71.

#### Port C (PC7..PC0)

Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port C also serves the functions of special features of the ATmega128 as listed on page 74. In ATmega103 compatibility mode, Port C is output only, and the port C pins are **not** tri-stated when a reset condition becomes active.

Note: The ATmega128 is by default shipped in ATmega103 compatibility mode. Thus, if the parts are not programmed before they are put on the PCB, PORTC will be output during first power up, and until the ATmega103 compatibility mode is disabled.

#### Port D (PD7..PD0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port D also serves the functions of various special features of the ATmega128 as listed on page 75.

#### Port E (PE7..PE0)

Port E is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port E output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port E pins that are externally pulled low will source current if the pull-up resistors are activated. The Port E pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port E also serves the functions of various special features of the ATmega128 as listed on page 78.

#### Port F (PF7..PF0)

Port F serves as the analog inputs to the A/D Converter.

Port F also serves as an 8-bit bi-directional I/O port, if the A/D Converter is not used. Port pins can provide internal pull-up resistors (selected for each bit). The Port F output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port F pins that are externally pulled low will source current if the pull-up resistors are activated. The Port F pins are tri-stated when a reset condition becomes active, even if the clock is not running. If the JTAG interface is enabled, the pull-up resistors on pins PF7(TDI), PF5(TMS), and PF4(TCK) will be activated even if a Reset occurs.

The TDO pin is tri-stated unless TAP states that shift out data are entered.

Port F also serves the functions of the JTAG interface.

In ATmega103 compatibility mode, Port F is an input Port only.

#### Port G (PG4..PG0)

Port G is a 5-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port G output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port G pins that are externally pulled low will source

current if the pull-up resistors are activated. The Port G pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port G also serves the functions of various special features.

The port G pins are tri-stated when a reset condition becomes active, even if the clock is not running.

In ATmega103 compatibility mode, these pins only serves as strobes signals to the external memory as well as input to the 32 kHz Oscillator, and the pins are initialized to PG0 = 1, PG1 = 1, and PG2 = 0 asynchronously when a reset condition becomes active, even if the clock is not running. PG3 and PG4 are oscillator pins.

**RESET**

Reset input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running. The minimum pulse length is given in Table 19 on page 48. Shorter pulses are not guaranteed to generate a reset.

**XTAL1**

Input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

**XTAL2**

Output from the inverting Oscillator amplifier.

**AVCC**

AVCC is the supply voltage pin for Port F and the A/D Converter. It should be externally connected to  $V_{CC}$ , even if the ADC is not used. If the ADC is used, it should be connected to  $V_{CC}$  through a low-pass filter.

**AREF**

AREF is the analog reference pin for the A/D Converter.

**PEN**

PEN is a programming enable pin for the SPI Serial Programming mode, and is internally pulled high . By holding this pin low during a Power-on Reset, the device will enter the SPI Serial Programming mode.  $\overline{PEN}$  has no function during normal operation.

## Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(\$FF)	Reserved	–	–	–	–	–	–	–	–	
..	Reserved	–	–	–	–	–	–	–	–	
(\$9E)	Reserved	–	–	–	–	–	–	–	–	
(\$9D)	UCSR1C	–	UMSEL1	UPM11	UPM10	USBS1	UCSZ11	UCSZ10	UCPOL1	191
(\$9C)	UDR1	USART1 I/O Data Register								189
(\$9B)	UCSR1A	RXC1	TXC1	UDRE1	FE1	DOR1	UPE1	U2X1	MPCM1	189
(\$9A)	UCSR1B	RXCIE1	TXCIE1	UDRIE1	RXEN1	TXEN1	UCSZ12	RXB81	TXB81	190
(\$99)	UBRR1L	USART1 Baud Rate Register Low								193
(\$98)	UBRR1H	–	–	–	–	USART1 Baud Rate Register High				193
(\$97)	Reserved	–	–	–	–	–	–	–	–	
(\$96)	Reserved	–	–	–	–	–	–	–	–	
(\$95)	UCSR0C	–	UMSEL0	UPM01	UPM00	USBS0	UCSZ01	UCSZ00	UCPOL0	191
(\$94)	Reserved	–	–	–	–	–	–	–	–	
(\$93)	Reserved	–	–	–	–	–	–	–	–	
(\$92)	Reserved	–	–	–	–	–	–	–	–	
(\$91)	Reserved	–	–	–	–	–	–	–	–	
(\$90)	UBRR0H	–	–	–	–	USART0 Baud Rate Register High				193
(\$8F)	Reserved	–	–	–	–	–	–	–	–	
(\$8E)	Reserved	–	–	–	–	–	–	–	–	
(\$8D)	Reserved	–	–	–	–	–	–	–	–	
(\$8C)	TCCR3C	FOC3A	FOC3B	FOC3C	–	–	–	–	–	135
(\$8B)	TCCR3A	COM3A1	COM3A0	COM3B1	COM3B0	COM3C1	COM3C0	WGM31	WGM30	131
(\$8A)	TCCR3B	ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30	134
(\$89)	TCNT3H	Timer/Counter3 – Counter Register High Byte								136
(\$88)	TCNT3L	Timer/Counter3 – Counter Register Low Byte								136
(\$87)	OCR3AH	Timer/Counter3 – Output Compare Register A High Byte								136
(\$86)	OCR3AL	Timer/Counter3 – Output Compare Register A Low Byte								136
(\$85)	OCR3BH	Timer/Counter3 – Output Compare Register B High Byte								137
(\$84)	OCR3BL	Timer/Counter3 – Output Compare Register B Low Byte								137
(\$83)	OCR3CH	Timer/Counter3 – Output Compare Register C High Byte								137
(\$82)	OCR3CL	Timer/Counter3 – Output Compare Register C Low Byte								137
(\$81)	ICR3H	Timer/Counter3 – Input Capture Register High Byte								137
(\$80)	ICR3L	Timer/Counter3 – Input Capture Register Low Byte								137
(\$7F)	Reserved	–	–	–	–	–	–	–	–	
(\$7E)	Reserved	–	–	–	–	–	–	–	–	
(\$7D)	ETIMSK	–	–	TICIE3	OCIE3A	OCIE3B	TOIE3	OCIE3C	OCIE1C	138
(\$7C)	ETIFR	–	–	ICF3	OCF3A	OCF3B	TOV3	OCF3C	OCF1C	139
(\$7B)	Reserved	–	–	–	–	–	–	–	–	
(\$7A)	TCCR1C	FOC1A	FOC1B	FOC1C	–	–	–	–	–	135
(\$79)	OCR1CH	Timer/Counter1 – Output Compare Register C High Byte								136
(\$78)	OCR1CL	Timer/Counter1 – Output Compare Register C Low Byte								136
(\$77)	Reserved	–	–	–	–	–	–	–	–	
(\$76)	Reserved	–	–	–	–	–	–	–	–	
(\$75)	Reserved	–	–	–	–	–	–	–	–	
(\$74)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	206
(\$73)	TWDR	Two-wire Serial Interface Data Register								208
(\$72)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	208
(\$71)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	207
(\$70)	TWBR	Two-wire Serial Interface Bit Rate Register								206
(\$6F)	OSCCAL	Oscillator Calibration Register								39
(\$6E)	Reserved	–	–	–	–	–	–	–	–	
(\$6D)	XMCR	–	SRL2	SRL1	SRL0	SRW01	SRW00	SRW11	–	29
(\$6C)	XMCRB	XMBK	–	–	–	–	XMM2	XMM1	XMM0	31
(\$6B)	Reserved	–	–	–	–	–	–	–	–	
(\$6A)	EICRA	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	87
(\$69)	Reserved	–	–	–	–	–	–	–	–	
(\$68)	SPMCSR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	279
(\$67)	Reserved	–	–	–	–	–	–	–	–	
(\$66)	Reserved	–	–	–	–	–	–	–	–	
(\$65)	PORTG	–	–	–	PORTG4	PORTG3	PORTG2	PORTG1	PORTG0	86
(\$64)	DDRG	–	–	–	DDG4	DDG3	DDG2	DDG1	DDG0	86
(\$63)	PING	–	–	–	PING4	PING3	PING2	PING1	PING0	86
(\$62)	PORTF	PORTF7	PORTF6	PORTF5	PORTF4	PORTF3	PORTF2	PORTF1	PORTF0	85

## Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(\$61)	DDRF	DDF7	DDF6	DDF5	DDF4	DDF3	DDF2	DDF1	DDF0	86
(\$60)	Reserved	–	–	–	–	–	–	–	–	
\$3F (\$5F)	SREG	I	T	H	S	V	N	Z	C	9
\$3E (\$5E)	SPH	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	12
\$3D (\$5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	12
\$3C (\$5C)	XDIV	XDIVEN	XDIV6	XDIV5	XDIV4	XDIV3	XDIV2	XDIV1	XDIV0	41
\$3B (\$5B)	RAMPZ	–	–	–	–	–	–	–	RAMPZ0	12
\$3A (\$5A)	EICRB	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	88
\$39 (\$59)	EIMSK	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	89
\$38 (\$58)	EIFR	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	INTF0	89
\$37 (\$57)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	106, 138, 158
\$36 (\$56)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	106, 139, 158
\$35 (\$55)	MCUCR	SRE	SRW10	SE	SM1	SM0	SM2	IVSEL	IVCE	29, 42, 61
\$34 (\$54)	MCUCSR	JTD	–	–	JTRF	WDRF	BORF	EXTRF	PORF	51, 256
\$33 (\$53)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	101
\$32 (\$52)	TCNT0	Timer/Counter0 (8 Bit)								103
\$31 (\$51)	OCR0	Timer/Counter0 Output Compare Register								103
\$30 (\$50)	ASSR	–	–	–	–	AS0	TCN0UB	OCR0UB	TCR0UB	104
\$2F (\$4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	131
\$2E (\$4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	134
\$2D (\$4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte								136
\$2C (\$4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte								136
\$2B (\$4B)	OCR1AH	Timer/Counter1 – Output Compare Register A High Byte								136
\$2A (\$4A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low Byte								136
\$29 (\$49)	OCR1BH	Timer/Counter1 – Output Compare Register B High Byte								136
\$28 (\$48)	OCR1BL	Timer/Counter1 – Output Compare Register B Low Byte								136
\$27 (\$47)	ICR1H	Timer/Counter1 – Input Capture Register High Byte								137
\$26 (\$46)	ICR1L	Timer/Counter1 – Input Capture Register Low Byte								137
\$25 (\$45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	156
\$24 (\$44)	TCNT2	Timer/Counter2 (8 Bit)								158
\$23 (\$43)	OCR2	Timer/Counter2 Output Compare Register								158
\$22 (\$42)	OCDR	IDRD/OCDR7	OCDR6	OCDR5	OCDR4	OCDR3	OCDR2	OCDR1	OCDR0	253
\$21 (\$41)	WDTCR	–	–	–	WDCE	WDE	WDP2	WDP1	WDP0	53
\$20 (\$40)	SFIOR	TSM	–	–	–	ACME	PUD	PSR0	PSR321	70, 107, 143, 228
\$1F (\$3F)	EEARH	–	–	–	–	EEPROM Address Register High				19
\$1E (\$3E)	EEARL	EEPROM Address Register Low Byte								19
\$1D (\$3D)	EEDR	EEPROM Data Register								20
\$1C (\$3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE	20
\$1B (\$3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	84
\$1A (\$3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	84
\$19 (\$39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	84
\$18 (\$38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	84
\$17 (\$37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	84
\$16 (\$36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	84
\$15 (\$35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	84
\$14 (\$34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	84
\$13 (\$33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	85
\$12 (\$32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	85
\$11 (\$31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	85
\$10 (\$30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	85
\$0F (\$2F)	SPDR	SPI Data Register								168
\$0E (\$2E)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	168
\$0D (\$2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	166
\$0C (\$2C)	UDR0	USART0 I/O Data Register								189
\$0B (\$2B)	UCSR0A	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0	189
\$0A (\$2A)	UCSR0B	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80	190
\$09 (\$29)	UBRR0L	USART0 Baud Rate Register Low								193
\$08 (\$28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	228
\$07 (\$27)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	244
\$06 (\$26)	ADCSRA	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	245
\$05 (\$25)	ADCH	ADC Data Register High Byte								246
\$04 (\$24)	ADCL	ADC Data Register Low byte								246
\$03 (\$23)	PORTE	PORTE7	PORTE6	PORTE5	PORTE4	PORTE3	PORTE2	PORTE1	PORTE0	85
\$02 (\$22)	DDRE	DDE7	DDE6	DDE5	DDE4	DDE3	DDE2	DDE1	DDE0	85



## Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
\$01 (\$21)	PINE	PINE7	PINE6	PINE5	PINE4	PINE3	PINE2	PINE1	PINE0	85
\$00 (\$20)	PINF	PINF7	PINF6	PINF5	PINF4	PINF3	PINF2	PINF1	PINF0	86

- Notes:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
  2. Some of the status flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers \$00 to \$1F only.

## Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
<b>ARITHMETIC AND LOGIC INSTRUCTIONS</b>					
ADD	Rd, Rr	Add two Registers	$Rd \leftarrow Rd + Rr$	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	$Rd \leftarrow Rd + Rr + C$	Z,C,N,V,H	1
ADIW	RdI,K	Add Immediate to Word	$RdH:RdL \leftarrow RdH:RdL + K$	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	$Rd \leftarrow Rd - Rr$	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	$Rd \leftarrow Rd - K$	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	$Rd \leftarrow Rd - Rr - C$	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	$Rd \leftarrow Rd - K - C$	Z,C,N,V,H	1
SBIW	RdI,K	Subtract Immediate from Word	$RdH:RdL \leftarrow RdH:RdL - K$	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	$Rd \leftarrow Rd \bullet Rr$	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	$Rd \leftarrow Rd \bullet K$	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	$Rd \leftarrow Rd \vee Rr$	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	$Rd \leftarrow Rd \vee K$	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	$Rd \leftarrow Rd \oplus Rr$	Z,N,V	1
COM	Rd	One's Complement	$Rd \leftarrow \$FF - Rd$	Z,C,N,V	1
NEG	Rd	Two's Complement	$Rd \leftarrow \$00 - Rd$	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	$Rd \leftarrow Rd \vee K$	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	$Rd \leftarrow Rd \bullet (\$FF - K)$	Z,N,V	1
INC	Rd	Increment	$Rd \leftarrow Rd + 1$	Z,N,V	1
DEC	Rd	Decrement	$Rd \leftarrow Rd - 1$	Z,N,V	1
TST	Rd	Test for Zero or Minus	$Rd \leftarrow Rd \bullet Rd$	Z,N,V	1
CLR	Rd	Clear Register	$Rd \leftarrow Rd \oplus Rd$	Z,N,V	1
SER	Rd	Set Register	$Rd \leftarrow \$FF$	None	1
MUL	Rd, Rr	Multiply Unsigned	$R1:R0 \leftarrow Rd \times Rr$	Z,C	2
MULS	Rd, Rr	Multiply Signed	$R1:R0 \leftarrow Rd \times Rr$	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	$R1:R0 \leftarrow Rd \times Rr$	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	$R1:R0 \leftarrow (Rd \times Rr) << 1$	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	$R1:R0 \leftarrow (Rd \times Rr) << 1$	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	$R1:R0 \leftarrow (Rd \times Rr) << 1$	Z,C	2
<b>BRANCH INSTRUCTIONS</b>					
RJMP	k	Relative Jump	$PC \leftarrow PC + k + 1$	None	2
IJMP		Indirect Jump to (Z)	$PC \leftarrow Z$	None	2
JMP	k	Direct Jump	$PC \leftarrow k$	None	3
RCALL	k	Relative Subroutine Call	$PC \leftarrow PC + k + 1$	None	3
ICALL		Indirect Call to (Z)	$PC \leftarrow Z$	None	3
CALL	k	Direct Subroutine Call	$PC \leftarrow k$	None	4
RET		Subroutine Return	$PC \leftarrow STACK$	None	4
RETI		Interrupt Return	$PC \leftarrow STACK$	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if $(Rd = Rr)$ $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
CP	Rd,Rr	Compare	$Rd - Rr$	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	$Rd - Rr - C$	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	$Rd - K$	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if $(Rr(b)=0)$ $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
SBRs	Rr, b	Skip if Bit in Register is Set	if $(Rr(b)=1)$ $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if $(P(b)=0)$ $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
SBS	P, b	Skip if Bit in I/O Register is Set	if $(P(b)=1)$ $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
BRBS	s, k	Branch if Status Flag Set	if $(SREG(s)=1)$ then $PC \leftarrow PC + k + 1$	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if $(SREG(s)=0)$ then $PC \leftarrow PC + k + 1$	None	1 / 2
BREQ	k	Branch if Equal	if $(Z = 1)$ then $PC \leftarrow PC + k + 1$	None	1 / 2
BRNE	k	Branch if Not Equal	if $(Z = 0)$ then $PC \leftarrow PC + k + 1$	None	1 / 2
BRCS	k	Branch if Carry Set	if $(C = 1)$ then $PC \leftarrow PC + k + 1$	None	1 / 2
BRCC	k	Branch if Carry Cleared	if $(C = 0)$ then $PC \leftarrow PC + k + 1$	None	1 / 2
BRSH	k	Branch if Same or Higher	if $(C = 0)$ then $PC \leftarrow PC + k + 1$	None	1 / 2
BRLO	k	Branch if Lower	if $(C = 1)$ then $PC \leftarrow PC + k + 1$	None	1 / 2
BRMI	k	Branch if Minus	if $(N = 1)$ then $PC \leftarrow PC + k + 1$	None	1 / 2
BRPL	k	Branch if Plus	if $(N = 0)$ then $PC \leftarrow PC + k + 1$	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if $(N \oplus V = 0)$ then $PC \leftarrow PC + k + 1$	None	1 / 2
BRLT	k	Branch if Less Than Zero, Signed	if $(N \oplus V = 1)$ then $PC \leftarrow PC + k + 1$	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if $(H = 1)$ then $PC \leftarrow PC + k + 1$	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if $(H = 0)$ then $PC \leftarrow PC + k + 1$	None	1 / 2
BRTS	k	Branch if T Flag Set	if $(T = 1)$ then $PC \leftarrow PC + k + 1$	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if $(T = 0)$ then $PC \leftarrow PC + k + 1$	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if $(V = 1)$ then $PC \leftarrow PC + k + 1$	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if $(V = 0)$ then $PC \leftarrow PC + k + 1$	None	1 / 2

## Instruction Set Summary (Continued)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1 / 2
<b>DATA TRANSFER INSTRUCTIONS</b>					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, -X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, -Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd, Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z + 1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	-X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	-Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q, Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q, Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z + 1	None	3
ELPM		Extended Load Program Memory	R0 ← (RAMPZ:Z)	None	3
ELPM	Rd, Z	Extended Load Program Memory	Rd ← (RAMPZ:Z)	None	3
ELPM	Rd, Z+	Extended Load Program Memory and Post-Inc	Rd ← (RAMPZ:Z), RAMPZ:Z ← RAMPZ:Z + 1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
<b>BIT AND BIT-TEST INSTRUCTIONS</b>					
SBI	P, b	Set Bit in I/O Register	I/O(P, b) ← 1	None	2
CBI	P, b	Clear Bit in I/O Register	I/O(P, b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z, C, N, V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z, C, N, V	1
ROL	Rd	Rotate Left Through Carry	Rd(0) ← C, Rd(n+1) ← Rd(n), C ← Rd(7)	Z, C, N, V	1
ROR	Rd	Rotate Right Through Carry	Rd(7) ← C, Rd(n) ← Rd(n+1), C ← Rd(0)	Z, C, N, V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z, C, N, V	1
SWAP	Rd	Swap Nibbles	Rd(3..0) ← Rd(7..4), Rd(7..4) ← Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1

## Instruction Set Summary (Continued)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
SEV		Set Twos Complement Overflow.	$V \leftarrow 1$	V	1
CLV		Clear Twos Complement Overflow	$V \leftarrow 0$	V	1
SET		Set T in SREG	$T \leftarrow 1$	T	1
CLT		Clear T in SREG	$T \leftarrow 0$	T	1
SEH		Set Half Carry Flag in SREG	$H \leftarrow 1$	H	1
CLH		Clear Half Carry Flag in SREG	$H \leftarrow 0$	H	1
<b>MCU CONTROL INSTRUCTIONS</b>					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-chip Debug Only	None	N/A

## Ordering Information

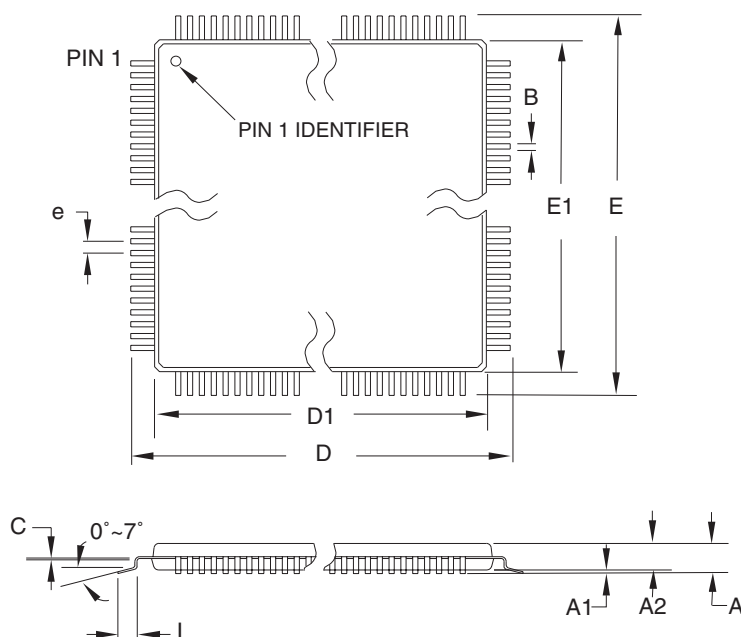
Speed (MHz)	Power Supply	Ordering Code	Package <sup>(1)</sup>	Operation Range
8	2.7 - 5.5V	ATmega128L-8AC	64A	Commercial (0°C to 70°C)
		ATmega128L-8MC	64M1	
		ATmega128L-8AI	64A	Industrial (-40°C to 85°C)
		ATmega128L-8AU <sup>(2)</sup>	64A	
16	4.5 - 5.5V	ATmega128L-8MI	64M1	
		ATmega128L-8MU <sup>(2)</sup>	64M1	
		ATmega128-16AC	64A	Commercial (0°C to 70°C)
		ATmega128-16MC	64M1	
		ATmega128-16AI	64A	Industrial (-40°C to 85°C)
		ATmega128-16AU <sup>(2)</sup>	64A	
		ATmega128-16MI	64M1	
		ATmega128-16MU <sup>(2)</sup>	64M1	

- Notes:
1. The device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
  2. Pb-free packaging alternative, complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.

Package Type	
<b>64A</b>	64-lead, 14 x 14 x 1.0 mm, Thin Profile Plastic Quad Flat Package (TQFP)
<b>64M1</b>	64-pad, 9 x 9 x 1.0 mm, Micro Lead Frame Package (MLF)

# Packaging Information

64A



**COMMON DIMENSIONS**  
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	—	—	1.20	
A1	0.05	—	0.15	
A2	0.95	1.00	1.05	
D	15.75	16.00	16.25	
D1	13.90	14.00	14.10	Note 2
E	15.75	16.00	16.25	
E1	13.90	14.00	14.10	Note 2
B	0.30	—	0.45	
C	0.09	—	0.20	
L	0.45	—	0.75	
e	0.80 TYP			

- Notes:
1. This package conforms to JEDEC reference MS-026, Variation AEB.
  2. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25 mm per side. Dimensions D1 and E1 are maximum plastic body size dimensions including mold mismatch.
  3. Lead coplanarity is 0.10 mm maximum.

10/5/2001



2325 Orchard Parkway  
San Jose, CA 95131

## TITLE

**64A**, 64-lead, 14 x 14 mm Body Size, 1.0 mm Body Thickness,  
0.8 mm Lead Pitch, Thin Profile Plastic Quad Flat Package (TQFP)

## DRAWING NO.

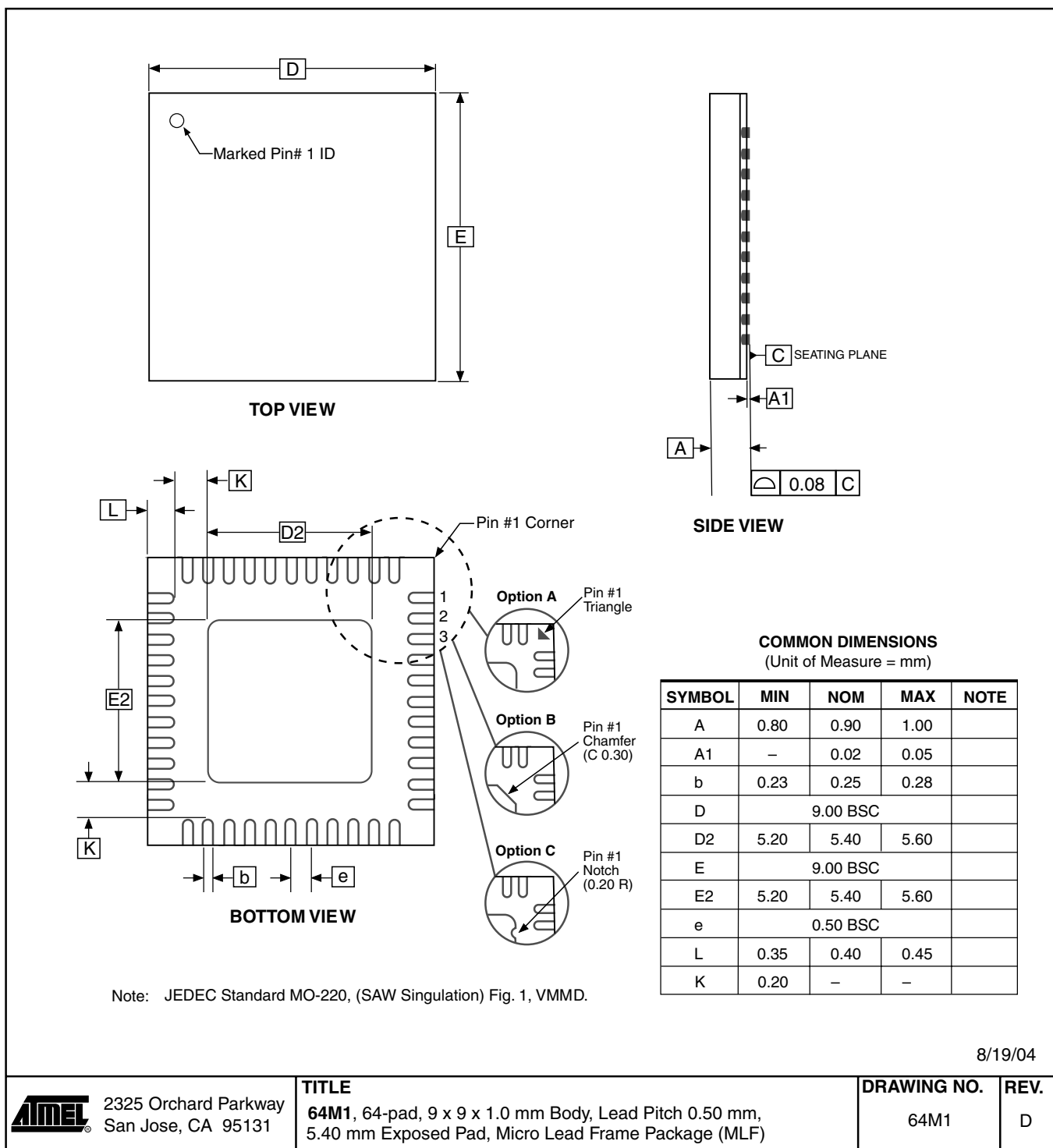
64A

## REV.

B



# 64M1



## Errata

The revision letter in this section refers to the revision of the ATmega128 device.

### ATmega128 Rev. I

- **Stabilizing time needed when changing XDIV Register**
- **Stabilizing time needed when changing OSCCAL Register**

#### 1. Stabilizing time needed when changing XDIV Register

After increasing the source clock frequency more than 2% with settings in the XDIV register, the device may execute some of the subsequent instructions incorrectly.

##### Problem Fix / Workaround

The NOP instruction will always be executed correctly also right after a frequency change. Thus, the next 8 instructions after the change should be NOP instructions. To ensure this, follow this procedure:

1. Clear the I bit in the SREG Register.
2. Set the new pre-scaling factor in XDIV register.
3. Execute 8 NOP instructions
4. Set the I bit in SREG

This will ensure that all subsequent instructions will execute correctly.

##### Assembly Code Example:

```
CLI                ; clear global interrupt enable
OUT  XDIV, temp    ; set new prescale value
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
SEI                ; clear global interrupt enable
```

#### 2. Stabilizing time needed when changing OSCCAL Register

After increasing the source clock frequency more than 2% with settings in the OSCCAL register, the device may execute some of the subsequent instructions incorrectly.

##### Problem Fix / Workaround

The behavior follows errata number 1., and the same Fix / Workaround is applicable on this errata.

A proposal for solving problems regarding the JTAG instruction IDCODE is presented below.

##### IDCODE masks data from TDI input

The public but optional JTAG instruction IDCODE is not implemented correctly according to IEEE1149.1; a logic one is scanned into the shift register instead of the TDI input while shifting the Device ID Register. Hence, captured data from the preceding devices in the boundary scan chain are lost and replaced by all-ones, and data to succeeding devices are replaced by all-ones during Update-DR.

If ATmega128 is the only device in the scan chain, the problem is not visible.



### Problem Fix / Workaround

Select the Device ID Register of the ATmega128 (Either by issuing the IDCODE instruction or by entering the Test-Logic-Reset state of the TAP controller) to read out the contents of its Device ID Register and possibly data from succeeding devices of the scan chain. Note that data to succeeding devices cannot be entered during this scan, but data to preceding devices can. Issue the BYPASS instruction to the ATmega128 to select its Bypass Register while reading the Device ID Registers of preceding devices of the boundary scan chain. Never read data from succeeding devices in the boundary scan chain or upload data to the succeeding devices while the Device ID Register is selected for the ATmega128. Note that the IDCODE instruction is the default instruction selected by the Test-Logic-Reset state of the TAP-controller.

### Alternative Problem Fix / Workaround

If the Device IDs of all devices in the boundary scan chain must be captured simultaneously (for instance if blind interrogation is used), the boundary scan chain can be connected in such way that the ATmega128 is the first device in the chain. Update-DR will still not work for the succeeding devices in the boundary scan chain as long as IDCODE is present in the JTAG Instruction Register, but the Device ID registered cannot be uploaded in any case.

## ATmega128 Rev. H

- Stabilizing time needed when changing XDIV Register
- Stabilizing time needed when changing OSCCAL Register

### 1. Stabilizing time needed when changing XDIV Register

After increasing the source clock frequency more than 2% with settings in the XDIV register, the device may execute some of the subsequent instructions incorrectly.

#### Problem Fix / Workaround

The NOP instruction will always be executed correctly also right after a frequency change. Thus, the next 8 instructions after the change should be NOP instructions. To ensure this, follow this procedure:

1. Clear the I bit in the SREG Register.
2. Set the new pre-scaling factor in XDIV register.
3. Execute 8 NOP instructions
4. Set the I bit in SREG

This will ensure that all subsequent instructions will execute correctly.

Assembly Code Example:

```
CLI                ; clear global interrupt enable
OUT  XDIV, temp    ; set new prescale value
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
SEI                ; clear global interrupt enable
```

## 2. Stabilizing time needed when changing OSCCAL Register

After increasing the source clock frequency more than 2% with settings in the OSCCAL register, the device may execute some of the subsequent instructions incorrectly.

### Problem Fix / Workaround

The behavior follows errata number 1., and the same Fix / Workaround is applicable on this errata.

A proposal for solving problems regarding the JTAG instruction IDCODE is presented below.

### IDCODE masks data from TDI input

The public but optional JTAG instruction IDCODE is not implemented correctly according to IEEE1149.1; a logic one is scanned into the shift register instead of the TDI input while shifting the Device ID Register. Hence, captured data from the preceding devices in the boundary scan chain are lost and replaced by all-ones, and data to succeeding devices are replaced by all-ones during Update-DR.

If ATmega128 is the only device in the scan chain, the problem is not visible.

### Problem Fix / Workaround

Select the Device ID Register of the ATmega128 (Either by issuing the IDCODE instruction or by entering the Test-Logic-Reset state of the TAP controller) to read out the contents of its Device ID Register and possibly data from succeeding devices of the scan chain. Note that data to succeeding devices cannot be entered during this scan, but data to preceding devices can. Issue the BYPASS instruction to the ATmega128 to select its Bypass Register while reading the Device ID Registers of preceding devices of the boundary scan chain. Never read data from succeeding devices in the boundary scan chain or upload data to the succeeding devices while the Device ID Register is selected for the ATmega128. Note that the IDCODE instruction is the default instruction selected by the Test-Logic-Reset state of the TAP-controller.

### Alternative Problem Fix / Workaround

If the Device IDs of all devices in the boundary scan chain must be captured simultaneously (for instance if blind interrogation is used), the boundary scan chain can be connected in such way that the ATmega128 is the first device in the chain. Update-DR will still not work for the succeeding devices in the boundary scan chain as long as IDCODE is present in the JTAG Instruction Register, but the Device ID registered cannot be uploaded in any case.

## ATmega128 Rev. G

- Stabilizing time needed when changing XDIV Register
- Stabilizing time needed when changing OSCCAL Register

## 1. Stabilizing time needed when changing XDIV Register

After increasing the source clock frequency more than 2% with settings in the XDIV register, the device may execute some of the subsequent instructions incorrectly.

### Problem Fix / Workaround

The NOP instruction will always be executed correctly also right after a frequency change. Thus, the next 8 instructions after the change should be NOP instructions. To ensure this, follow this procedure:

1. Clear the I bit in the SREG Register.
2. Set the new pre-scaling factor in XDIV register.

3. Execute 8 NOP instructions

4. Set the I bit in SREG

This will ensure that all subsequent instructions will execute correctly.

Assembly Code Example:

```
CLI                ; clear global interrupt enable
OUT  XDIV, temp    ; set new prescale value
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
SEI                ; clear global interrupt enable
```

## 2. Stabilizing time needed when changing OSCCAL Register

After increasing the source clock frequency more than 2% with settings in the OSC-CAL register, the device may execute some of the subsequent instructions incorrectly.

### Problem Fix / Workaround

The behavior follows errata number 1., and the same Fix / Workaround is applicable on this errata.

A proposal for solving problems regarding the JTAG instruction IDCODE is presented below.

### IDCODE masks data from TDI input

The public but optional JTAG instruction IDCODE is not implemented correctly according to IEEE1149.1; a logic one is scanned into the shift register instead of the TDI input while shifting the Device ID Register. Hence, captured data from the preceding devices in the boundary scan chain are lost and replaced by all-ones, and data to succeeding devices are replaced by all-ones during Update-DR.

If ATmega128 is the only device in the scan chain, the problem is not visible.

### Problem Fix / Workaround

Select the Device ID Register of the ATmega128 (Either by issuing the IDCODE instruction or by entering the Test-Logic-Reset state of the TAP controller) to read out the contents of its Device ID Register and possibly data from succeeding devices of the scan chain. Note that data to succeeding devices cannot be entered during this scan, but data to preceding devices can. Issue the BYPASS instruction to the ATmega128 to select its Bypass Register while reading the Device ID Registers of preceding devices of the boundary scan chain. Never read data from succeeding devices in the boundary scan chain or upload data to the succeeding devices while the Device ID Register is selected for the ATmega128. Note that the IDCODE instruction is the default instruction selected by the Test-Logic-Reset state of the TAP-controller.

### Alternative Problem Fix / Workaround

If the Device IDs of all devices in the boundary scan chain must be captured simultaneously (for instance if blind interrogation is used), the boundary scan chain can

be connected in such way that the ATmega128 is the first device in the chain. Update-DR will still not work for the succeeding devices in the boundary scan chain as long as IDCODE is present in the JTAG Instruction Register, but the Device ID registered cannot be uploaded in any case.

## ATmega128 Rev. F

- **Stabilizing time needed when changing XDIV Register**
- **Stabilizing time needed when changing OSCCAL Register**

### 1. Stabilizing time needed when changing XDIV Register

After increasing the source clock frequency more than 2% with settings in the XDIV register, the device may execute some of the subsequent instructions incorrectly.

#### Problem Fix / Workaround

The NOP instruction will always be executed correctly also right after a frequency change. Thus, the next 8 instructions after the change should be NOP instructions. To ensure this, follow this procedure:

1. Clear the I bit in the SREG Register.
2. Set the new pre-scaling factor in XDIV register.
3. Execute 8 NOP instructions
4. Set the I bit in SREG

This will ensure that all subsequent instructions will execute correctly.

#### Assembly Code Example:

```
CLI                ; clear global interrupt enable
OUT  XDIV, temp    ; set new prescale value
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
NOP                ; no operation
SEI                ; clear global interrupt enable
```

### 2. Stabilizing time needed when changing OSCCAL Register

After increasing the source clock frequency more than 2% with settings in the OSCCAL register, the device may execute some of the subsequent instructions incorrectly.

#### Problem Fix / Workaround

The behavior follows errata number 1., and the same Fix / Workaround is applicable on this errata.

A proposal for solving problems regarding the JTAG instruction IDCODE is presented below.

#### IDCODE masks data from TDI input

The public but optional JTAG instruction IDCODE is not implemented correctly according to IEEE1149.1; a logic one is scanned into the shift register instead of the TDI input while shifting the Device ID Register. Hence, captured data from the pre-

ceding devices in the boundary scan chain are lost and replaced by all-ones, and data to succeeding devices are replaced by all-ones during Update-DR.

If ATmega128 is the only device in the scan chain, the problem is not visible.

#### **Problem Fix / Workaround**

Select the Device ID Register of the ATmega128 (Either by issuing the IDCODE instruction or by entering the Test-Logic-Reset state of the TAP controller) to read out the contents of its Device ID Register and possibly data from succeeding devices of the scan chain. Note that data to succeeding devices cannot be entered during this scan, but data to preceding devices can. Issue the BYPASS instruction to the ATmega128 to select its Bypass Register while reading the Device ID Registers of preceding devices of the boundary scan chain. Never read data from succeeding devices in the boundary scan chain or upload data to the succeeding devices while the Device ID Register is selected for the ATmega128. Note that the IDCODE instruction is the default instruction selected by the Test-Logic-Reset state of the TAP-controller.

#### **Alternative Problem Fix / Workaround**

If the Device IDs of all devices in the boundary scan chain must be captured simultaneously (for instance if blind interrogation is used), the boundary scan chain can be connected in such way that the ATmega128 is the first device in the chain. Update-DR will still not work for the succeeding devices in the boundary scan chain as long as IDCODE is present in the JTAG Instruction Register, but the Device ID registered cannot be uploaded in any case.

## Datasheet Revision History

Please note that the referring page numbers in this section are referred to this document. The referring revision in this section are referring to the document revision.

### Changes from Rev. 2467L-05/04 to Rev. 2467M-11/04

1. Removed “analog ground”, replaced by “ground”.
2. Updated Table 11 on page 38, Table 114 on page 287, Table 128 on page 306, and Table 132 on page 323. Updated Figure 114 on page 239.
3. Added note to “Port C (PC7..PC0)” on page 6.
4. Updated “Ordering Information” on page 14.

### Changes from Rev. 2467K-03/04 to Rev. 2467L-05/04

1. Removed “Preliminary” and “TBD” from the datasheet, replaced occurrences of ICx with ICPx.
2. Updated Table 8 on page 36, Table 19 on page 48, Table 22 on page 54, Table 96 on page 243, Table 126 on page 302, Table 128 on page 306, Table 132 on page 323, and Table 134 on page 325.
3. Updated “External Memory Interface” on page 24.
4. Updated “Device Identification Register” on page 255.
5. Updated “Electrical Characteristics” on page 321.
6. Updated “ADC Characteristics” on page 327.
7. Updated “ATmega128 Typical Characteristics” on page 335.
8. Updated “Ordering Information” on page 14.

### Changes from Rev. 2467J-12/03 to Rev. 2467K-03/04

1. Updated “Errata” on page 17.

### Changes from Rev. 2467I-09/03 to Rev. 2467J-12/03

1. Updated “Calibrated Internal RC Oscillator” on page 39.

### Changes from Rev. 2467H-02/03 to Rev. 2467I-09/03

1. Updated note in “XTAL Divide Control Register – XDIV” on page 41.
2. Updated “JTAG Interface and On-chip Debug System” on page 46.
3. Updated values for  $V_{BOT}$  (BODLEVEL = 1) in Table 19 on page 48.
4. Updated “Test Access Port – TAP” on page 248 regarding JTAGEN.
5. Updated description for the JTD bit on page 257.
6. Added a note regarding JTAGEN fuse to Table 118 on page 290.

**Changes from Rev.  
2467G-09/02 to Rev.  
2467H-02/03**

7. Updated  $R_{PU}$  values in “DC Characteristics” on page 321.
8. Added a proposal for solving problems regarding the JTAG instruction IDCODE in “Errata” on page 17.
1. Corrected the names of the two Prescaler bits in the SFIOR Register.
2. Added Chip Erase as a first step under “Programming the Flash” on page 318 and “Programming the EEPROM” on page 319.
3. Removed reference to the “Multipurpose Oscillator” application note and the “32 kHz Crystal Oscillator” application note, which do not exist.
4. Corrected OCn waveforms in Figure 52 on page 123.
5. Various minor Timer1 corrections.
6. Added information about PWM symmetry for Timer0 and Timer2.
7. Various minor TWI corrections.
8. Added reference to Table 124 on page 293 from both SPI Serial Programming and Self Programming to inform about the Flash Page size.
9. Added note under “Filling the Temporary Buffer (Page Loading)” on page 282 about writing to the EEPROM during an SPM Page load.
10. Removed ADHSM completely.
11. Added section “EEPROM Write During Power-down Sleep Mode” on page 23.
12. Updated drawings in “Packaging Information” on page 15.

**Changes from Rev.  
2467F-09/02 to Rev.  
2467G-09/02**

1. Changed the Endurance on the Flash to 10,000 Write/Erase Cycles.

**Changes from Rev.  
2467E-04/02 to Rev.  
2467F-09/02**

1. Added 64-pad MLF Package and updated “Ordering Information” on page 14.
2. Added the section “Using all Locations of External Memory Smaller than 64 KB” on page 31.
3. Added the section “Default Clock Source” on page 35.
4. Renamed SPMCR to SPMCSR in entire document.
5. When using external clock there are some limitations regards to change of frequency. This is descried in “External Clock” on page 40 and Table 131, “External Clock Drive,” on page 323.
6. Added a sub section regarding OCD-system and power consumption in the section “Minimizing Power Consumption” on page 45.

## 7. Corrected typo (WGM-bit setting) for:

- “Fast PWM Mode” on page 96 (Timer/Counter0).
- “Phase Correct PWM Mode” on page 98 (Timer/Counter0).
- “Fast PWM Mode” on page 150 (Timer/Counter2).
- “Phase Correct PWM Mode” on page 152 (Timer/Counter2).

## 8. Corrected Table 81 on page 192 (USART).

## 9. Corrected Table 102 on page 261 (Boundary-Scan)

## 10. Updated V<sub>il</sub> parameter in “DC Characteristics” on page 321.

### Changes from Rev. 2467D-03/02 to Rev. 2467E-04/02

## 1. Updated the Characterization Data in Section “ATmega128 Typical Characteristics” on page 335.

## 2. Updated the following tables:

Table 19 on page 48, Table 20 on page 52, Table 68 on page 157, Table 102 on page 261, and Table 136 on page 328.

## 3. Updated Description of OSCCAL Calibration Byte.

In the data sheet, it was not explained how to take advantage of the calibration bytes for 2, 4, and 8 MHz Oscillator selections. This is now added in the following sections:

Improved description of “Oscillator Calibration Register – OSCCAL” on page 39 and “Calibration Byte” on page 291.

### Changes from Rev. 2467C-02/02 to Rev. 2467D-03/02

## 1. Added more information about “ATmega103 Compatibility Mode” on page 5.

## 2. Updated Table 2, “EEPROM Programming Time,” on page 21.

## 3. Updated typical Start-up Time in Table 7 on page 35, Table 9 and Table 10 on page 37, Table 12 on page 38, Table 14 on page 39, and Table 16 on page 40.

## 4. Updated Table 22 on page 54 with typical WDT Time-out.

## 5. Corrected description of ADSC bit in “ADC Control and Status Register A – ADCSRA” on page 245.

## 6. Improved description on how to do a polarity check of the ADC differential results in “ADC Conversion Result” on page 242.

## 7. Corrected JTAG version numbers in “JTAG Version Numbers” on page 256.

## 8. Improved description of addressing during SPM (usage of RAMPZ) on “Addressing the Flash During Self-Programming” on page 280, “Performing Page Erase by SPM” on page 282, and “Performing a Page Write” on page 282.

## 9. Added note regarding OCDEN Fuse below Table 118 on page 290.

## 10. Updated Programming Figures:



Figure 135 on page 292 and Figure 144 on page 304 are updated to also reflect that AVCC must be connected during Programming mode. Figure 139 on page 299 added to illustrate how to program the fuses.

11. **Added a note regarding usage of the PROG\_PAGELOAD and PROG\_PAGEREAD instructions on page 310.**
12. **Added Calibrated RC Oscillator characterization curves in section “ATmega128 Typical Characteristics” on page 335.**
13. **Updated “Two-wire Serial Interface” section.**  
More details regarding use of the TWI Power-down operation and using the TWI as master with low TWBRR values are added into the data sheet. Added the note at the end of the “Bit Rate Generator Unit” on page 204. Added the description at the end of “Address Match Unit” on page 205.
14. **Added a note regarding usage of Timer/Counter0 combined with the clock. See “XTAL Divide Control Register – XDIV” on page 41.**

#### **Changes from Rev. 2467B-09/01 to Rev. 2467C-02/02**

1. **Corrected Description of Alternate Functions of Port G**  
Corrected description of TOSC1 and TOSC2 in “Alternate Functions of Port G” on page 82.
2. **Added JTAG Version Numbers for rev. F and rev. G**  
Updated Table 100 on page 256.
3. **Added Some Preliminary Test Limits and Characterization Data**  
Removed some of the TBD's in the following tables and pages:  
Table 19 on page 48, Table 20 on page 52, “DC Characteristics” on page 321, Table 131 on page 323, Table 134 on page 325, and Table 136 on page 328.
4. **Corrected “Ordering Information” on page 14.**
5. **Added some Characterization Data in Section “ATmega128 Typical Characteristics” on page 335.**
6. **Removed Alternative Algorithm for Leaving JTAG Programming Mode.**  
See “Leaving Programming Mode” on page 318.
7. **Added Description on How to Access the Extended Fuse Byte Through JTAG Programming Mode.**  
See “Programming the Fuses” on page 320 and “Reading the Fuses and Lock Bits” on page 320.



## Atmel Corporation

2325 Orchard Parkway  
San Jose, CA 95131, USA  
Tel: 1(408) 441-0311  
Fax: 1(408) 487-2600

## Regional Headquarters

### Europe

Atmel Sarl  
Route des Arsenaux 41  
Case Postale 80  
CH-1705 Fribourg  
Switzerland  
Tel: (41) 26-426-5555  
Fax: (41) 26-426-5500

### Asia

Room 1219  
Chinachem Golden Plaza  
77 Mody Road Tsimshatsui  
East Kowloon  
Hong Kong  
Tel: (852) 2721-9778  
Fax: (852) 2722-1369

### Japan

9F, Tonetsu Shinkawa Bldg.  
1-24-8 Shinkawa  
Chuo-ku, Tokyo 104-0033  
Japan  
Tel: (81) 3-3523-3551  
Fax: (81) 3-3523-7581

## Atmel Operations

### Memory

2325 Orchard Parkway  
San Jose, CA 95131, USA  
Tel: 1(408) 441-0311  
Fax: 1(408) 436-4314

### Microcontrollers

2325 Orchard Parkway  
San Jose, CA 95131, USA  
Tel: 1(408) 441-0311  
Fax: 1(408) 436-4314

### La Chantierie

BP 70602  
44306 Nantes Cedex 3, France  
Tel: (33) 2-40-18-18-18  
Fax: (33) 2-40-18-19-60

### ASIC/ASSP/Smart Cards

Zone Industrielle  
13106 Rousset Cedex, France  
Tel: (33) 4-42-53-60-00  
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.  
Colorado Springs, CO 80906, USA  
Tel: 1(719) 576-3300  
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park  
Maxwell Building  
East Kilbride G75 0QR, Scotland  
Tel: (44) 1355-803-000  
Fax: (44) 1355-242-743

### RF/Automotive

Theresienstrasse 2  
Postfach 3535  
74025 Heilbronn, Germany  
Tel: (49) 71-31-67-0  
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.  
Colorado Springs, CO 80906, USA  
Tel: 1(719) 576-3300  
Fax: 1(719) 540-1759

### Biometrics/Imaging/Hi-Rel MPU/ High Speed Converters/RF Datacom

Avenue de Rochepleine  
BP 123  
38521 Saint-Egreve Cedex, France  
Tel: (33) 4-76-58-30-00  
Fax: (33) 4-76-58-34-80

---

## Literature Requests

[www.atmel.com/literature](http://www.atmel.com/literature)

**Disclaimer:** The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. **EXCEPT AS SET FORTH IN ATMEL'S TERMS AND CONDITIONS OF SALE LOCATED ON ATMEL'S WEB SITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Atmel's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

© Atmel Corporation 2004. All rights reserved. Atmel®, logo and combinations thereof, AVR®, and AVR Studio® are registered trademarks, and Everywhere You Are<sup>SM</sup> are the trademarks of Atmel Corporation or its subsidiaries. Microsoft®, Windows®, Windows NT®, and Windows XP® are the registered trademarks of Microsoft Corporation. Other terms and product names may be trademarks of others.



Printed on recycled paper.

2467MS-AVR-11/04

---

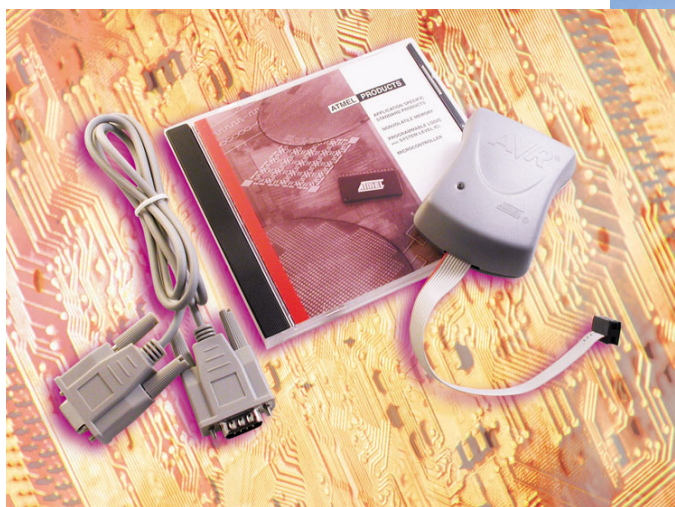
## C.3 AVR ISP, In-System Programmer

# AVR<sup>®</sup> ISP In-System Programmer

## COMPACT AND EASY-TO-USE TOOL FOR AVR IN-SYSTEM PROGRAMMING

The Atmel<sup>®</sup> AVR ISP is an In-System Programmer for Atmel's AVR<sup>®</sup> Flash microcontrollers. The AVR ISP gives the designer a compact and reliable programming tool to program all In-System Programmable AVR microcontrollers through a 6- or 10-pin ISP connector. The AVR ISP uses AVR Studio<sup>®</sup>, Atmel's Integrated Development Environment (IDE) for code writing and debugging.

The programming software can be controlled from both a Windows<sup>®</sup> environment and a DOS command-line interface.



- AVR Studio Operated
- Serial In-System Programming
- RS-232 Interface to PC
- Upgrades are done from AVR Studio
- Target Voltage 2.7 – 6.0V
- Powered from Target. No Need for Additional Power Supply

**Corporate Headquarters**

2325 Orchard Parkway  
San Jose, CA 95131  
USA  
TEL: (1)(408) 441-0311  
FAX: (1)(408) 487-2600

**Europe**

Atmel Sarl  
Route des Arsenaux 41  
Case Postale 80  
CH-1705 Fribourg  
Switzerland  
TEL: (41) 26-426-5555  
FAX: (41) 26-426-5500

**Asia**

Room 1219  
Chinachem Golden Plaza  
77 Mody Road Tsimshatsui  
East Kowloon  
Hong Kong  
TEL: (852) 2721-9778  
FAX: (852) 2722-1369

**Japan**

9F, Tonetsu Shinkawa Bldg.  
1-24-8 Shinkawa  
Chuo-ku, Tokyo 104-0033  
Japan  
TEL: (81) 3-3523-3551  
FAX: (81) 3-3523-7581

**e-mail**

literature@atmel.com

**Web Site**

<http://www.atmel.com>



© Atmel Corporation 2004. All rights reserved. Atmel®, AVR® and AVR Studio® and combinations thereof are the registered trademarks of Atmel Corporation or its subsidiaries.

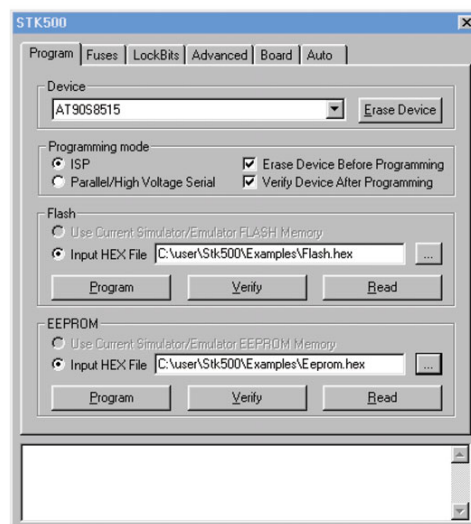
Windows® is a registered trademark of Microsoft Corporation.  
Other terms and product names may be the trademarks of others.

2492D-AVR-02/04/15M

The AVR ISP is a compact and easy-to-use In-System Programming tool for developing applications with Atmel's AVR microcontrollers. Due to the small size, it is also an excellent tool for field upgrades of existing applications using AVR microcontrollers. The AVR ISP is powered by the target application and an additional power supply is thus not required for AVR ISP Programmer.

The AVR ISP Programming interface is integrated in AVR Studio. The Flash, EEPROM and all Fuse and Lock Bit options ISP-programmable can be programmed individually or with the sequential automatic programming option. The AVR clock frequency and supply voltage can also be controlled from AVR Studio.

A DOS Programming software is included for efficient batch programming in a production environment.

**Supported Devices**

ATtiny12	AT90S2333	ATmega8535	ATmega64
ATtiny13	AT90S4414	ATmega161	ATmega103
ATtiny2313	AT90S2343	ATmega162	ATmega128
ATtiny15	AT90S4433	ATmega163	AT89S51
ATtiny22	AT90S4434	ATmega16	AT89S52
ATtiny26	AT90S8515	ATmega169	AT86RF401
AT90S1200	AT90S8535	ATmega323	
AT90S2313	ATmega8	ATmega32	
AT90S2323	ATmega8515	ATmega48	

*Note: Low power versions are also supported.*

**Ordering Information**

The AVR ISP is available from Atmel franchised distributors.

The ordering code is **ATAVRISP**

The latest version of AVR Studio is available free of charge from Atmel web site: [www.atmel.com](http://www.atmel.com)

## C.4 AVR033: Getting Started with the CodeVisionAVR C Compiler

# AVR033: Getting Started with the CodeVisionAVR C Compiler

## Features

- Installing and Configuring CodeVisionAVR to Work with the Atmel STK500 Starter Kit and AVR Studio® Debugger
- Creating a New Project Using the CodeWizardAVR Automatic Program Generator
- Editing and Compiling the C Code
- Loading the Executable Code into the Target Microcontroller on the STK500 Starter Kit

## Introduction

The purpose of this application note is to guide the user through the preparation of an example C program using the CodeVisionAVR C compiler. The example, which is the subject of this application note, is a simple program for the Atmel AT90S8515 microcontroller on the STK500 starter kit.

## Preparation

Install the CodeVisionAVR C Compiler in the default directory: C:\cvavr.

Install the Atmel AVR Studio debugger in the default directory:

C:\Program Files\Atmel\AVR Studio.

The demonstration program to be developed in the next few pages requires an Atmel AT90S8515 microcontroller and the STK500 starter kit.

Set up the starter kit according to the instructions in the STK500 User Guide.

Make sure the power is off and insert the AT90S8515 chip into the appropriate socket marked SCKT3000D3.

Set the VTARGET, RESET and XTAL1 jumpers. Also set the OSCSEL jumper between pins 1 and 2.

Connect one 10-pin ribbon cable between the PORTB and LEDs headers.

This will allow displaying the state of AT90S8515's PORTB outputs.

Connect one 6-pin ribbon cable between the ISP6PIN and SPROG3 headers.

This will allow the CodeVisionAVR IDE to automatically program the AVR chip after a successful compilation.

In order to use this feature, one supplementary setting must be done:

Open the CodeVisionAVR IDE and select the "Settings\Programmer" menu option.

The dialog window as shown in Figure 1 will open.



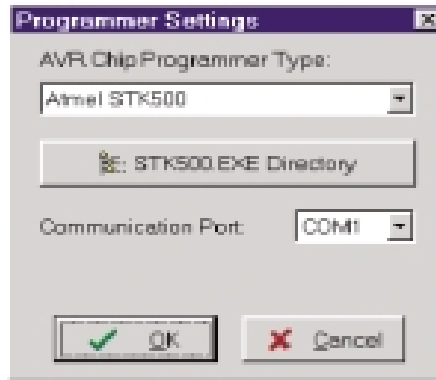
8-bit **AVR**®  
Microcontroller

Application  
Note

Rev. 2500A–10/01



**Figure 1. Programmer Settings**

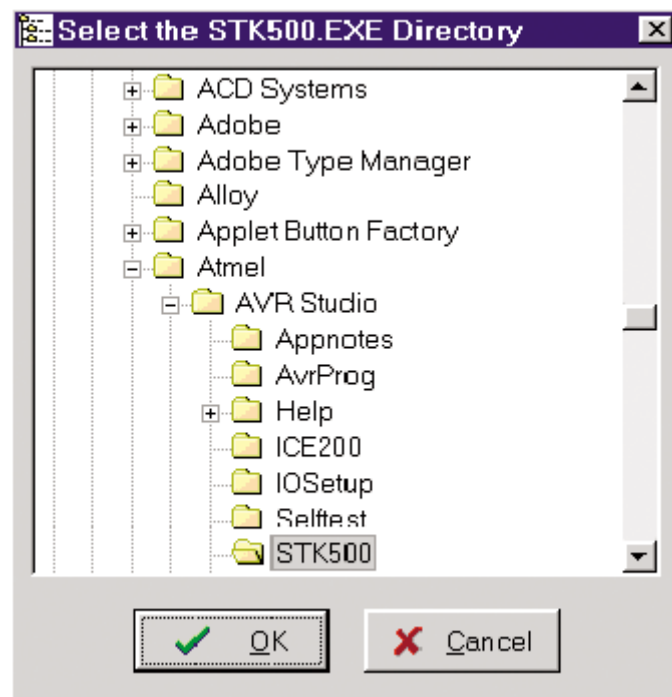


Make sure to select as Chip Programmer Type the Atmel STK500 AVR and the corresponding Communication Port that is used with the STK500 starter kit.

Then press the “STK500.EXE Directory” button in order to specify the location of the stk500.exe command line utility supplied with AVR Studio.

The dialog window as shown in Figure 2 will open.

**Figure 2. Directory Selection**



Select the “c:\Program Files\Atmel\AVR Studio\STK500” directory and press the “OK” button.

Then press once again the “OK” button in order to save the Programmer Settings.

In order to be able to invoke the AVR Studio debugger from within the CodeVisionAVR IDE one final setting must be done.




Select the “Settings|Debugger” menu option. The dialog window as shown in Figure 3 will open.

**Figure 3.** Debugger Settings



Enter “C:\Program Files\Atmel\AVR Studio\AvrStudio.exe” and press the “OK” button.

## Creating a New Project

In order to create a new project, select the “File|New” menu option or press the  tool-bar button.

The window shown in Figure 4 will be displayed.

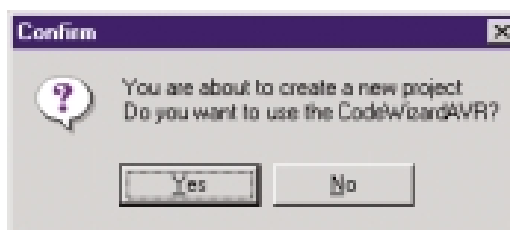
**Figure 4.** New Project Window



Select “Project” and press “OK”.

Then the window shown in Figure 5 will be displayed.

**Figure 5.** Confirmation

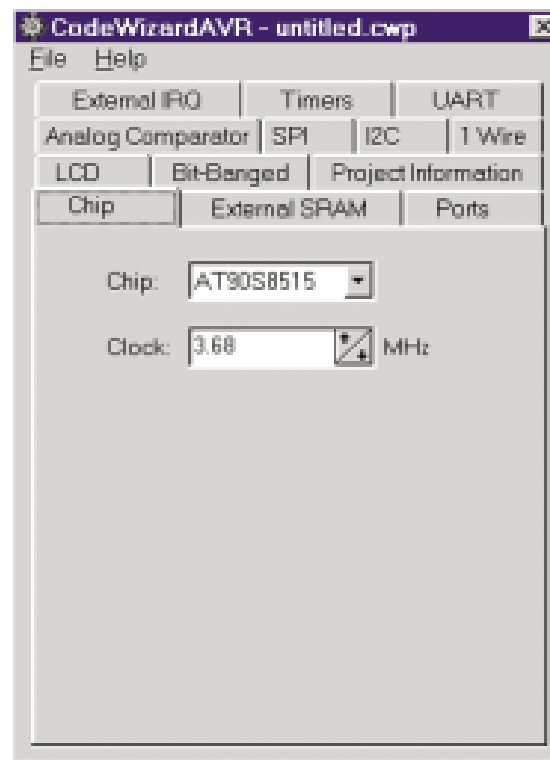


Press “Yes” to use the CodeWizardAVR Automatic Program Generator.

## Using the CodeWizardAVR Automatic Program Generator

The CodeWizardAVR simplifies the task of writing start-up code for different AVR microcontrollers.

**Figure 6.** Selections

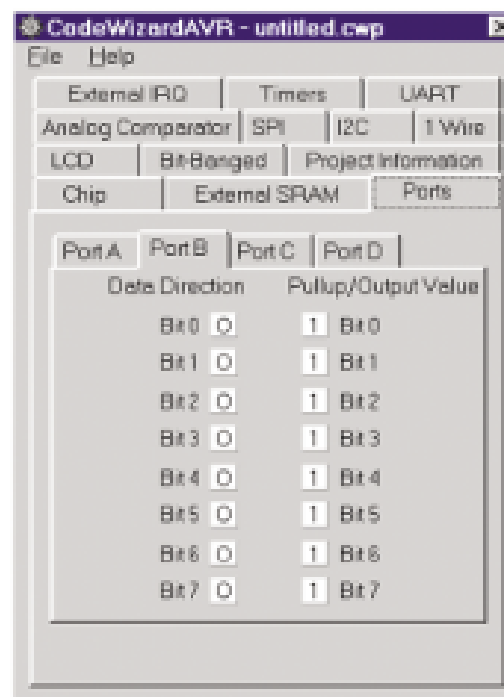


The window shown in Figure 6 opens and, for this example project, we shall select the AT90S8515 microcontroller and set the clock rate to 3.68 MHz since that is the clock on the STK500 starter kit.

## Configuring the Input/Output Ports

Select the “Ports” tab to determine how the I/O ports are to be initialized for the target system.

**Figure 7.** I/O Ports Initialization



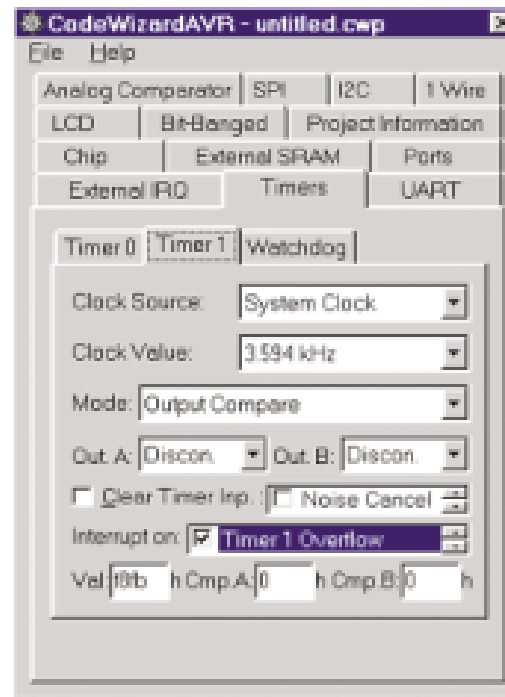
The default setting is to have the ports for all the target systems to be inputs (Data Direction bits to be all 1s) in their Tri-state mode.

For this exercise, we want to set Port B (by selecting the Port B tab) to be all outputs and we do this by setting all the Data Direction bits to O (by clicking on them). We also set the Output Values to be all 1s since this corresponds to the LEDs on the STK500 being off.

## Configuring Timer1

For this project, we want to configure Timer1 to generate overflow interrupts. We select the Timers tab and then select the Timer1 tab resulting in Figure 8.

**Figure 8.** Timer Tab



Set the options as shown in Figure 8. We have selected a clock rate of 3.594 kHz (the system clock of 3.68 MHz divided by 1024).

The timer is set to operate in the default “Output Compare” mode and to generate interrupts on overflow.

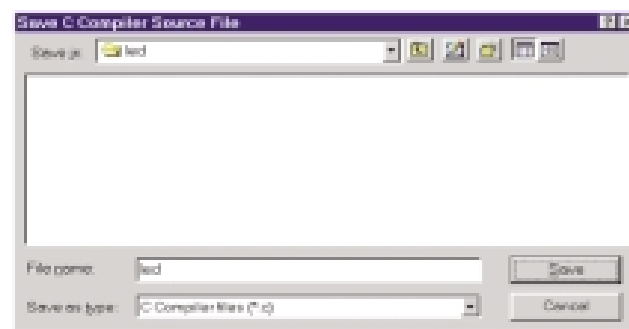
To obtain the frequency of LED movement of 2 per second we need to reinitialize the Timer1 value to  $0x10000 - (3594/2) = 0xF8FB$  on every overflow.


## Completing the Project

By selecting the File|Generate, Save and Exit menu option the CodeWizard will generate a skeleton C program with, in this case, the Port B and Timer1 overflow interrupt set up correctly.

The dialog window shown in Figure 9 will appear.

**Figure 9.** Save Source File Dialog Box

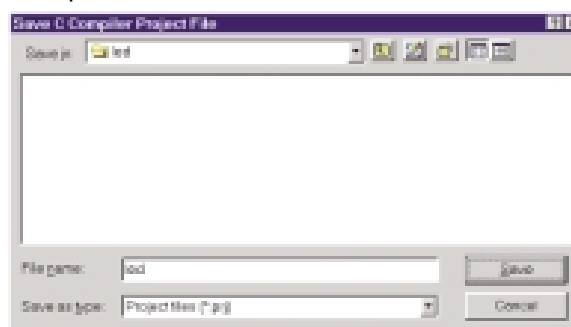


By pressing the  button, a new directory C:\cvavr\led must be created. It will hold all the files of our sample project.

Then we must specify the File name of the C source file: led.c and press the “Save” button.

A new dialog window will open. This is shown in Figure 10.

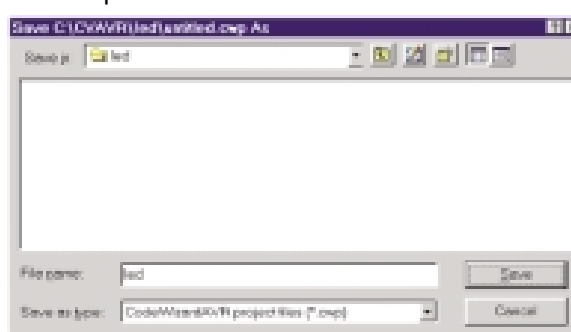
**Figure 10.** File Name Specification



Here, we must specify the File name led.prj, as the project name and put it in the same folder: C:\cvavr\led.

Finally, we will be prompted to save the CodeWizard project file, as shown in Figure 11.

**Figure 11.** File Save Prompt



We must specify the File name as led.cwp and press the “Save” button.

Saving all the CodeWizardAVR peripherals configuration in the led.cwp project file, will allow us to reuse some of our initialization code in future projects.

The led.c source file is now automatically opened and available.


One can then start editing the code produced by the CodeWizardAVR.

The source listing is given on Appendix A of this application note.

In this example, only the interrupt handler code needs to be amended to manage the LED display.

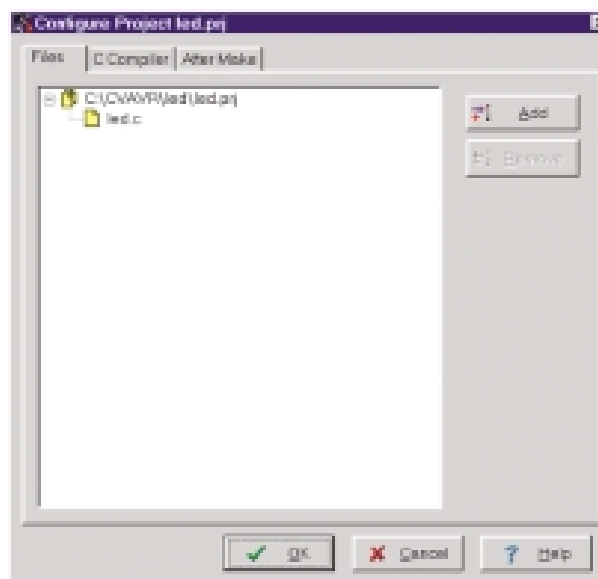
The small bit of code that was added is shown with bold font, the remainder was supplied by the CodeWizardAVR.

## Viewing or Modifying the Project Configuration

At any time, a project configuration may be changed using the Project|Configure menu option or by pressing the  toolbar button.

The dialog window shown in Figure 12 will open.

**Figure 12.** Configure Window Dialog Box

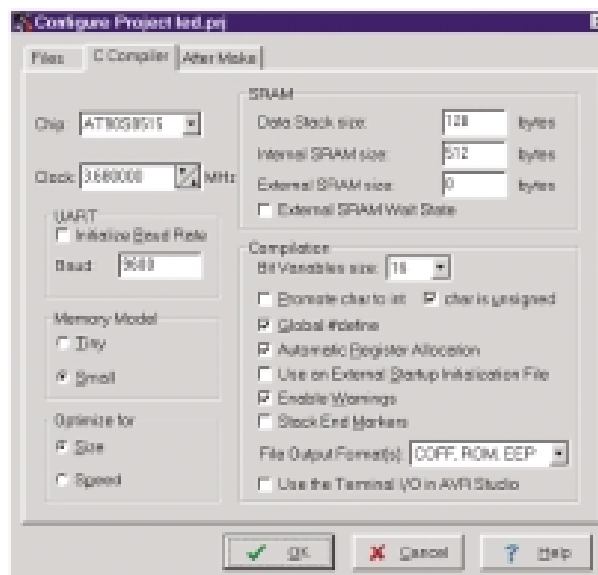


To add, respectively remove, files from the project select the “Files” tab and use the “Add”, respectively “Remove” buttons.

To change the target microcontroller, the clock rate or the various compiler options select the “C Compiler” tab.

The dialog box shown in Figure 13 opens and the configuration may be altered.

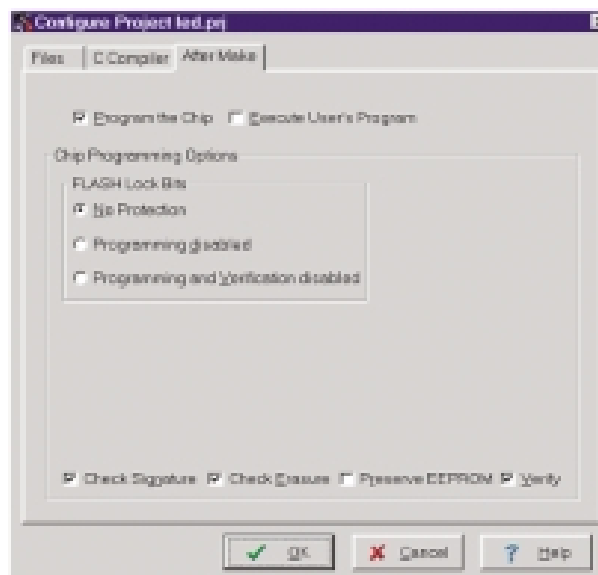
**Figure 13.** C Compiler Configuration



We may also select whether we wish to automatically program the target microprocessor after the Make or not.

This is chosen by selecting the “After Make” tab, which gives us the next window, shown in Figure 14.

**Figure 14.** After Make Configuration



For the purposes of this example, “Program the Chip” option must be checked.

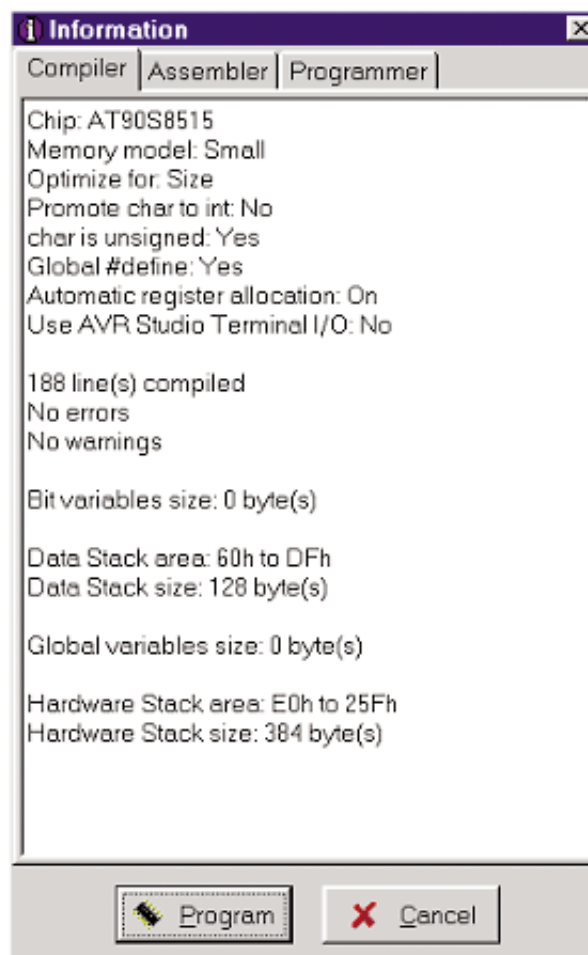
This will enable automatic programming of the AVR chip after the Make is complete.

## Making the Project

The “Project” Pull-down menu gives the Make option. Click on it or on the button on the toolbar.

After a successful compile and assembly, the Information window will be displayed as shown in Figure 15.

**Figure 15.** Information Window

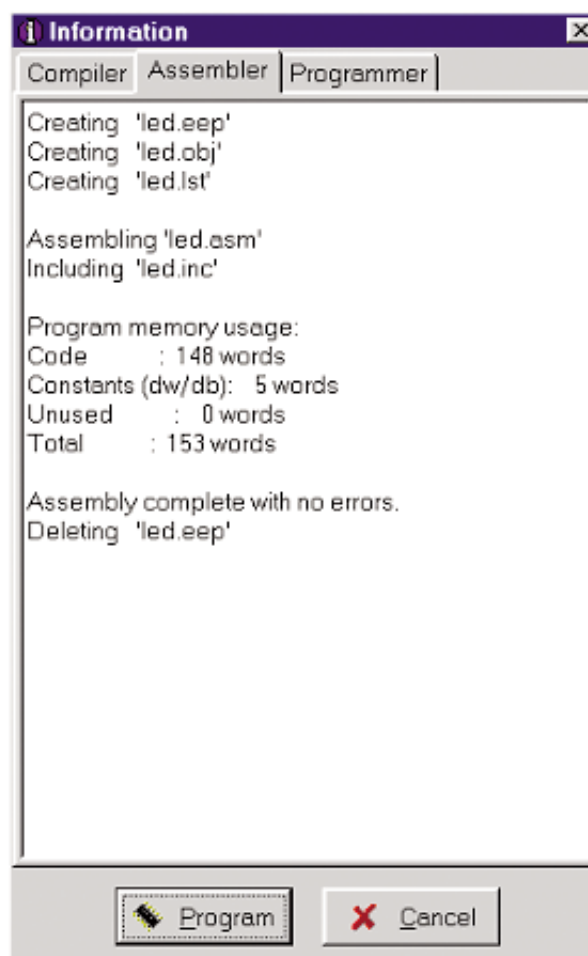


This window shows how the compiler used the RAM memory.

If the Assembler tab is clicked, the Assembler window shows the size of the assembled code as shown in Figure 16.

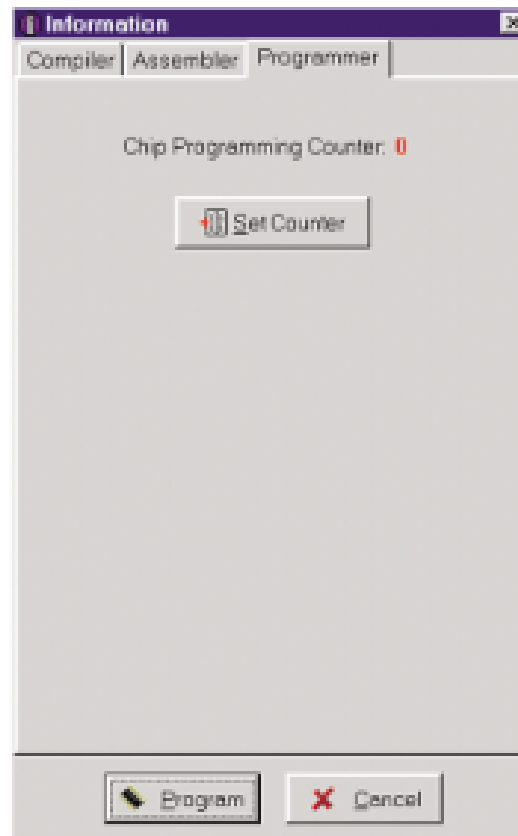


Figure 16. Assembler Information



Selecting the Programmer tab displays the value of the Chip Programming Counter. Pressing the Set Counter button can initialize this counter.

**Figure 17.** Programmer Information



If the Make process was successful, then power-up the STK500 starter kit and press the Program button to start the automatic chip programming.

After the programming process is complete, the code will start to execute in the target microcontroller on the STK500 starter kit.

## Short Reference

### Preparations

1. Install the CodeVisionAVR C Compiler
2. Install the Atmel AVR Studio Debugger
3. Install the Atmel STK500 Starter Kit
4. Configure the STK500 Programmer Support in the CodeVisionAVR IDE by selecting: Settings→Programmer→AVR Chip Programmer Type: STK500→Specify STK500.EXE Directory: C:\Program Files\Atmel\AVR Studio\STK500→Communication Port
5. Configure the AVR Studio Support in the CodeVisionAVR IDE by selecting: Settings→Debugger→Enter: C:\Program Files\Atmel\AVR Studio

## Getting Started

1. Create a new project by selecting:  
File→New→Select Project
2. Specify that the CodeWizardAVR will be used for producing the C source and project files: Use the CodeWizard?→Yes
3. In the CodeWizardAVR window specify the chip type and clock frequency:  
Chip→Chip: AT90S8515→Clock: 3.86MHz
4. Configure the I/O Ports: Ports→Port B→  
Data Direction: all Outputs→Output Value: all 1's
5. Configure Timer1: Timers→Timer1→  
Clock Value: 3.594kHz→Interrupt on: Timer1 Overflow→Val: 0xF8FB
6. Generate the C source, C project and CodeWizardAVR project files by selecting:  
File|Generate, Save and Exit→  
Create new directory: C:\cvavr\led→  
Save: led.c→Save: led.prj →Save: led.cwp
7. Edit the C source code
8. View or Modify the Project Configuration by selecting Project→Configure→  
After Make→Program the Chip
9. Compile the program by selecting:  
Project→Make
10. Automatically program the AT90S8515 chip on the STK500 starter kit:  
Apply power→Information→Program.

## Appendix A - The Source Code

```

/*****
This program was produced by the
CodeWizardAVR V1.0.1.8c Standard
Automatic Program Generator
© Copyright 1998-2001
Pavel Haiduc, HP InfoTech S.R.L.
http://infotech.ir.ro
e-mail: hpinfotech@xnet.ro, hpinfotech@xmail.ro

Project :
Version :
Date    :
Author  :
Company :
Comments:

Chip type      : AT90S8515
Clock frequency : 3.680000 MHz
Memory model   : Small
Internal SRAM size : 512
External SRAM size : 0
Data Stack size : 128
*****/

#include <90s8515.h>

```

```
// the LED 0 on PORTB will be on
unsigned char led_status=0xFE;

// Timer 1 overflow interrupt service routine
interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
    // Reinitialize Timer's 1 value
    TCNT1H=0xF8;
    TCNT1L=0xFB;
    // Place your code here
    // move the LED
    led_status<<=1;
    led_status|=1;
    if (led_status==0xFF) led_status=0xFE;
    // turn on the LED
    PORTB=led_status;
}

void main(void)
{
    // Input/Output Ports initialization
    // Port A
    PORTA=0x00;
    DDRA=0x00;

    // Port B
    PORTB=0xFF;
    DDRB=0xFF;

    // Port C
    PORTC=0x00;
    DDRC=0x00;

    // Port D
    PORTD=0x00;
    DDRD=0x00;

    // Timer/Counter 0 initialization
    // Clock source: System Clock
    // Clock value: Timer 0 Stopped
    // Mode: Output Compare
    // OC0 output: Disconnected
    TCCR0=0x00;
    TCNT0=0x00;

    // Timer/Counter 1 initialization
    // Clock source: System Clock
    // Clock value: 3.594 kHz
```

```
// Mode: Output Compare
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x05;
TCNT1H=0xF8;
TCNT1L=0xFB;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
GIMSK=0x00;
MCUCR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x80;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;

// Global enable interrupts
#asm("sei")

// the rest is done by TIMER1 overflow interrupts
while (1);
}
```



## **Atmel Headquarters**

### ***Corporate Headquarters***

2325 Orchard Parkway  
San Jose, CA 95131  
TEL (408) 441-0311  
FAX (408) 487-2600

### ***Europe***

Atmel SarL  
Route des Arsenaux 41  
Casa Postale 80  
CH-1705 Fribourg  
Switzerland  
TEL (41) 26-426-5555  
FAX (41) 26-426-5500

### ***Asia***

Atmel Asia, Ltd.  
Room 1219  
Chinachem Golden Plaza  
77 Mody Road Tsimhatsui  
East Kowloon  
Hong Kong  
TEL (852) 2721-9778  
FAX (852) 2722-1369

### ***Japan***

Atmel Japan K.K.  
9F, Tonetsu Shinkawa Bldg.  
1-24-8 Shinkawa  
Chuo-ku, Tokyo 104-0033  
Japan  
TEL (81) 3-3523-3551  
FAX (81) 3-3523-7581

## **Atmel Product Operations**

### ***Atmel Colorado Springs***

1150 E. Cheyenne Mtn. Blvd.  
Colorado Springs, CO 80906  
TEL (719) 576-3300  
FAX (719) 540-1759

### ***Atmel Grenoble***

Avenue de Rochepleine  
BP 123  
38521 Saint-Egreve Cedex, France  
TEL (33) 4-7658-3000  
FAX (33) 4-7658-3480

### ***Atmel Heilbronn***

Theresienstrasse 2  
POB 3535  
D-74025 Heilbronn, Germany  
TEL (49) 71 31 67 25 94  
FAX (49) 71 31 67 24 23

### ***Atmel Nantes***

La Chantrerie  
BP 70602  
44306 Nantes Cedex 3, France  
TEL (33) 0 2 40 18 18 18  
FAX (33) 0 2 40 18 19 60

### ***Atmel Rousset***

Zone Industrielle  
13106 Rousset Cedex, France  
TEL (33) 4-4253-6000  
FAX (33) 4-4253-6001

### ***Atmel Smart Card ICs***

Scottish Enterprise Technology Park  
East Kilbride, Scotland G75 0QR  
TEL (44) 1355-357-000  
FAX (44) 1355-242-743

---

### ***e-mail***

literature@atmel.com

### ***Web Site***

<http://www.atmel.com>

### ***BBS***

1-(408) 436-4309

## **© Atmel Corporation 2001.**

Atmel Corporation makes no warranty for the use of its products, other than those expressly contained in the Company's standard warranty which is detailed in Atmel's Terms and Conditions located on the Company's web site. The Company assumes no responsibility for any errors which may appear in this document, reserves the right to change devices or specifications detailed herein at any time without notice, and does not make any commitment to update the information contained herein. No licenses to patents or other intellectual property of Atmel are granted by the Company in connection with the sale of Atmel products, expressly or by implication. Atmel's products are not authorized for use as critical components in life support devices or systems.

Atmel®, AVR®, and AVR Studio® are registered trademarks Atmel Corporation.

Terms and product names in this document may be trademarks of others.



Printed on recycled paper.

2500A-10/01/xM

---

## C.5 LCD Display Datasheet

# L2034

## ■ Features of L2034 Series

- 20 characters × 4 lines
- STN gray type LCD is used
- 5 × 7 dot matrix + cursor
- 1/16 duty
- 5V single power supply

## ■ Specification

### A. Mechanical Characteristics

Item	Specifications	Unit
Module size (H×V×T) (Reflective type)	98.0×60.0×11.6	mm
Module size (H×V×T) (Built-in LED backlight type)	98.0×60.0×15.8	mm
Viewing area (H×V)	76.0×25.2	mm
Character size (5×7 dot, H×V)	2.95×4.15	mm
Dot size (H×V)	0.55×0.55	mm
Dot space	0.05	mm
Center to center dimension of mounting holes (H×V)	93.0×55.0	mm
Weight (Reflective type)	55	g
Weight (Built-in LED backlight type)	70	g

H : Horizontal, V : Vertical, T : Thickness (max.)

### B. Absolute Maximum Ratings

V<sub>SS</sub> = 0V

Item	Symbol	Conditions	Min.	Max.	Unit
Power supply voltage	V <sub>DD</sub>		-0.3	6.0	V
	V <sub>LC</sub>		V <sub>DD</sub> -12.0	V <sub>DD</sub>	V
Input voltage	V <sub>IN</sub>		-0.3	V <sub>DD</sub> +0.3	V
Operating temp.	T <sub>opr</sub>		0	+50	°C
Storage temp.	T <sub>stg</sub>		-20	+60	°C
Storage humidity		≤48hrs	+20	+85	%RH
		≤1000hrs	+20	+65	%RH

### C. Electrical Characteristics

V<sub>DD</sub> = 5V ± 5%, V<sub>SS</sub> = 0V, T<sub>a</sub> = 0°C to 50°C

Item	Symbol	Conditions	Min.	Typ.	Max.	Unit
Power supply voltage	V <sub>DD</sub>		4.75	5.00	5.25	V
	V <sub>DD</sub> -V <sub>LC</sub>		4.0	—	11.0	V
Input voltage	High	V <sub>IH1</sub>	2.2	—	V <sub>DD</sub>	V
	Low	V <sub>IL1</sub>	0	—	0.6	V
Output voltage	High	V <sub>OH1</sub>	-I <sub>OH</sub> = 0.205mA	2.4	—	V
	Low	V <sub>OL1</sub>	I <sub>OL</sub> = 1.2mA	—	0.4	V
Current consumption	I <sub>DD</sub>	T <sub>a</sub> = 25°C V <sub>DD</sub> = 5V V <sub>LC</sub> = 0.25V	—	2.9	4.0	mA
	I <sub>LC</sub>		—	1.2	2.0	mA
Clock oscillation frequency	f <sub>osc</sub>	Resistance oscillation	140	220	300	kHz

\* Applied to DB<sub>0</sub> ~ DB<sub>7</sub>, E, R/ $\overline{W}$ , RS

\*\* Applied to DB<sub>0</sub> ~ DB<sub>7</sub>

## D. Optical Characteristics

### D-1 Reflective type

Viewing angle : 6 o'clock (∅ = 0°), T<sub>a</sub> = 25°C, V<sub>opr</sub> = 4.75V

Item	Symbol	Conditions	Min.	Typ.	Max.	Unit
Viewing angle	θ <sub>1</sub>	C ≥ 2.0 ∅ = 0°	—	—	-15	deg.
	θ <sub>2</sub>		55	—	—	
	θ <sub>2</sub> - θ <sub>1</sub>		70	—	—	
Contrast	C	θ = 25°, ∅ = 0°	2	4	—	—
Response time (rise)	t <sub>on</sub>	θ = 0°	—	270	400	ms
Response time (fall)	t <sub>off</sub>	∅ = 0°	—	60	100	
Response time (rise)	t <sub>on</sub>	θ = 0°, ∅ = 0° T <sub>a</sub> = 0°C	—	720	1100	ms
Response time (fall)	t <sub>off</sub>	V <sub>opr</sub> = 5.0V	—	170	350	

### D-2 Transflective type

Viewing angle : 6 o'clock (∅ = 0°), T<sub>a</sub> = 25°C, V<sub>opr</sub> = 4.75V, Backlight OFF

Item	Symbol	Conditions	Min.	Typ.	Max.	Unit
Viewing angle	θ <sub>1</sub>	C ≥ 2.0 ∅ = 0°	—	—	-10	deg.
	θ <sub>2</sub>		50	—	—	
	θ <sub>2</sub> - θ <sub>1</sub>		60	—	—	
Contrast	C	θ = 25°, ∅ = 0°	2	4	—	—
Response time (rise)	t <sub>on</sub>	θ = 0°	—	270	400	ms
Response time (fall)	t <sub>off</sub>	∅ = 0°	—	60	100	
Response time (rise)	t <sub>on</sub>	θ = 0°, ∅ = 0° T <sub>a</sub> = 0°C	—	720	1100	ms
Response time (fall)	t <sub>off</sub>	V <sub>opr</sub> = 5.0V	—	170	350	

## E. Recommended Operating Voltage

The recommended value of (V<sub>opr</sub>) for an ambient temperature is as follows.

V<sub>opr</sub> = V<sub>DD</sub>-V<sub>LC</sub>

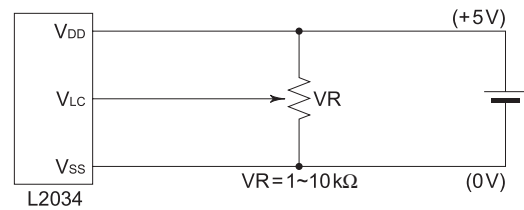
Temperature (°C)	0	25	50
V <sub>opr</sub> (V)	5.00	4.75	4.50



## ■ STN Reflective type

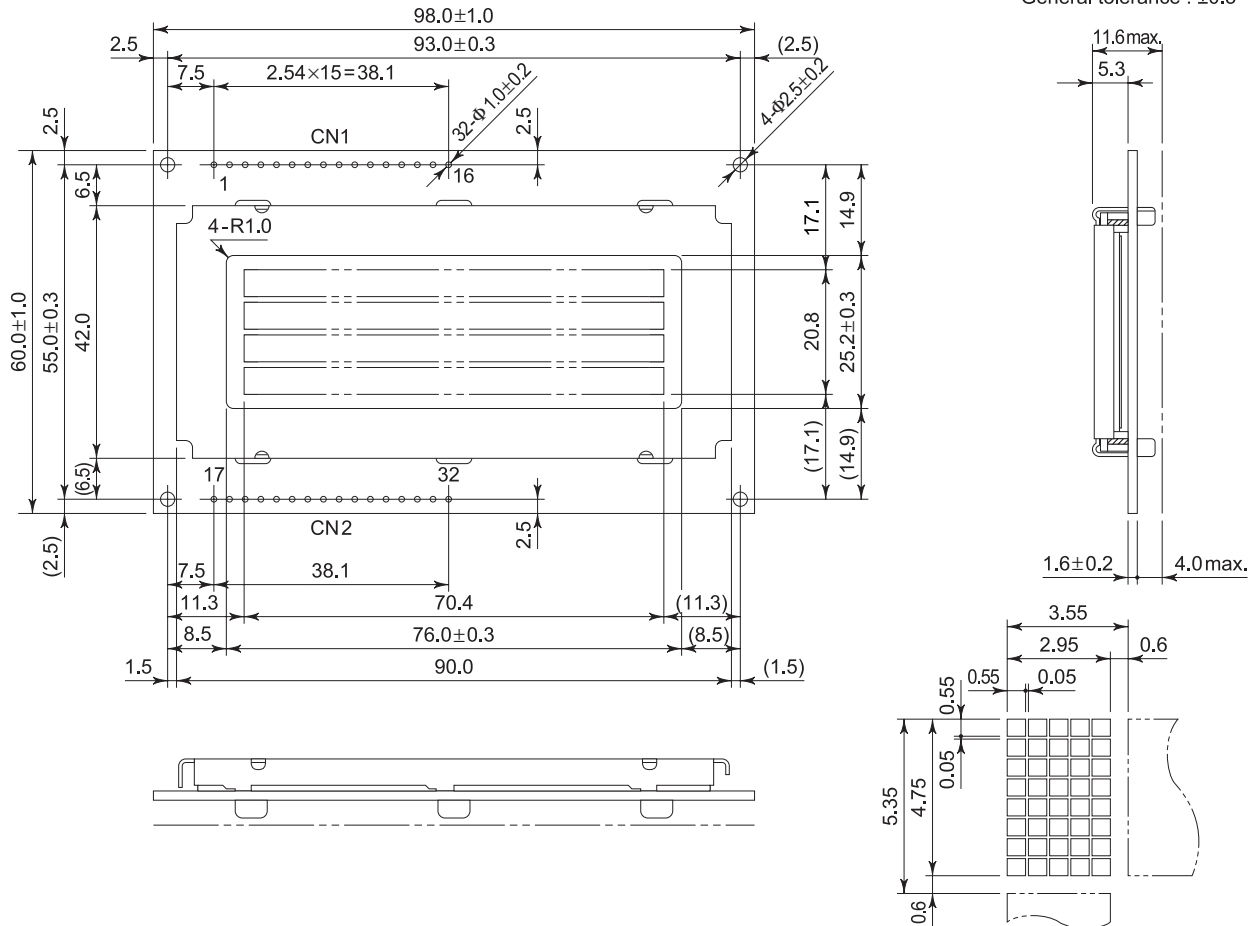
Item	L203400J000
Mechanical Characteristics	A
Absolute Maximum Ratings	B
Electrical Characteristics	C
Optical Characteristics	D-1
Recommended Operating Voltage	E

## F-1 Power Supply



## F-2 Dimensions

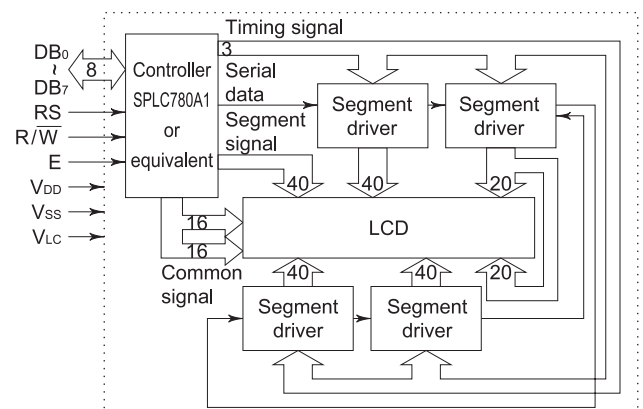
Unit : mm  
General tolerance :  $\pm 0.5$



## F-3 Pin Functions

No.	No.	Name	Function
1	17	V <sub>SS</sub>	GND
2	18	V <sub>DD</sub>	Power supply voltage +5V
3	19	V <sub>LC</sub>	Liquid crystal driving voltage
4	20	RS	L : Instruction code input H : Data input
5	21	R/W	L : Data write (LCM ← MPU) H : Data read (LCM → MPU)
6	22	E	Enable
7	23	DB <sub>0</sub>	Data bus line
8	24	DB <sub>1</sub>	Data bus line
9	25	DB <sub>2</sub>	Data bus line
10	26	DB <sub>3</sub>	Data bus line
11	27	DB <sub>4</sub>	Data bus line
12	28	DB <sub>5</sub>	Data bus line
13	29	DB <sub>6</sub>	Data bus line
14	30	DB <sub>7</sub>	Data bus line
15	31	NC	—
16	32	NC	—

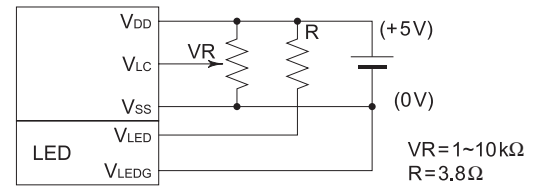
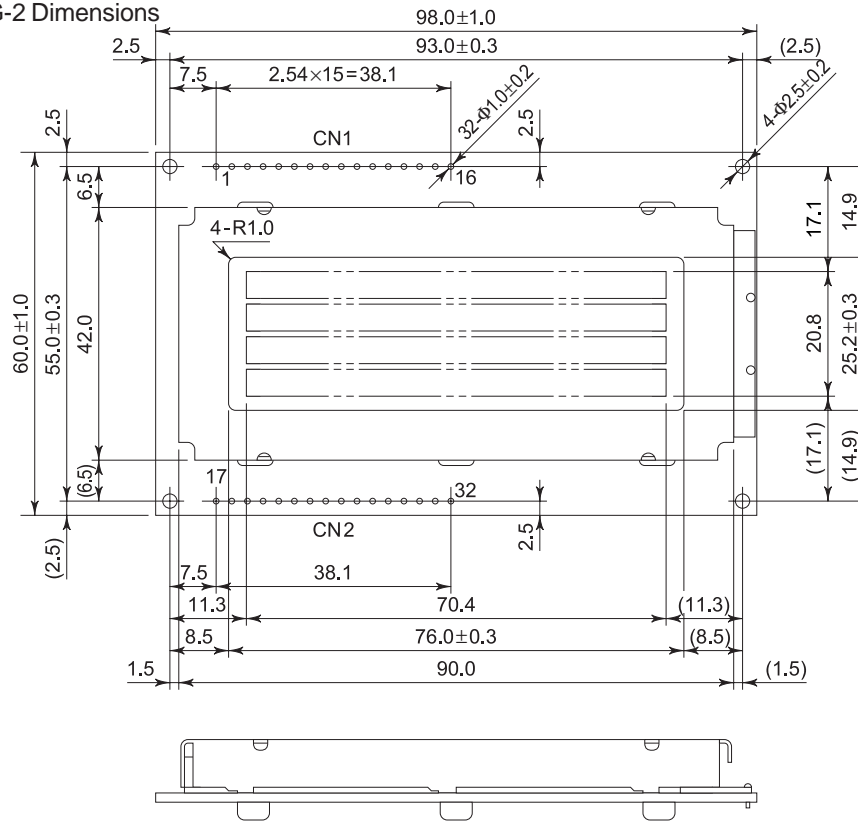
## F-4 Block Diagram



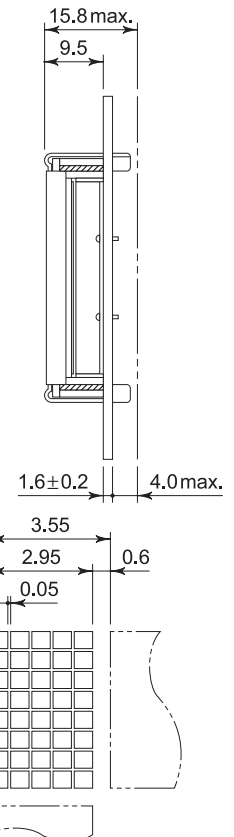
**L2034**

■ STN Transflective,  
Built-in LED Backlight type

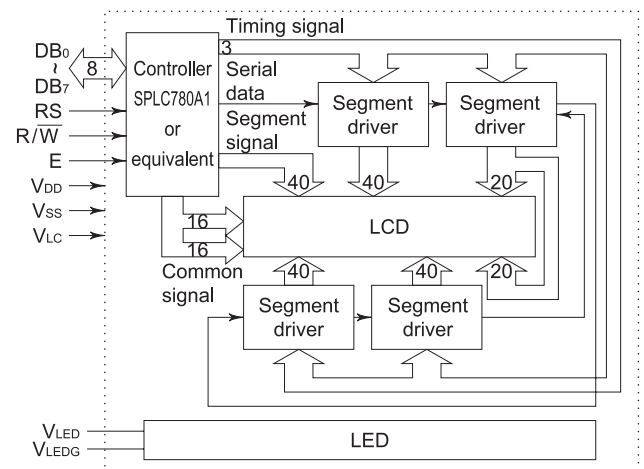
Item	L2034B1J000
Mechanical Characteristics	A
Absolute Maximum Ratings	B
Electrical Characteristics	C
Optical Characteristics	D-2
Recommended Operating Voltage	E

**G-1 Power Supply****G-2 Dimensions**

Unit : mm  
General tolerance :  $\pm 0.5$

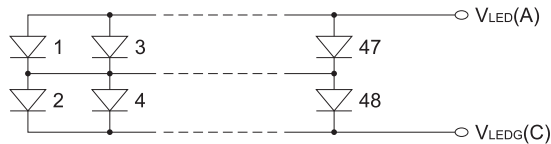
**G-3 Pin Functions**

No.	No.	Name	Function
1	17	VSS	GND
2	18	VDD	Power supply voltage +5V
3	19	VLC	Liquid crystal driving voltage
4	20	RS	L : Instruction code input H : Data input
5	21	R/W	L : Data write (LCM $\leftarrow$ MPU) H : Data read (LCM $\rightarrow$ MPU)
6	22	E	Enable
7	23	DB <sub>0</sub>	Data bus line
8	24	DB <sub>1</sub>	Data bus line
9	25	DB <sub>2</sub>	Data bus line
10	26	DB <sub>3</sub>	Data bus line
11	27	DB <sub>4</sub>	Data bus line
12	28	DB <sub>5</sub>	Data bus line
13	29	DB <sub>6</sub>	Data bus line
14	30	DB <sub>7</sub>	Data bus line
15	31	VLED	Anode
16	32	VLEDG	Cathode

**G-4 Block Diagram**

G-5 LED Backlight

G-5-1 LED Circuit Diagram

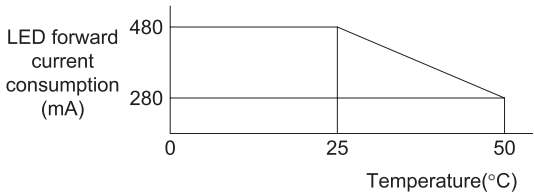


G-5-2 Absolute Maximum Ratings

Ta = 25°C

Item	Symbol	Specifications	Unit
LED forward current consumption*	IF	480	mA
LED reverse voltage	VR	8	V
LED allowable dissipation	PD	2.0	W

\* LED forward current consumption and operating temperature characteristics are as follows.



G-5-3 Electrical Characteristics

Ta = 25°C

Item	Symbol	Conditions	Min.	Typ.	Max.	Unit
LED forward input voltage	VF	IF = 240mA	3.8	4.1	4.4	V
LED reverse current	IR	VR = 8V	—	—	2.4	mA

G-5-4 Optical Characteristics

Ta = 25°C

Item	Symbol	Conditions	Specifications	Unit
Surface brightness (panel upper side)	Bp	IF = 240mA Vopr = 0V	4.5 min. 5 typ.	cd/m²
LED brightness	L	IF = 240mA	40 min. 50 typ.	cd/m²
LED service life			50,000 typ.	h
LED color			Yellowgreen	

---

## C.6 RS232 Transceiver Datasheet



**ANALOG  
DEVICES**

**EMI-EMC-Compliant,  $\pm 15$  kV ESD Protected,  
RS-232 Line Drivers/Receivers**

## ADM206E/ADM207E/ADM208E/ADM211E/ADM213E

### FEATURES

Complies with 89/336/EEC EMC Directive

ESD Protection to IEC1000-4-2 (801.2)

$\pm 8$  kV: Contact Discharge

$\pm 15$  kV: Air-Gap Discharge

$\pm 15$  kV: Human Body Model

Fast Transient Burst (EFT) Immunity (IEC1000-4-4)

Low EMI Emissions (EN55022)

Eliminates Costly TranZorbs®

230 kbits/s Data Rate Guaranteed

Single 5 V Power Supply

Shutdown Mode 1  $\mu$ W

Plug-In Upgrade for MAX2xxE

Space Saving TSSOP Package Available

### APPLICATIONS

Laptop Computers

Notebook Computers

Printers

Peripherals

Modems

### GENERAL DESCRIPTION

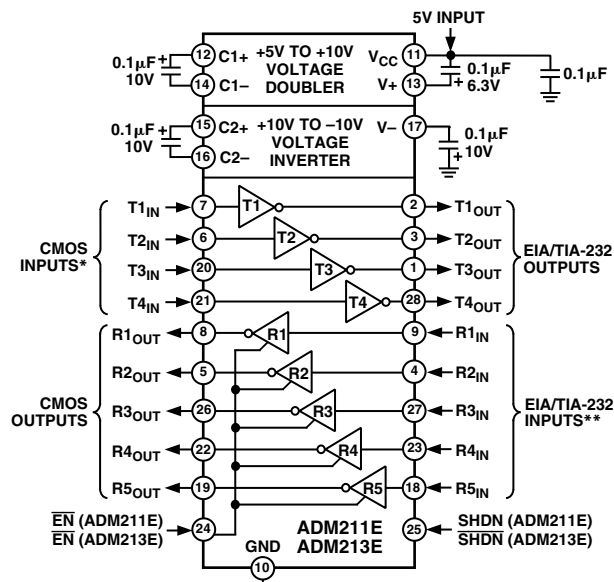
The ADM2xxE is a family of robust RS-232 and V.28 interface devices that operates from a single 5 V power supply. These products are suitable for operation in harsh electrical environments and are compliant with the EU directive on EMC (89/336/EEC). The level of emissions and immunity are both in compliance. EM immunity includes ESD protection in excess of  $\pm 15$  kV on all I-O lines (1000-4-2), fast transient burst protection (1000-4-4) and radiated immunity (1000-4-3). EM emissions include radiated and conducted emissions as required by Information Technology Equipment EN55022, CISPR22.

All devices fully conform to the EIA-232E and CCITT V.28 specifications and operate at data rates up to 230 kbps.

Shutdown and enable control pins are provided on some of the products. See Table I.

The shutdown function on the ADM211E disables the charge pump and all transmitters and receivers. On the ADM213E the

### FUNCTIONAL BLOCK DIAGRAM



NOTES:  
\* INTERNAL 400k $\Omega$  PULL-UP RESISTOR ON EACH CMOS INPUT  
\*\* INTERNAL 5k $\Omega$  PULL-DOWN RESISTOR ON EACH RS-232 INPUT

charge pump, all transmitters, and three of the five receivers are disabled. The remaining two receivers remain active, thereby allowing monitoring of peripheral devices. This feature allows the device to be shut down until a peripheral device begins communication. The active receivers can alert the processor which can then take the ADM213E out of the shutdown mode.

Operating from a single 5 V supply, four external 0.1  $\mu$ F capacitors are required.

The ADM207E and ADM208E are available in 24-lead DIP, SO, SSOP, and TSSOP packages. The ADM211E and ADM213E are available in 28-lead SO, SSOP, and TSSOP packages.

All products are backward-compatible with earlier ADM2xx products, facilitating easy upgrading of older designs.

Table I. Selection Table

Model	Supply Voltage	Drivers	Receivers	ESD Protection	Shutdown	Enable	Packages
ADM206E	5 V	4	3	$\pm 15$ kV	Yes	Yes	R-24
ADM207E	5 V	5	3	$\pm 15$ kV	No	No	N, R, RS, RU-24
ADM208E	5 V	4	4	$\pm 15$ kV	No	No	N, R, RS, RU-24
ADM211E	5 V	4	5	$\pm 15$ kV	Yes	Yes	R, RS, RU-28
ADM213E	5 V	4	5	$\pm 15$ kV	Yes ( $\overline{\text{SD}}$ )*	Yes (EN)	R, RS, RU-28

\*Two receivers active.

### REV. D

Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices. Trademarks and registered trademarks are the property of their respective owners.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.  
Tel: 781/329-4700 [www.analog.com](http://www.analog.com)  
Fax: 781/461-3113 © 2005 Analog Devices, Inc. All rights reserved.

# ADM206E/ADM207E/ADM208E/ADM211E/ADM213E—SPECIFICATIONS

( $V_{CC} = 5.0\text{ V} \pm 10\%$ ,  $C1-C4 = 0.1\text{ }\mu\text{F}$ . All specifications  $T_{MIN}$  to  $T_{MAX}$  unless otherwise noted.)

Parameter	Min	Typ	Max	Unit	Test Conditions/Comments
Operating Voltage Range	4.5	5.0	5.5	V	No Load
$V_{CC}$ Power Supply Current		3.5	13	mA	
Shutdown Supply Current		0.2	10	$\mu\text{A}$	
Input Pull-Up Current		10	25	$\mu\text{A}$	$T_{IN} = \text{GND}$ $T_{IN}, \text{EN}, \overline{\text{EN}}, \text{SHDN}, \overline{\text{SHDN}},$ $T_{IN}$ $\text{EN}, \overline{\text{EN}}, \text{SHDN}, \overline{\text{SHDN}}$ $I_{OUT} = 1.6\text{ mA}$ $I_{OUT} = -40\text{ }\mu\text{A}$ $\overline{\text{EN}} = V_{CC}, \text{EN} = \text{GND}, 0\text{ V} \leq R_{OUT} \leq V_{CC}$
Input Logic Threshold Low, $V_{INL}$			0.8	V	
Input Logic Threshold High, $V_{INH}$	2.0			V	
Input Logic Threshold High, $V_{INH}$	2.0			V	
CMOS Output Voltage Low, $V_{OL}$			0.4	V	
CMOS Output Voltage High, $V_{OH}$	3.5			V	
CMOS Output Leakage Current		0.05	$\pm 10$	$\mu\text{A}$	
EIA-232 Input Voltage Range*	-30		+30	V	$T_A = 0^\circ\text{C to } 85^\circ\text{C}$ All Transmitter Outputs Loaded with $3\text{ k}\Omega$ to Ground $V_{CC} = 0\text{ V}, V_{OUT} = \pm 2\text{ V}$
EIA-232 Input Threshold Low	0.8	1.3		V	
EIA-232 Input Threshold High		2.0	2.4	V	
EIA-232 Input Hysteresis		0.65		V	
EIA-232 Input Resistance	3	5	7	$\text{k}\Omega$	
Output Voltage Swing	$\pm 5.0$	$\pm 9.0$		V	
Transmitter Output Resistance	300			$\Omega$	
RS-232 Output Short Circuit Current	$\pm 6$	$\pm 20$	$\pm 60$	mA	
Maximum Data Rate	230			kbps	$R_L = 3\text{ k}\Omega$ to $7\text{ k}\Omega$ , $C_L = 50\text{ pF}$ to $2500\text{ pF}$
Receiver Propagation Delay					
TPHL, TPLH		0.4	2	$\mu\text{s}$	$C_L = 150\text{ pF}$
Receiver Output Enable Time, $t_{ER}$		120		ns	
Receiver Output Disable Time, $t_{DR}$		120		ns	$R_L = 3\text{ k}\Omega$ , $C_L = 2500\text{ pF}$ $R_L = 3\text{ k}\Omega$ , $C_L = 50\text{ pF}$ to $2500\text{ pF}$ Measured from $+3\text{ V}$ to $-3\text{ V}$ or $-3\text{ V}$ to $+3\text{ V}$
Transmitter Propagation Delay					
TPHL, TPLH		1		$\mu\text{s}$	
Transition Region Slew Rate		8		V/ $\mu\text{s}$	
ESD Protection (I-O Pins)		$\pm 15$		kV	Human Body Model IEC1000-4-2 Air Discharge IEC1000-4-2 Contact Discharge IEC1000-4-3
		$\pm 15$		kV	
		$\pm 8$		kV	
EMI Immunity		10		V/m	

\*Guaranteed by design.

Specifications subject to change without notice.

Table II. ADM211E Truth Table

SHDN	$\overline{\text{EN}}$	Status	$T_{OUT1-4}$	$R_{OUT1-5}$
0	0	Normal Operation	Enabled	Enabled
0	1	Normal Operation	Enabled	Disabled
1	X	Shutdown	Disabled	Disabled

X = Don't Care.

Table III. ADM213E Truth Table

SHDN	EN	Status	$T_{OUT1-4}$	$R_{OUT1-3}$	$R_{OUT4-5}$
0	0	Shutdown	Disabled	Disabled	Disabled
0	1	Shutdown	Disabled	Disabled	Enabled
1	0	Normal Operation	Enabled	Disabled	Disabled
1	1	Normal Operation	Enabled	Enabled	Enabled

# ADM206E/ADM207E/ADM208E/ADM211E/ADM213E

## ABSOLUTE MAXIMUM RATINGS\*

(T<sub>A</sub> = 25°C unless otherwise noted.)

V<sub>CC</sub> ..... -0.3 V to +6 V  
V<sub>+</sub> ..... (V<sub>CC</sub>-0.3 V) to +14 V  
V<sub>-</sub> ..... +0.3 V to -14 V

### Input Voltages

T<sub>IN</sub> ..... -0.3 V to (V<sub>+</sub>, +0.3 V)  
R<sub>IN</sub> ..... ±30 V

### Output Voltages

T<sub>OUT</sub> ..... ±15 V  
R<sub>OUT</sub> ..... -0.3 V to (V<sub>CC</sub> +0.3 V)

### Short-Circuit Duration

T<sub>OUT</sub> ..... Continuous

### Power Dissipation

N-24 PDIP (Derate 13.5 mW/°C above 70°C) .. 1000 mW  
R-24 SOIC (Derate 12 mW/°C above 70°C) ..... 900 mW

RS-24 SSOP (Derate 12 mW/°C above 70°C) ..... 850 mW  
RU-24 TSSOP (Derate 12 mW/°C above 70°C) ... 900 mW  
R-28 SOIC (Derate 12 mW/°C above 70°C) ..... 900 mW  
RS-28 SSOP (Derate 10 mW/°C above 70°C) ..... 900 mW  
RU-28 TSSOP (Derate 12 mW/°C above 70°C) ... 900 mW

### Operating Temperature Range

Industrial (A Version) ..... -40°C to +85°C

### Storage Temperature Range

Lead Temperature (Soldering, 10 sec) ..... -65°C to +150°C

### ESD Rating (MIL-STD-883B) (I-O Pins)

ESD Rating (IEC1000-4-2 Air) (I-O Pins) ..... ±15 kV

### ESD Rating (IEC1000-4-2 Contact) (I-O Pins)

ESD Rating (IEC1000-4-2 Contact) (I-O Pins) ..... ±8 kV

\*This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operation sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect reliability.

## CAUTION

ESD (electrostatic discharge) sensitive device. Electrostatic charges as high as 4000 V readily accumulate on the human body and test equipment and can discharge without detection. Although the ADM206E/ADM207E/ADM208E/ADM211E/ADM213E features proprietary ESD protection circuitry, permanent damage may occur on devices subjected to high-energy electrostatic discharges. Therefore, proper ESD precautions are recommended to avoid performance degradation or loss of functionality.



# ADM206E/ADM207E/ADM208E/ADM211E/ADM213E

## ORDERING GUIDE

Model	Temperature Range	Package Description	Package Option
ADM206EAR	–40°C to +85°C	SOIC	R-24
ADM206EAR-REEL	–40°C to +85°C	SOIC	R-24
ADM206EARZ*	–40°C to +85°C	SOIC	R-24
ADM206EARZ-REEL*	–40°C to +85°C	SOIC	R-24
ADM207EAN	–40°C to +85°C	PDIP	N-24
ADM207EANZ*	–40°C to +85°C	PDIP	N-24
ADM207EAR	–40°C to +85°C	SOIC	R-24
ADM207EAR-REEL	–40°C to +85°C	SOIC	R-24
ADM207EARZ*	–40°C to +85°C	SOIC	R-24
ADM207EARZ-REEL*	–40°C to +85°C	SOIC	R-24
ADM207EARS	–40°C to +85°C	SSOP	RS-24
ADM207EARS-REEL	–40°C to +85°C	SSOP	RS-24
ADM207EARSZ*	–40°C to +85°C	SSOP	RS-24
ADM207EARSZ-REEL*	–40°C to +85°C	SSOP	RS-24
ADM207EARU	–40°C to +85°C	TSSOP	RU-24
ADM207EARU-REEL	–40°C to +85°C	TSSOP	RU-24
ADM207EARU-REEL7	–40°C to +85°C	TSSOP	RU-24
ADM208EAN	–40°C to +85°C	PDIP	N-24
ADM208EANZ*	–40°C to +85°C	PDIP	N-24
ADM208EAR	–40°C to +85°C	SOIC	R-24
ADM208EAR-REEL	–40°C to +85°C	SOIC	R-24
ADM208EARZ*	–40°C to +85°C	SOIC	R-24
ADM208EARZ-REEL*	–40°C to +85°C	SOIC	R-24
ADM208EARS	–40°C to +85°C	SSOP	RS-24
ADM208EARS-REEL	–40°C to +85°C	SSOP	RS-24
ADM208EARSZ*	–40°C to +85°C	SSOP	RS-24
ADM208EARSZ-REEL*	–40°C to +85°C	SSOP	RS-24
ADM208EARU	–40°C to +85°C	TSSOP	RU-24
ADM208EARU-REEL	–40°C to +85°C	TSSOP	RU-24
ADM208EARU-REEL7	–40°C to +85°C	TSSOP	RU-24
ADM211EAR	–40°C to +85°C	SOIC	R-28
ADM211EAR-REEL	–40°C to +85°C	SOIC	R-28
ADM211EARZ*	–40°C to +85°C	SOIC	R-28
ADM211EARZ-REEL*	–40°C to +85°C	SOIC	R-28
ADM211EARS	–40°C to +85°C	SSOP	RS-28
ADM211EARS-REEL	–40°C to +85°C	SSOP	RS-28
ADM211EARSZ*	–40°C to +85°C	SSOP	RS-28
ADM211EARSZ-REEL*	–40°C to +85°C	SSOP	RS-28
ADM211EARU	–40°C to +85°C	TSSOP	RU-28
ADM211EARU-REEL	–40°C to +85°C	TSSOP	RU-28
ADM211EARU-REEL7	–40°C to +85°C	TSSOP	RU-28
ADM211EARUZ*	–40°C to +85°C	TSSOP	RU-28
ADM211EARUZ-REEL*	–40°C to +85°C	TSSOP	RU-28
ADM211EARUZ-REEL7*	–40°C to +85°C	TSSOP	RU-28
ADM213EAR	–40°C to +85°C	SOIC	R-28
ADM213EAR-REEL	–40°C to +85°C	SOIC	R-28
ADM213EARZ*	–40°C to +85°C	SOIC	R-28
ADM213EARZ-REEL*	–40°C to +85°C	SOIC	R-28
ADM213EARS	–40°C to +85°C	SSOP	RS-28
ADM213EARS-REEL	–40°C to +85°C	SSOP	RS-28
ADM213EARSZ*	–40°C to +85°C	SSOP	RS-28
ADM213EARSZ-REEL*	–40°C to +85°C	SSOP	RS-28
ADM213EARU	–40°C to +85°C	TSSOP	RU-28
ADM213EARU-REEL	–40°C to +85°C	TSSOP	RU-28
ADM213EARU-REEL7	–40°C to +85°C	TSSOP	RU-28
ADM213EARUZ*	–40°C to +85°C	TSSOP	RU-28
ADM213EARUZ-REEL*	–40°C to +85°C	TSSOP	RU-28
ADM213EARUZ-REEL7*	–40°C to +85°C	TSSOP	RU-28

\*Z = Pb-free part.



# ADM206E/ADM207E/ADM208E/ADM211E/ADM213E

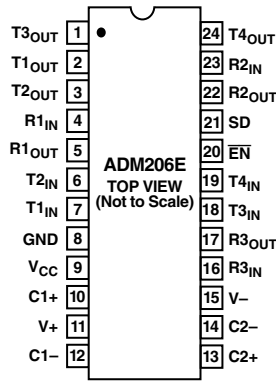


Figure 1. ADM206E DIP/SOIC/SSOP Pin Configuration

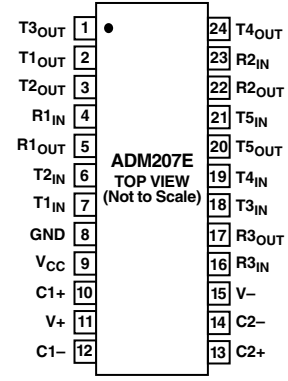
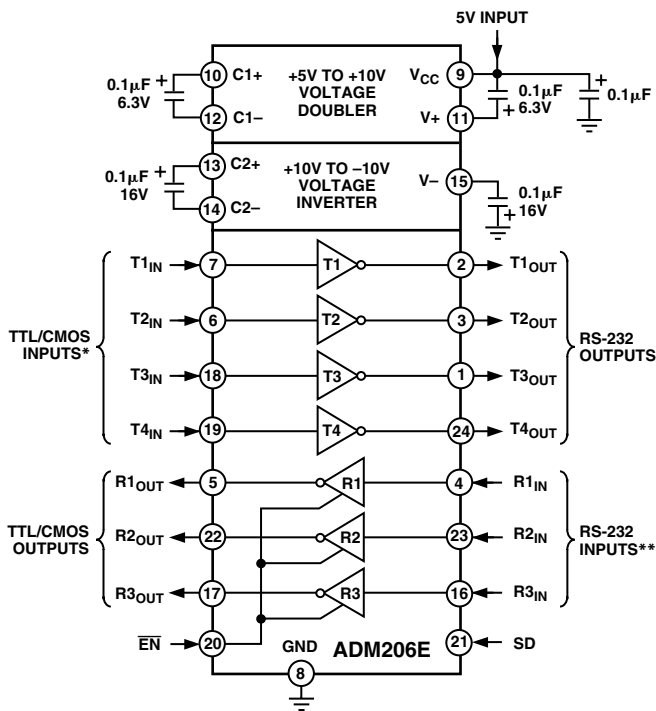
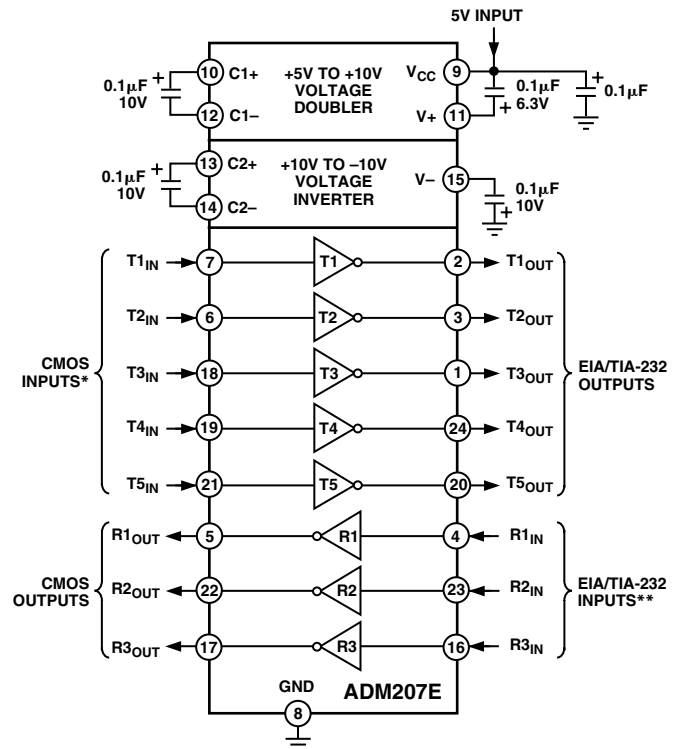


Figure 3. ADM207E Pin Configuration



\*INTERNAL 400kΩ PULL-UP RESISTOR ON EACH TTL/CMOS INPUT  
 \*\*INTERNAL 5kΩ PULL-DOWN RESISTOR ON EACH RS-232 INPUT

Figure 2. ADM206E Typical Operating Circuit



\*INTERNAL 400kΩ PULL-UP RESISTOR ON EACH CMOS INPUT  
 \*\*INTERNAL 5kΩ PULL-DOWN RESISTOR ON EACH RS-232 INPUT

Figure 4. ADM207E Typical Operating Circuit

# ADM206E/ADM207E/ADM208E/ADM211E/ADM213E

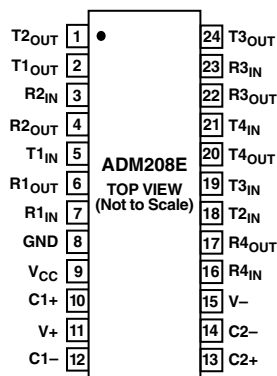


Figure 5. ADM208E Pin Configuration

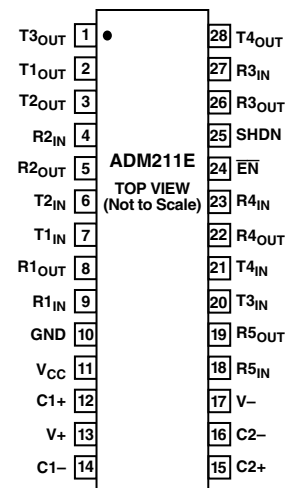
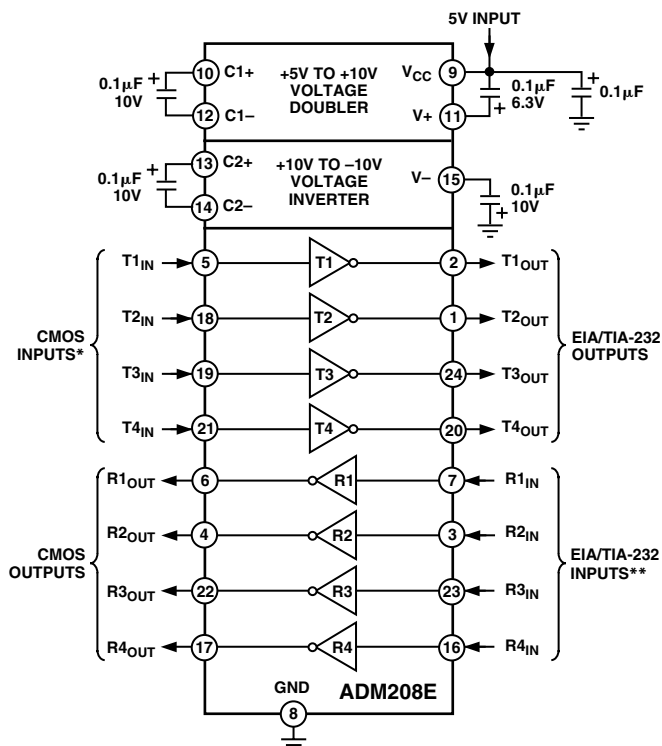
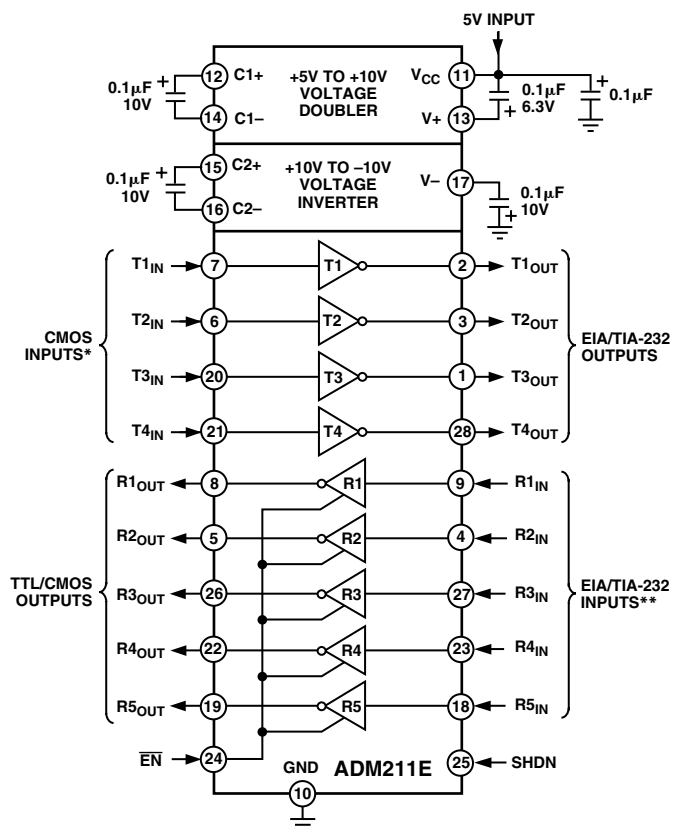


Figure 7. ADM211E Pin Configuration



\*INTERNAL 400k $\Omega$  PULL-UP RESISTOR ON EACH CMOS INPUT  
\*\*INTERNAL 5k $\Omega$  PULL-DOWN RESISTOR ON EACH RS-232 INPUT

Figure 6. ADM208E Typical Operating Circuit



\*INTERNAL 400k $\Omega$  PULL-UP RESISTOR ON EACH CMOS INPUT  
\*\*INTERNAL 5k $\Omega$  PULL-DOWN RESISTOR ON EACH RS-232 INPUT

Figure 8. ADM211E Typical Operating Circuit

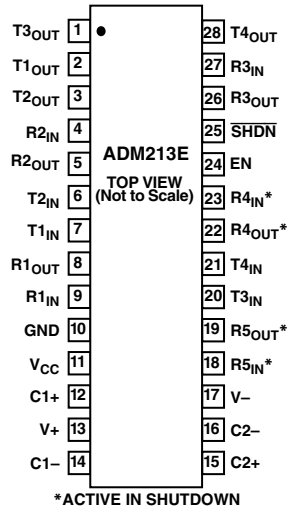
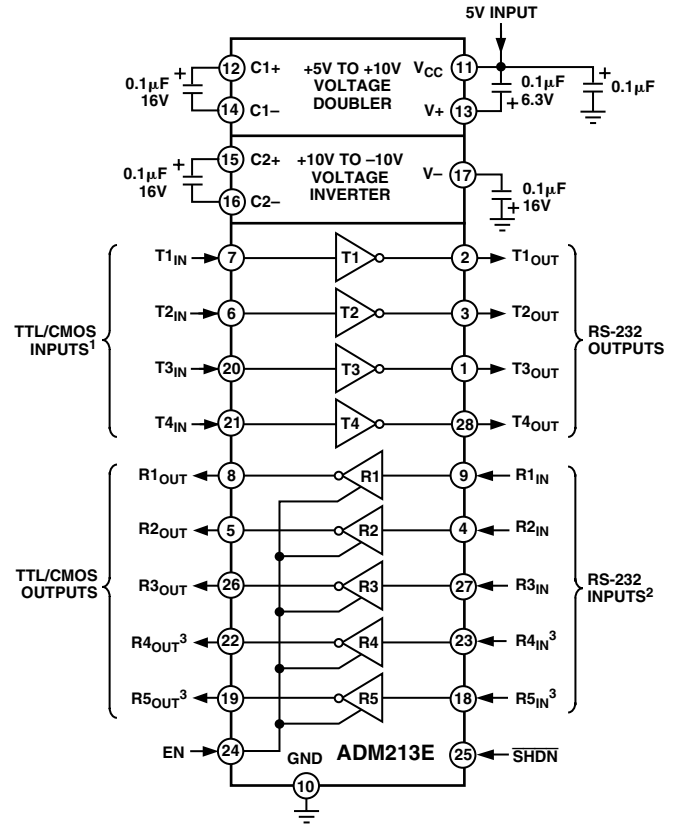


Figure 9. ADM213E Pin Configuration



## NOTES

- <sup>1</sup>INTERNAL 400kΩ PULL-UP RESISTOR ON EACH CMOS INPUT
- <sup>2</sup>INTERNAL 5kΩ PULL-DOWN RESISTOR ON EACH RS-232 INPUT
- <sup>3</sup>ACTIVE IN SHUTDOWN

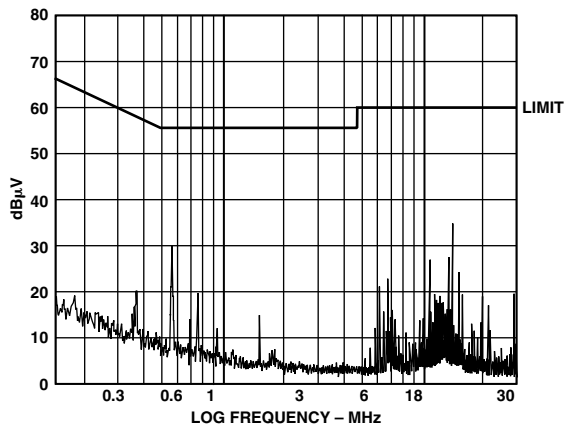
Figure 10. ADM213E Typical Operating Circuit

## PIN FUNCTION DESCRIPTIONS

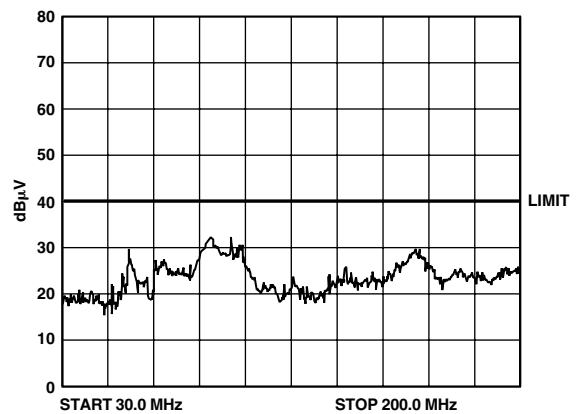
Mnemonic	Function
V <sub>CC</sub>	Power Supply Input: 5 V ± 10%.
V+	Internally Generated Positive Supply (+9 V nominal).
V-	Internally Generated Negative Supply (-9 V nominal).
GND	Ground Pin. Must Be Connected to 0 V.
C1+, C1-	External Capacitor 1 is connected between these pins. 0.1 μF capacitor is recommended but larger capacitors up to 47 μF may be used.
C2+, C2-	External Capacitor 2 is connected between these pins. 0.1 μF capacitor is recommended but larger capacitors up to 47 μF may be used.
T <sub>IN</sub>	Transmitter (Driver) Inputs. These inputs accept TTL/CMOS levels. An internal 400 kΩ pull-up resistor to V <sub>CC</sub> is connected on each input.
T <sub>OUT</sub>	Transmitter (Driver) Outputs. These are RS-232 signal levels (Typically ±9 V).
R <sub>IN</sub>	Receiver Inputs. These inputs accept RS-232 signal levels. An internal 5 kΩ pull-down resistor to GND is connected on each input.
R <sub>OUT</sub>	Receiver Outputs. These are CMOS output logic levels.
EN/ $\overline{\text{EN}}$	Receiver Enable (Active High on ADM213E, Active Low on ADM211E); This input is used to enable/disable the receiver outputs. With $\overline{\text{EN}}$ = Low ADM211E (EN = High ADM213E), the receiver outputs are enabled. With $\overline{\text{EN}}$ = High (EN = Low ADM213E), the receiver outputs are placed in a high impedance state.
$\overline{\text{SHDN}}$ /SHDN	Shutdown Control (Active Low on ADM213E, Active High on ADM211E); Refer to Table II. In shutdown the charge pump is disabled, the transmitter outputs are turned off and all receiver outputs (ADM211E), receivers R1, R2, R3 (ADM213E) are placed in a high impedance state. Receivers R4 and R5 on the ADM213E continue to operate normally during shutdown. Power consumption in shutdown for all parts reduces to 5 μW.

# ADM206E/ADM207E/ADM208E/ADM211E/ADM213E

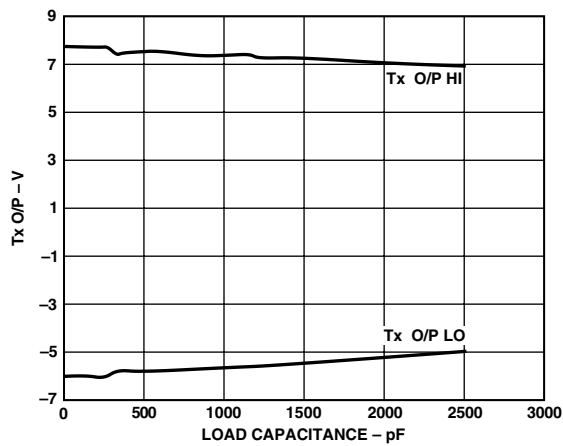
## Typical Performance Characteristics



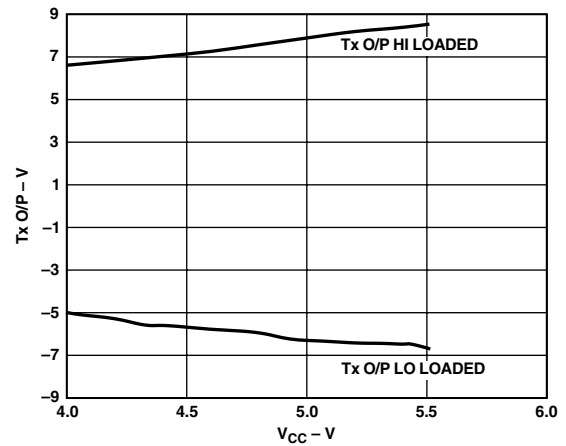
TPC 1. EMC Conducted Emissions



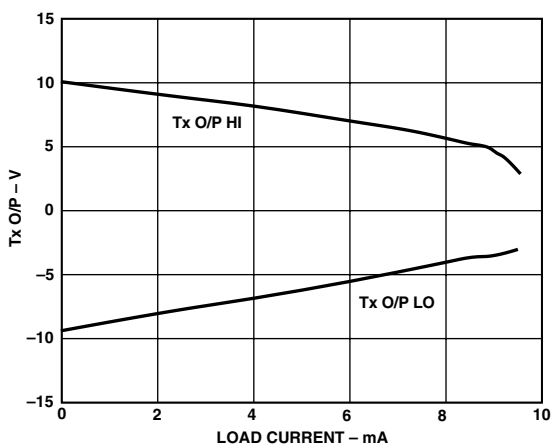
TPC 4. EMC Radiated Emissions



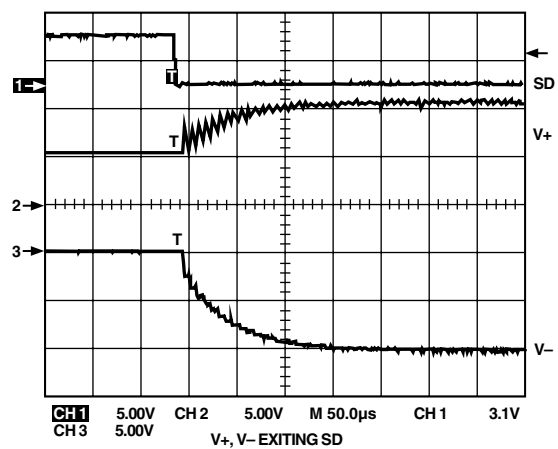
TPC 2. Transmitter Output Voltage High/Low vs. Load Capacitance @ 230 kbps



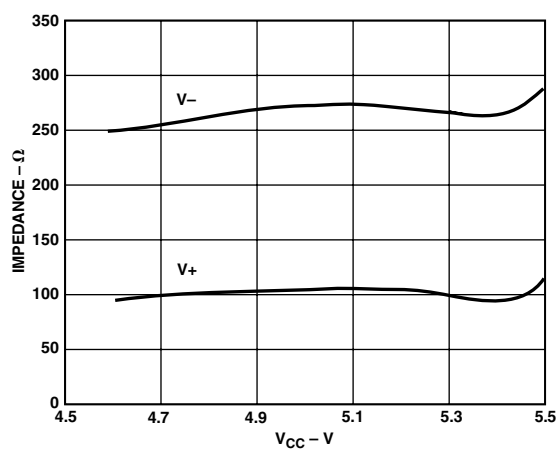
TPC 5. Transmitter Output Voltage vs.  $V_{CC}$



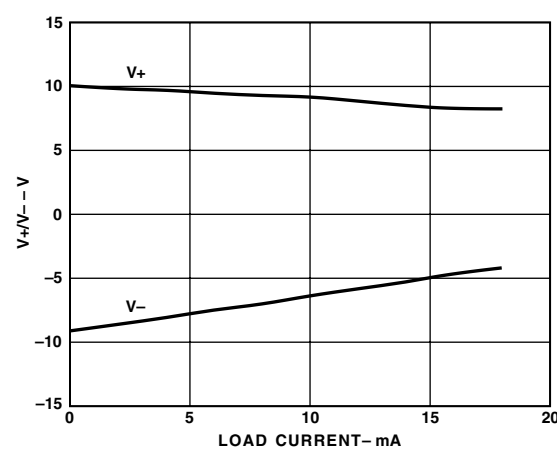
TPC 3. Transmitter Output Voltage vs. Load Current



TPC 6. Charge Pump  $V_+$ ,  $V_-$  Exiting Shutdown



TPC 7. Charge Pump Impedance vs. V<sub>CC</sub>



TPC 8. Charge Pump V<sub>+</sub>, V<sub>-</sub> vs. Current

# ADM206E/ADM207E/ADM208E/ADM211E/ADM213E

## GENERAL DESCRIPTION

The ADM206E/ADM207E/ADM208E/ADM211E/ADM213E are ruggedized RS-232 line drivers/receivers which operate from a single 5 V supply. Step-up voltage converters coupled with level shifting transmitters and receivers allow RS-232 levels to be developed while operating from a single 5 V supply.

Features include low power consumption, high transmission rates, and compatibility with the EU directive on electromagnetic compatibility. EM compatibility includes protection against radiated and conducted interference, including high levels of electrostatic discharge.

All RS-232 inputs and outputs contain protection against electrostatic discharges up to  $\pm 15$  kV and electrical fast transients up to  $\pm 2$  kV. This ensures compliance to IE1000-4-2 and IEC1000-4-4 requirements.

The devices are ideally suited for operation in electrically harsh environments or where RS-232 cables are frequently being unplugged. They are also immune to high RF field strengths without special shielding precautions.

Emissions are also controlled to within very strict limits. CMOS technology is used to keep the power dissipation to an absolute minimum allowing maximum battery life in portable applications. The ADMxxE is a modification, enhancement, and improvement to the AD230-AD241 family and its derivatives. It is essentially plug-in compatible and does not have materially different applications.

## CIRCUIT DESCRIPTION

The internal circuitry consists of four main sections.

1. A charge pump voltage converter
2. 5 V logic to EIA-232 transmitters
3. EIA-232 to 5 V logic receivers
4. Transient protection circuit on all I-O lines

### Charge Pump DC-DC Voltage Converter

The charge pump voltage converter consists of an 200 kHz oscillator and a switching matrix. The converter generates a  $\pm 10$  V supply from the input 5 V level. This is done in two stages using a switched capacitor technique as illustrated below. First, the 5 V input supply is doubled to 10 V using capacitor C1 as the charge storage element. The 10 V level is then inverted to generate  $-10$  V using C2 as the storage element.

Capacitors C3 and C4 are used to reduce the output ripple. If desired, larger capacitors (up to 47  $\mu$ F) can be used for capacitors C1-C4. This facilitates direct substitution with older generation charge pump RS-232 transceivers.

The V+ and V- supplies may also be used to power external circuitry if the current requirements are small. Please refer to TPC 9 in the Typical Performance Characteristics section.

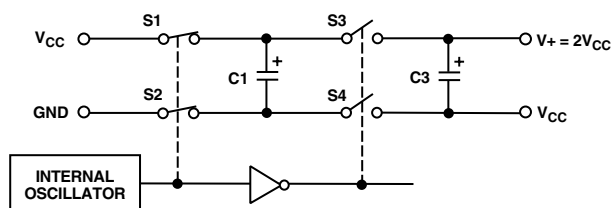


Figure 11. Charge Pump Voltage Doubler

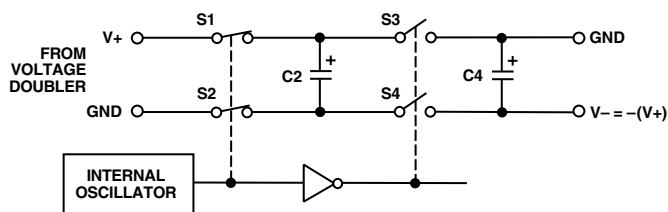


Figure 12. Charge Pump Voltage Inverter

### Transmitter (Driver) Section

The drivers convert 5 V logic input levels into EIA-232 output levels. With  $V_{CC} = 5$  V and driving an EIA-232 load, the output voltage swing is typically  $\pm 9$  V.

Unused inputs may be left unconnected, as an internal 400 k $\Omega$  pull-up resistor pulls them high forcing the outputs into a low state. The input pull-up resistors typically source 8  $\mu$ A when grounded, so unused inputs should either be connected to  $V_{CC}$  or left unconnected in order to minimize power consumption.

### Receiver Section

The receivers are inverting level shifters which accept EIA-232 input levels and translate them into 5 V logic output levels. The inputs have internal 5 k $\Omega$  pull-down resistors to ground and are also protected against overvoltages of up to  $\pm 25$  V. The guaranteed switching thresholds are 0.4 V minimum and 2.4 V maximum. Unconnected inputs are pulled to 0 V by the internal 5 k $\Omega$  pull-down resistor. This, therefore, results in a Logic 1 output level for unconnected inputs or for inputs connected to GND.

The receivers have Schmitt trigger input with a hysteresis level of 0.65 V. This ensures error-free reception for both noisy inputs and for inputs with slow transition times.

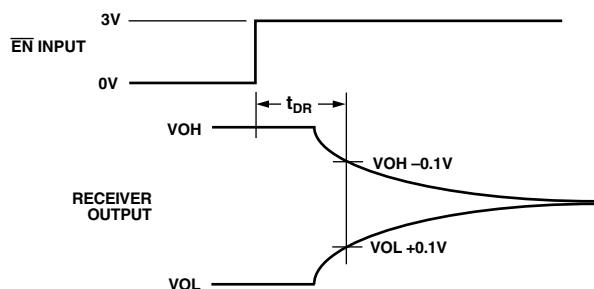
### ENABLE AND SHUTDOWN

Table II and Table III show the truth tables for the enable and shutdown control signals. The enable function is intended to facilitate data bus connections where it is desirable to three-state the receiver outputs. In the disabled mode, all receiver outputs are placed in a high impedance state. The shutdown function is intended to shut the device down, thereby minimizing the quiescent current. In shutdown, all transmitters are disabled and all receivers on the ADM211E are three-stated. On the ADM213E, receivers R4 and R5 remain enabled in shutdown. Note that the transmitters are disabled but are not three-stated in shutdown, so it is not permitted to connect multiple (RS-232) driver outputs together.

The shutdown feature is very useful in battery-operated systems since it reduces the power consumption to 1  $\mu$ W. During shutdown the charge pump is also disabled. The shutdown control input is active high on the ADM211E, and it is active low on the ADM213E. When exiting shutdown, the charge pump is restarted and it takes approximately 100  $\mu$ s for it to reach its steady state operating condition.

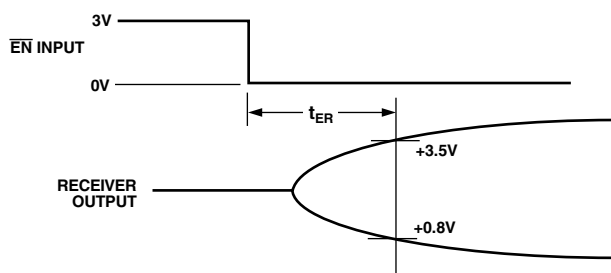
## High Baud Rate

The ADM2xxE feature high slew rates permitting data transmission at rates well in excess of the EIA-232-E specifications. RS-232 levels are maintained at data rates up to 230 kb/s even under worst case loading conditions. This allows for high speed data links between two terminals, making it suitable for the new generation modem standards which require data rates of 200 kb/s. The slew rate is internally controlled to less than 30 V/ $\mu$ s to minimize EMI interference.



NOTE:  
EN IS THE COMPLEMENT OF  $\overline{\text{EN}}$  FOR THE ADM213E

Figure 13. Receiver Disable Timing



NOTE:  
EN IS THE COMPLEMENT OF  $\overline{\text{EN}}$  FOR THE ADM213E

Figure 14. Receiver Enable Timing

## ESD/EFT Transient Protection Scheme

The ADM2xxE use protective clamping structures on all inputs and outputs that clamp the voltage to a safe level and dissipates the energy present in ESD (electrostatic) and EFT (electrical fast transients) discharges. A simplified schematic of the protection structure is shown in Figures 15a and 15b. Each input and output contains two back-to-back high speed clamping diodes. During normal operation, with maximum RS-232 signal levels, the diodes have no effect as one or the other is reverse-biased, depending on the polarity of the signal. If, however, the voltage exceeds about  $\pm 50$  V, reverse breakdown occurs and the voltage is clamped at this level. The diodes are large p-n junctions designed to handle the instantaneous current surge which can exceed several amperes.

The transmitter outputs and receiver inputs have a similar protection structure. The receiver inputs can also dissipate some of the energy through the internal 5 k $\Omega$  resistor to GND as well as through the protection diodes.

The protection structure achieves ESD protection up to  $\pm 15$  kV and EFT protection up to  $\pm 2$  kV on all RS-232 I-O lines. The methods used to test the protection scheme are discussed later.

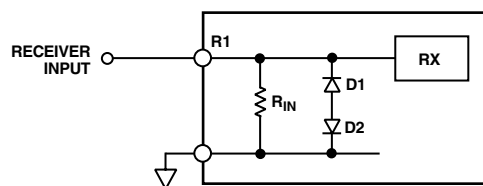


Figure 15a. Receiver Input Protection Scheme

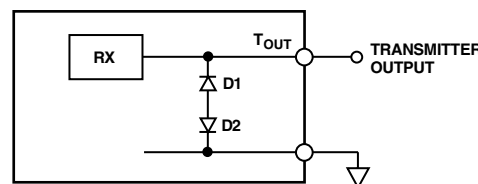


Figure 15b. Transmitter Output Protection Scheme

## ESD TESTING (IEC1000-4-2)

IEC1000-4-2 (previously 801-2) specifies compliance testing using two coupling methods, contact-discharge and air-gap discharge. Contact discharge calls for a direct connection to the unit being tested. Air-gap discharge uses a higher test voltage but does not make direct contact with the unit under test. With air discharge, the discharge gun is moved toward the unit under test, developing an arc across the air gap; hence the term air discharge. This method is influenced by humidity, temperature, barometric pressure, distance, and rate of closure of the discharge gun. The contact-discharge method, while less realistic, is more repeatable, and is gaining acceptance in preference to the air-gap method.

Although very little energy is contained within an ESD pulse, the extremely fast rise time, coupled with high voltages, can cause failures in unprotected semiconductors. Catastrophic destruction can occur immediately as a result of arcing or heating. Even if catastrophic failure does not occur immediately, the device may suffer from parametric degradation that may result in degraded performance. The cumulative effects of continuous exposure can eventually lead to complete failure.

I-O lines are particularly vulnerable to ESD damage. Simply touching or plugging in an I-O cable can result in a static discharge that can damage or destroy the interface product connected to the I-O port. Traditional ESD test methods such as the MIL-STD-883B method 3015.7 do not fully test a product's susceptibility to this type of discharge. This test was intended to test a product's susceptibility to ESD damage during handling. Each pin is tested with respect to all other pins. There are some important differences between the traditional test and the IEC test:

- The IEC test is much more stringent in terms of discharge energy. The peak current injected is over four times greater.
- The current rise time is significantly faster in the IEC test.
- The IEC test is carried out while power is applied to the device.

It is possible that the ESD discharge could induce latch-up in the device under test. This test, therefore, is more representative of a real-world I-O discharge where the equipment is operating normally with power applied. For maximum peace of mind, however, both tests should be performed, thus ensuring maximum protection both during handling and later during field service.



# ADM206E/ADM207E/ADM208E/ADM211E/ADM213E

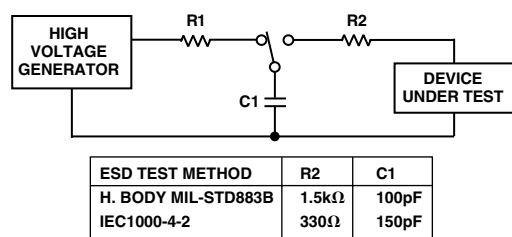


Figure 16. ESD Test Standards

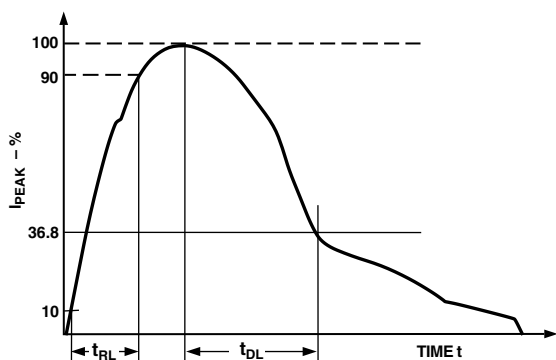


Figure 17. Human Body Model ESD Current Waveform

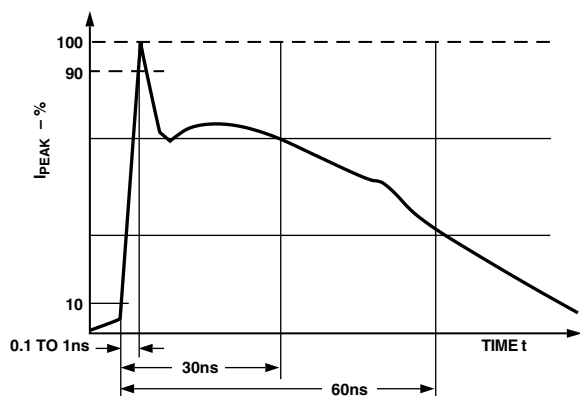


Figure 18. IEC1000-4-2 ESD Current Waveform

ADM2xxE products are tested using both of the above mentioned test methods. All pins are tested with respect to all other pins as per the MIL-STD-883B specification. In addition, all I-O pins are tested per the IEC test specification. The products are tested under the following conditions:

- (a) Power-On—Normal Operation
- (b) Power-On—Shutdown Mode
- (c) Power-Off

There are four levels of compliance defined by IEC1000-4-2. ADM2xxE products meet the most stringent compliance level for both contact and for air-gap discharge. This means that the products are able to withstand contact discharges in excess of 8 kV and air-gap discharges in excess of 15 kV.

Table IV. IEC1000-4-2 Compliance Levels

Level	Contact Discharge (kV)	Air Discharge (kV)
1	2	2
2	4	4
3	6	8
4	8	15

Table V. ADM2xxE ESD Test Results

ESD Test Method	I-O Pin (kV)
MIL-STD-883B	$\pm 15$
IEC1000-4-2	
Contact	$\pm 8$
Air	$\pm 15$

## FAST TRANSIENT BURST TESTING (IEC1000-4-4)

IEC1000-4-4 (previously 801-4) covers electrical fast transient burst (EFT) immunity. Electrical fast transients occur as a result of arcing contacts in switches and relays. The tests simulate the interference generated when, for example, a power relay disconnects an inductive load. A spark is generated due to the well known back EMF effect. In fact, the spark consists of a burst of sparks as the relay contacts separate. The voltage appearing on the line, therefore, consists of a burst of extremely fast transient impulses. A similar effect occurs when switching on fluorescent lights.

The fast transient burst test defined in IEC1000-4-4 simulates this arcing, and its waveform is illustrated in Figure 19. It consists of a burst of 2.5 kHz to 5 kHz transients repeating at 300 ms intervals. It is specified for both power and data lines.

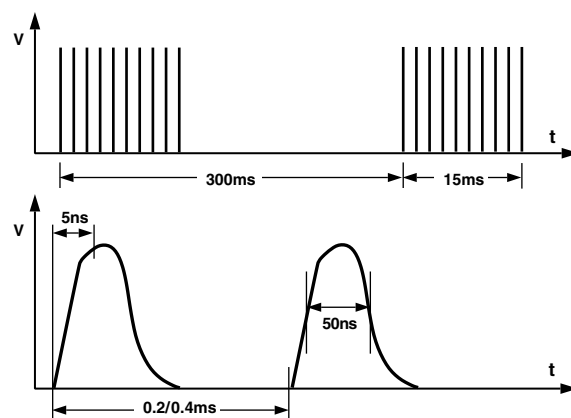


Figure 19. IEC1000-4-4 Fast Transient Waveform



**Table VI.**

Level	V Peak (kV) PSU	V Peak (kV) I-O
1	0.5	0.25
2	1	0.5
3	2	1
4	4	2

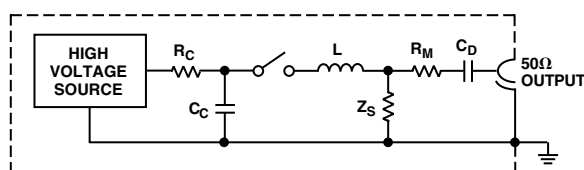
A simplified circuit diagram of the actual EFT generator is illustrated in Figure 20.

The transients are coupled onto the signal lines using an EFT coupling clamp. The clamp is 1 m long and it completely surrounds the cable providing maximum coupling capacitance (50 pF to 200 pF typ) between the clamp and the cable. High energy transients are capacitively coupled onto the signal lines. Fast rise times (5 ns) as specified by the standard result in very effective coupling. This test is very severe since high voltages are coupled onto the signal lines. The repetitive transients can often cause problems where single pulses do not. Destructive latch-up may be induced due to the high energy content of the transients. Note that this stress is applied while the interface products are powered up and are transmitting data. The EFT test applies hundreds of pulses with higher energy than ESD. Worst-case transient current on an I-O line can be as high as 40 A.

Test results are classified according to the following:

1. Normal performance within specification limits
2. Temporary degradation or loss of performance that is self-recoverable
3. Temporary degradation or loss of function or performance that requires operator intervention or system reset
4. Degradation or loss of function that is not recoverable due to damage

ADM2xxE products have been tested under worst-case conditions using unshielded cables, and meet Classification 2. Data transmission during the transient condition is corrupted, but it may be resumed immediately following the EFT event without user intervention.



*Figure 20. IEC1000-4-4 Fast Transient Generator*

## IEC1000-4-3 RADIATED IMMUNITY

IEC1000-4-3 (previously IEC801-3) describes the measurement method and defines the levels of immunity to radiated electromagnetic fields. It was originally intended to simulate the electromagnetic fields generated by portable radio transceivers or any other device that generates continuous wave radiated electromagnetic energy. Its scope has since been broadened to include spurious EM energy which can be radiated from fluorescent lights, thyristor drives, inductive loads, etc.

Testing for immunity involves irradiating the device with an EM field. There are various methods of achieving this, including use of anechoic chamber, stripline cell, TEM cell, GTEM cell. A stripline cell consists of two parallel plates with an electric field developed between them. The device under test is placed within the cell and exposed to the electric field. There are three severity levels having field strengths ranging from 1 V to 10 V/m. Results are classified in a similar fashion to those for IEC1000-4-4.

1. Normal operation
2. Temporary degradation or loss of function that is self-recoverable when the interfering signal is removed
3. Temporary degradation or loss of function that requires operator intervention or system reset when the interfering signal is removed
4. Degradation or loss of function that is not recoverable due to damage

The ADM2xxE family of products easily meets Classification 1 at the most stringent (Level 3) requirement. In fact, field strengths up to 30 V/m showed no performance degradation, and error-free data transmission continued even during irradiation.

**Table VII. Test Severity Levels (IEC1000-4-3)**

Level	Field Strength V/m
1	1
2	3
3	10

## EMISSIONS/INTERFERENCE

EN55 022, CISPR22 defines the permitted limits of radiated and conducted interference from information technology (IT) equipment. The objective of the standard is to minimize the level of emissions both conducted and radiated.

For ease of measurement and analysis, conducted emissions are assumed to predominate below 30 MHz and radiated emissions are assumed to predominate above 30 MHz.

## CONDUCTED EMISSIONS

This is a measure of noise that is conducted onto the line power supply. Switching transients from the charge pump that are 20 V in magnitude and containing significant energy can lead to conducted emissions. Other sources of conducted emissions can be due to overlap in switch on times in the charge pump voltage converter. In the voltage doubler shown below, if S2 has not fully turned off before S4 turns on, this results in a transient current glitch between V<sub>CC</sub> and GND which results in conducted emissions. It is therefore important that the switches in the charge pump guarantee break-before-make switching under all conditions so that instantaneous short-circuit conditions do not occur.

The ADM2xxE have been designed to minimize the switching transients and ensure break-before-make switching thereby minimizing conducted emissions. This has resulted in the level of emissions being well below the limits required by the specification. No additional filtering/decoupling other than the recommended 0.1 μF capacitor is required.

# ADM206E/ADM207E/ADM208E/ADM211E/ADM213E

Conducted emissions are measured by monitoring the line power supply. The equipment used consists of a LISN (line impedance stabilizing network) which essentially presents a fixed impedance at RF, and a spectrum analyzer. The spectrum analyzer scans for emissions up to 30 MHz. A plot for the ADM211E is shown in Figure 23.

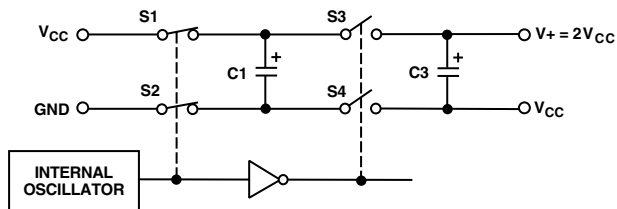


Figure 21. Charge Pump Voltage Doubler

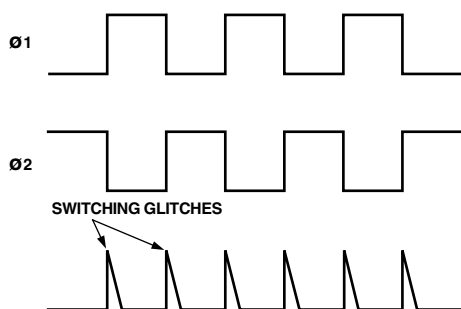


Figure 22. Switching Glitches

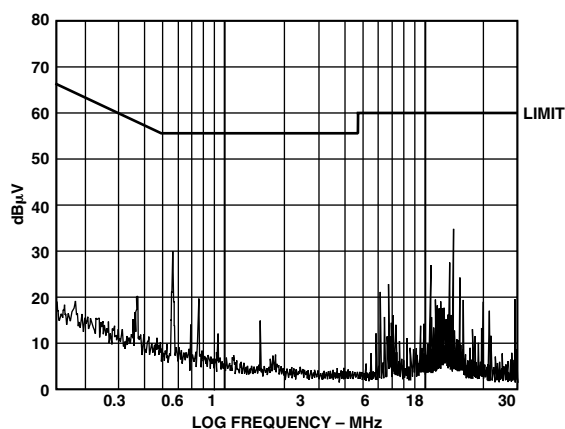


Figure 23. Conducted Emissions Plot

## RADIATED EMISSIONS

Radiated emissions are measured at frequencies in excess of 30 MHz. RS-232 outputs designed for operation at high baud rates while driving cables can radiate high frequency EM energy. The reasons already discussed which cause conducted emissions can also be responsible for radiated emissions. Fast RS-232 output transitions can radiate interference, especially when lightly loaded and driving unshielded cables. Charge pump devices are also prone to radiating noise due to the high frequency oscillator and high voltages being switched by the charge pump. The move towards smaller capacitors in order to conserve board space has resulted in higher frequency oscillators being employed in the charge pump design. This has resulted in higher levels of emission, both conducted and radiated.

The RS-232 outputs on the ADM2xxE products feature a controlled slew rate in order to minimize the level of radiated emissions, yet are fast enough to support data rates up to 230 kbaud.

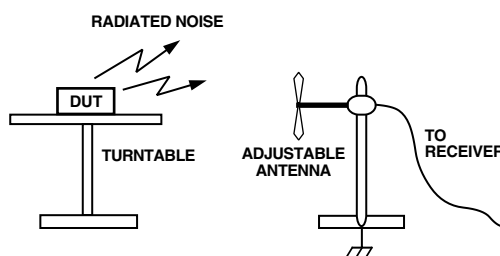


Figure 24. Radiated Emissions Test Setup

Figure 25 shows a plot of radiated emissions versus frequency. This shows that the levels of emissions are well within specifications without the need for any additional shielding or filtering components. The ADM2xxE were operated at maximum baud rates and configured in a typical RS-232 interface.

Testing for radiated emissions was carried out in a shielded anechoic chamber.

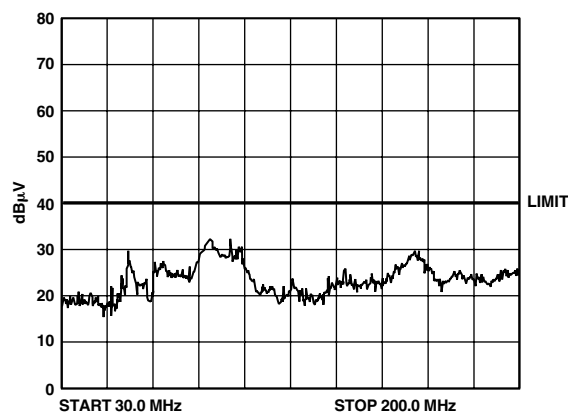


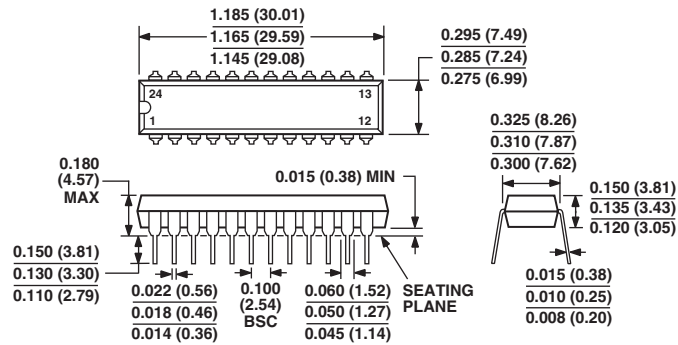
Figure 25. Radiated Emissions Plot

# ADM206E/ADM207E/ADM208E/ADM211E/ADM213E

## OUTLINE DIMENSIONS

### 24-Lead Plastic Dual In-Line Package [PDIP] (N-24)

Dimensions shown in inches and (millimeters)

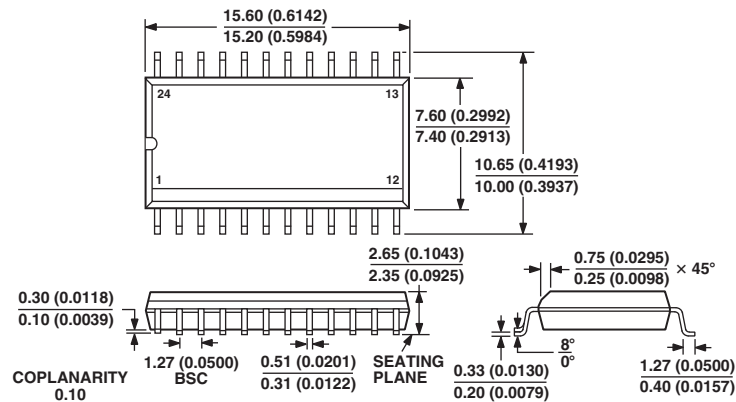


COMPLIANT TO JEDEC STANDARDS MO-095AG

CONTROLLING DIMENSIONS ARE IN INCHES; MILLIMETER DIMENSIONS (IN PARENTHESES) ARE ROUNDED-OFF INCH EQUIVALENTS FOR REFERENCE ONLY AND ARE NOT APPROPRIATE FOR USE IN DESIGN

### 24-Lead Standard Small Outline Package [SOIC] Wide Body (R-24)

Dimensions shown in millimeters and (inches)



COMPLIANT TO JEDEC STANDARDS MS-013AD

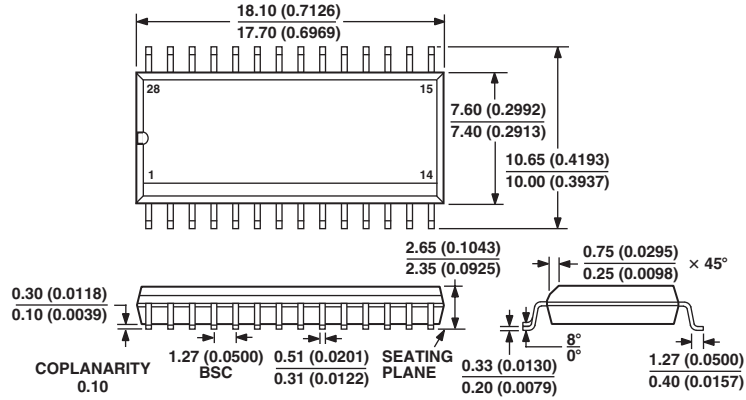
CONTROLLING DIMENSIONS ARE IN MILLIMETERS; INCH DIMENSIONS (IN PARENTHESES) ARE ROUNDED-OFF MILLIMETER EQUIVALENTS FOR REFERENCE ONLY AND ARE NOT APPROPRIATE FOR USE IN DESIGN

# ADM206E/ADM207E/ADM208E/ADM211E/ADM213E

## OUTLINE DIMENSIONS

### 28-Lead Standard Small Outline Package [SOIC] Wide Body (R-28)

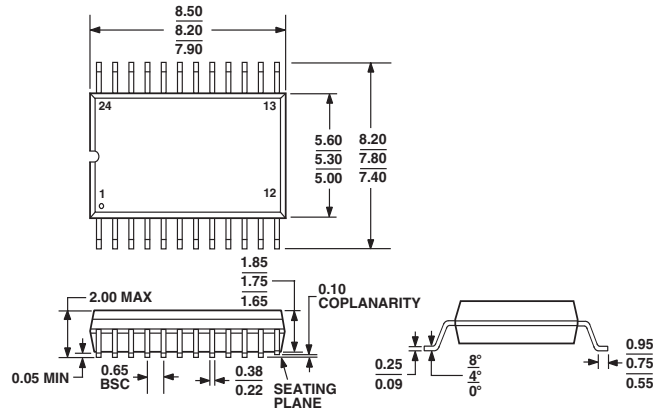
Dimensions shown in millimeters and (inches)



COMPLIANT TO JEDEC STANDARDS MS-013AE  
CONTROLLING DIMENSIONS ARE IN MILLIMETERS; INCH DIMENSIONS  
(IN PARENTHESES) ARE ROUNDED-OFF MILLIMETER EQUIVALENTS FOR  
REFERENCE ONLY AND ARE NOT APPROPRIATE FOR USE IN DESIGN

### 24-Lead Shrink Small Outline Package [SSOP] (RS-24)

Dimensions shown in millimeters



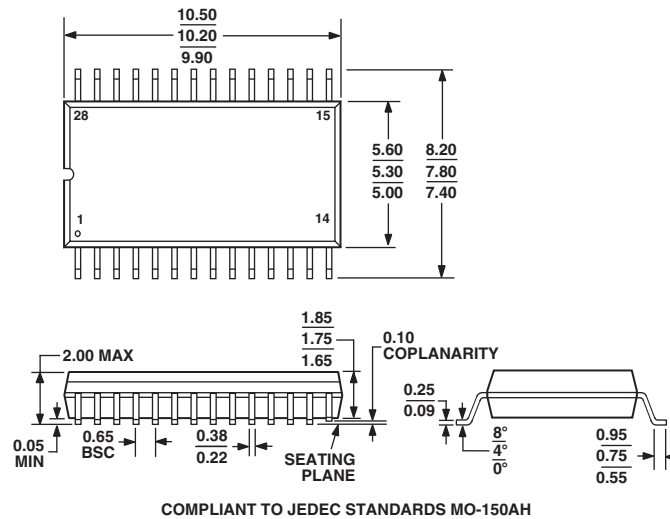
COMPLIANT TO JEDEC STANDARDS MO-150AG

# ADM206E/ADM207E/ADM208E/ADM211E/ADM213E

## OUTLINE DIMENSIONS

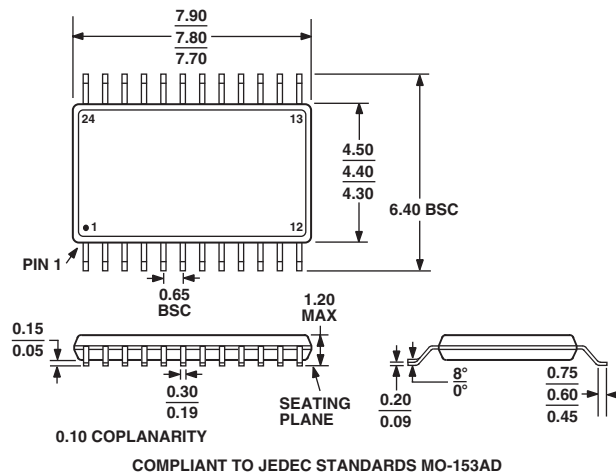
### 28-Lead Shrink Small Outline Package [SSOP] (RS-28)

Dimensions shown in millimeters



### 24-Lead Thin Shrink Small Outline Package [TSSOP] (RU-24)

Dimensions shown in millimeters

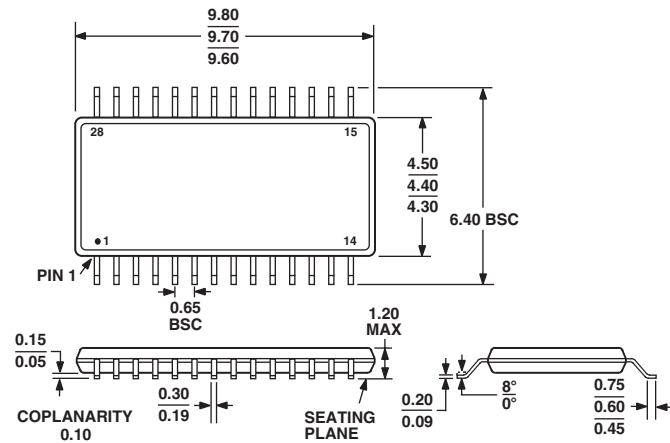


ADM206E/ADM207E/ADM208E/ADM211E/ADM213E

OUTLINE DIMENSIONS

28-Lead Thin Shrink Small Outline Package [TSSOP]  
(RU-28)

Dimensions shown in millimeters



COMPLIANT TO JEDEC STANDARDS MO-153AE

Revision History

Location	Page
4/05—Data Sheet changed from REV. C to REV. D.	
Changes to SPECIFICATIONS	2
Changes to ORDERING GUIDE	4
Updated OUTLINE DIMENSIONS	16
3/01—Data Sheet changed from REV. B to REV. C.	
Features	
Change 460 kbits/s to 230 kbits/s	1
Specifications Table	
Changed in Min, Typ, Max, Test Conditions/Comments columns	2
Absolute Maximum Ratings	
Deleted some items	3
Figures	
Change made in Figure 6	5
Typical Performance Characteristics	
Changes made in plots	7, 8
Table V.	
Column removed	11



