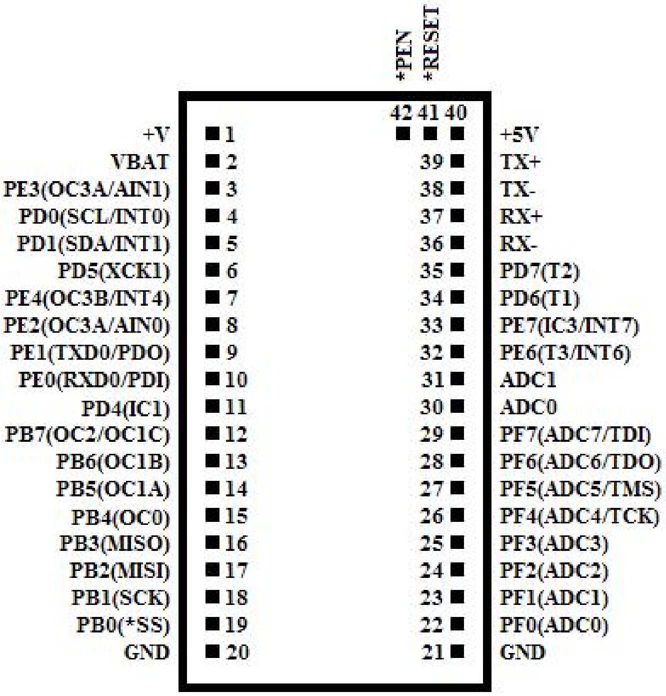
Micromint Modules

Micro64
AVR® Based Controller
■ Pin 1 www.micromint.com

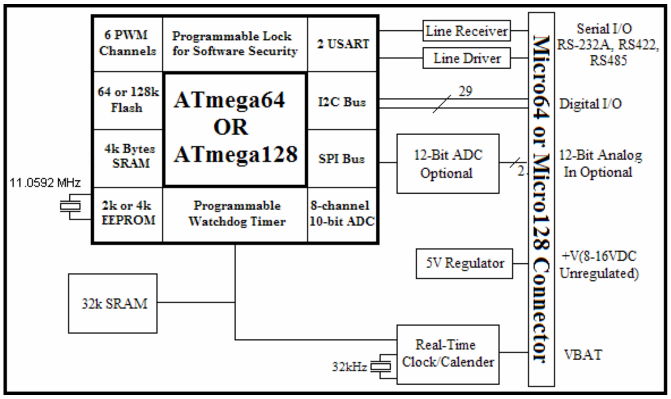
Microcomputer/Controller Featuring the ATmega64 or the ATmega128

FEATURES

- Small size – complete computer/controller with I/O less than 1.5 cubic inches (1.5” x 2.1” x 0.52)
- Low power only 275 mW typical
- Dual powered – operates on +5V or 6.5-20V at 55 mA (typical)
- Program and Data Memories
  - 64k or 128k Bytes of In-System Reprogrammable Flash with 10,000 Write/Erase Cycles
  - In-System Programming by On-chip Bootloader
  - 2 or 4 K Bytes EEPROM with 100,000 Write/Erase Cycles
  - 36K Bytes SRAM
  - Programming lock for Software Security
  - SPI interface for In-System Programming



- Peripheral Features
  - Real-Time Clock Calendar
  - Optional 2-channel 12-bit ADC
  - 8-channel 10-bit ADC
    - 8 Single-ended Channels
    - 7 Differential Channels
    - 2 Differential Channels with Programmable Gain (1x, 10x, 200x)
  - Byte-oriented Two-wire Serial Interface
  - Dual Programmable Serial USARTs
    - 1 TTL
    - 1 RS-232A, RS-422, or RS485
  - Master/Slave SPI Serial Interface
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  - Two Expanded 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Two 8-bit PWM Channels
  - 6 PWM Channels with Programmable Resolution from 1 to 16 Bits
  - Programmable Watchdog Timer with On-chip Oscillator
  - On Chip Analog Comparator
  - 29 Digital I/O that can sink or source 20mA



ABSOLUTE MAXIMUM RATINGS

Operating Temperature:	
Commercial	0°C to +70°C
Industrial	-40°C to +85°C
Storage Temperature	-50°C to +125°C
Voltage on +V (Pin 1) referenced to GND	0 to +16 VDC Unregulated

Voltage on +5V (Pin 40) referenced to GND With pin 1 open	0 to +5.5 VDC Regulated
Voltage on Vbat (pin 2) referenced to GND	0 to +5.5 VDC Regulated

Industrial temperature version is available; minimum quantities apply.

PIN DESCRIPTIONS

Micro64/128 is a 40-pin package (2.25” x 1.4” x 0.5”) with 0.1” pin and 1.2” row spacing. Some pins have multiple functions depending on system configuration. **DIO – Digital Input/Output**

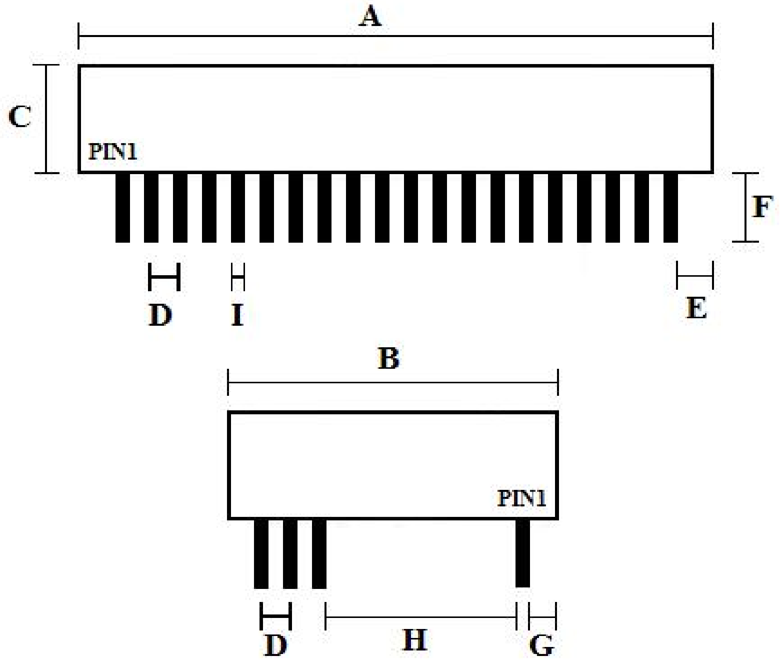
Pin	Signal	Description
1	+V	Micro64/128power supply input. +V is nominally 8-16 VDC. If pin 1 is open, Micro64/128 can be powered with +5 VDC directly on pin 40.
2	Vbat	2.5 VDC to 5.5 VDC Battery backup input for the optional Real-Time Clock Calendar.
3	PE3	DIO/AIN1/OC3A (Analog Comparator Negative Input or Output Compare and PWM Output A for Timer/Counter3)
4	PD0	DIO/INT0/SCL (External Interrupt0 Input or I^{2}C clock) Optionally used as the 12-bit ADC Chip Select .
5	PD1	DIO/INT1/SDA (External Interrupt1 Input or I^{2}C data) Optionally used as the 12-bit ADC Data I/O .
6	PD5	DIO/XCK1 (USART1 External Clock Input/Output) Optionally used as the 12-bit ADC Clock .
7	PE4	DIO/INT4/OC3B (External Interrupt4 Input or Output Compare and PWM Output for Timer/Counter 3)
8	PE2	DIO/AIN0/XCK0 (Analog Comparator Positive input or USART0 External Clock Input/Output)
9	PE1	DIO/PD0/TXD0 (Programming Data Output or UART0 Transmit Pin)
10	PE0	DIO/PDI/RXD0 (Programming Data Input or UART0 Receive Pin)
11	PD4	DIO/IC1 (Time/Counter1 Input Capture Trigger)
12	PB7	DIO/OC2/OC1C (Output Compare and PWM Output for Timer/Counter2 or Output Compare and PWM Output C for Timer/Counter1)
13	PB6	DIO/OC1B (Output Compare and PWM Output B for Timer/Counter1)
14	PB5	DIO/OC1B (Output Compare and PWM Output A for Timer/Counter1)
15	PB4	DIO/OC0 (Output Compare and PWM Output for Timer/Counter0)
16	PB3	DIO/MISO (SPI Bus Master Input/Slave Output)
17	PB2	DIO/MOSI (SPI Bus Master Output/Slave Input)
18	PB1	DIO/SCK (SPI Bus Serial Clock)

Pin	Signal	Description
19	PB0	DIO/*SS (SPI Bus Slave Select)
20	GND	Signal Point Analog and Digital Ground
21	GND	Signal Point Analog and Digital Ground
22	PF0	DIO/ADC0 (10-Bit ADC Input Channel 0)
23	PF1	DIO/ADC1 (10-Bit ADC Input Channel 1)
24	PF2	DIO/ADC2 (10-Bit ADC Input Channel 2)
25	PF3	DIO/ADC3 (10-Bit ADC Input Channel 3)
26	PF4	DIO/ADC4/TCK(10-Bit ADC Input Channel 4 or JTAG Test Clock)
27	PF5	DIO/ADC5/TMS(10-Bit ADC Input Channel 5 or JTAG Test Mode Select)
28	PF6	DIO/ADC6/TDO (10-Bit ADC Input Channel 6 or JTAG Test Data Output)
29	PF7	DIO/ADC7/TDI (10-Bit ADC Input Channel 7 or JTAG Test Data Input)
30	ADC0	12-bit ADC Channel 0 Input, Input Range 0 to +5VDC
31	ADC1	12-bit ADC Channel 1 Input, Input Range 0 to +5VDC
32	PE6	DIO/INT6/T3 (External Interrupt 6 Input or Timer/Counter3 Clock Input)

Pin	Signal	Description
33	PE7	DIO/INT7/IC3 (External Interrupt 7 Input or Timer/Counter3 Input Capture Trigger)
34	PD6	DIO/T1 (Timer/Counter1 Clock Input) Serial Transmitter Disable Control
35	PD7	DIO/T2 (Timer/Counter2 Clock Input)
36	RX-	RS-422/-485/-232A Inverted Serial (Receive Pair/Recxmit Pair/Receive)
37	RX+	RS-422/-485/-232A Noninverted Serial (Receive Pair/Rec-Xmit Pair)
38	TX-	RS-422/-485/-232A Inverted Serial (Transmit Pair/Rec-Xmit Pair/Transmit)
39	TX+	RS-422/-485/-232A Noninverted Serial (Transmit Pair/Rec-Xmit Pair)
36	RX-	RS-422/-485/-232A Inverted Serial (Receive Pair/Recxmit Pair/Receive)
40	+5V	This is the internal reference voltage for the 10 and 12 bit ADC. This output may be used to power minimal external circuitry or sensors. Micro64/128 may be powered on +5V only through this pin, provided Pin 1 is left unconnected.
41	RESET	When Brought to a logic low, RESET provides a master clear of the module
42	*PEN	Programming enable pin for the SPI Serial Programming mode. By holding this pin low during a Power-on Reset, the device will enter the SPI Serial Programming mode. PEN has no function during normal operation.

MECHANICAL AND ENVIRONMENTAL CHARACTERISTICS

Length	2.25 inches
Width	1.375 inches
Height	0.52 inches
Weight	45 grams
Operating Temperature	0°C to +70°C (Optional -40°C to +85°C)
Humidity	0 to 100% (noncondensing)



	Inches		Millimeters	
DIM	min	max	min	max
A	2.240	2.260	56.896	57.404
B	1.365	1.385	34.671	35.179
C	0.510	0.530	12.954	13.462
D	0.095	0.105	2.413	2.667
E	0.153	0.193	3.886	4.902
F	0.220	0.280	5.588	7.112
G	0.170	0.200	4.318	5.080
H	0.990	1.100	25.146	27.940
I	0.230	0.270	0.508	0.686

DC ELECTRICAL CHARACTERISTICS

Operating Temperature	Ta = 0^{o}C to + 70^{o}C
Operating Voltage	Vcc = 4.75 V to 5.5V
	Vss = 0.0 V

Characteristic	Minimum	Typical	Maximum	Units	Condition
Supply Voltage					
(V_{CC} to Pin 40)	4.75	5	5.5	V	
(+V to Pin 1)	6.5	9	16	V	
(Vbat to Pin 2)	2.45	3	5.5	V	
Supply Current (Icc)		55	80	mA	
Input Low Voltage (V_{IL})	-0.5		0.5	V	
Input High Voltage (V_{IH})	3.5		5.5	V	
Output Low Voltage (V_{OL})			0.7	V	I_{OL} = 20 mA, Vcc = 5V
Output High Voltage (V_{OH})	4.0			V	I_{OH} = -20 mA, Vcc = 5V

COMMUNICATION LINE DC ELECTRICAL CHARACTERISTICS

Characteristic	Minimum	Typical	Maximum	Units	Condition
Differential Driver Output Voltage			5	V	Unloaded See Note 1
RS-422	2		5	V	R=50 Ohms
RS-485	1.5		5	V	R=27 Ohms
Maximum Receiver Input voltage			+/-14	V	
ESD Protection		2000		V	

12 BIT A/D CONVERTER CHARACTERISTICS

Characteristic	Minimum	Typical	Maximum	Units	Condition
Resolution	12			bits	
Integral Nonlinearity Error		+/- 0.75	+/- 1.00	bits	
Offset Error		+/- 1.25	+/- 3	bits	
Gain Error		+/- 1.25	+/- 5	bits	
Voltage Reference	4.75	5	5.5	V	V_{REF} is V_{CC}
Analog Input Range	V_{SS}		Vcc	V	
Sample Rate			22,600	per second	

Note 1: RS-232A is characterized as a +/-5V bipolar signal (as opposed to RS-232C at +/-12 V). Drivers and receivers are actually RS-42 and the interface is an RS-423 connection (single ended to differential).

1.0 Micro64/128 Hardware Overview

Micro64/128 is an industrial oriented controller in a 2.25” by 1.375” 40-pin DIP encapsulated package. It is a combination of an Atmel AVR technology RISC microcontroller that includes on chip Flash, EEPROM, SRAM, and other features, an I^{2}C Real-Time Clock/Calendar , and 32k of additional SRAM. Optionally a 2-channel 12-bit ADC can be added. **Custom hardware configurations are available upon request.**

1.1 Atmega64 Microcontroller

The Atmega64 has the following features available to the Micro64 user: 64K bytes of In-System Programmable Flash with Read-While-Write capabilities, 2K bytes EEPROM, 4K bytes internal SRAM, 32k bytes external SRAM, 29 general purpose I/O lines, 32 general purpose working registers, four flexible Timer/Counters with compare modes and PWM, two 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, 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 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 continues to run.

1.2 Atmega128 Microcontroller

The ATmega128 has the following features available to the Micro128 user: 128K bytes of In-System Programmable Flash with Read-While-Write capabilities, 4K bytes EEPROM, 4K bytes internal SRAM , 32k bytes external SRAM, 29 general purpose I/O lines, 32 general purpose working registers, 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, 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 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.

1.3 M41T80 I^{2}C Real Time Clock

The M41T80 I^{2}C real time clock/calendar has a low operating current of 200µA. Eight registers are used for the clock/calendar function and are configured in binary coded decimal (BCD) format. An additional 5 registers provide status/control of an Alarm. Addresses and data are transferred serially via a two line, bi-directional I^{2}C interface. The built-in address register is incremented automatically after each WRITE or READ data byte.

Functions available to the user include a time-of-day clock/calendar and Alarm interrupts. The eight clock address locations contain the century, year, month, date, day, hour, minute, second and tenths/hundredths of a second in 24 hour BCD format. Corrections for 28, 29 (leap year - valid until year 2100), 30 and 31-day months are made automatically. The Alarm interrupt's output pin is connected PE5 (INT5) of the Atmega64 or Atmega128. The I^{2}C clock is connected to PD0 and the data line is connected to PD1.

1.4 Optional MCP3202 12-bit ADC

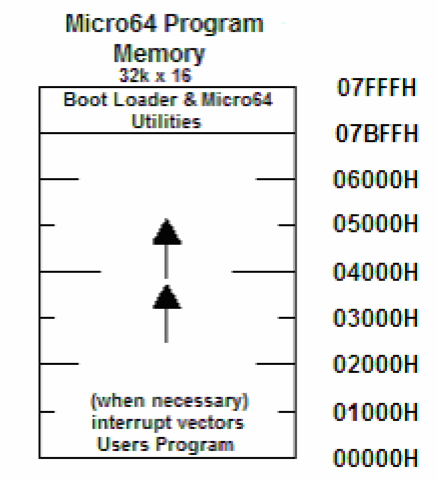
The MCP3202 is a successive approximation 12-bit Analog to Digital Converter with on board sample and hold circuitry. It is programmable to provide a single pseudo-differential input pair or dual single-ended inputs. Communication to the ADC is done using a bit banged SPI bus. The ADC's chip select is connected to PD0, the data in and data out are connected to PD1, and the clock signal is connected to PD5.

2.0 Memory Map

Micro64 and Micro128 memory is broken up into three different sections. Flash for program space, SRAM for volatile data storage, and EEPROM for nonvolatile data storage.

2.1 Micro64 Program Space

Micro64 has a total of 64k of program space. The upper 2k contains the bootloader and Micro64 Utilities. This leaves 62k available for the end user.



2.2 Micro128 Program Space

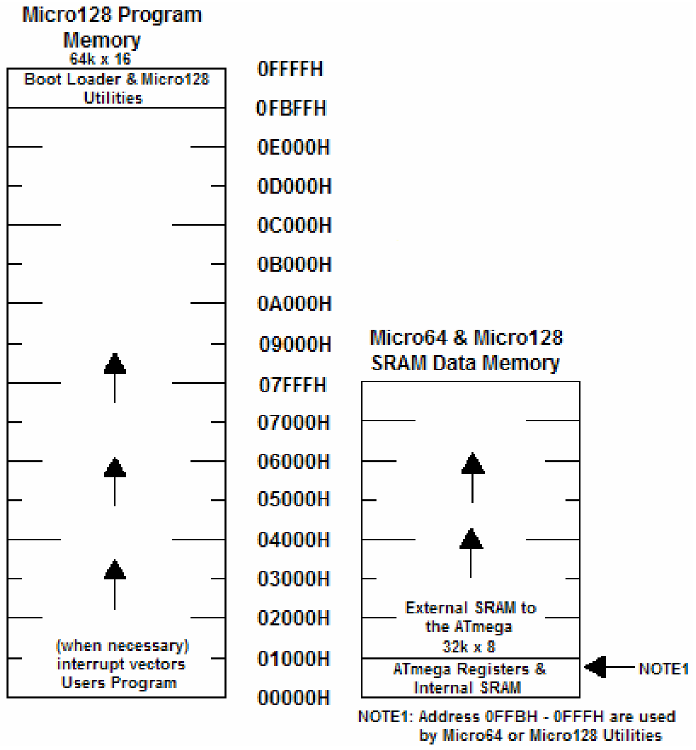
Micro64 has a total of 128k of program space. The upper 2k contains the bootloader and Micro128 Utilities. This leaves 126k available for the end user.

2.3 Micro64 & Micro128 SRAM Data Memory

Micro64 and Micro128 have 36k of SRAM for volatile data storage. The SRAM is broken up into three blocks. A two 4k blocks and a 28k block. One of the 4k blocks resides inside the ATmega64 or ATmega128 microcontroller. The 28k and the other 4 k block reside external to the ATmega64 or ATmega128. Micro64 and Micro128 Utilities uses 4 bytes in the 4k block. The bytes that the utilities use are located from address 0FFBH through 0FFFH.

2.4 Micro64 & Micro128 EEPROM Data Memory

Micro64 contains 2k bytes of EEPROM data space. Micro128 contains 4k bytes of EEPROM data space. The EEPROM has an endurance of at least 100,000 write/erase cycles. The access between the EEPROM and the CPU is done with three registers. The EEPROM Address Registers, the EEPROM Data Register, and the EEPROM Control Register.


Micro128 Program Memory
64k x 16
Boot Loader & Micro128 Utilities
(when necessary) interrupt vectors Users Program
0FFFFH
0FBFFH
0E000H
0D000H
0C000H
0B000H
0A000H
09000H
07FFFH
07000H
06000H
05000H
04000H
03000H
02000H
01000H
00000H
Micro64 & Micro128 SRAM Data Memory
External SRAM to the ATmega 32k x 8
ATmega Registers & Internal SRAM
NOTE1
NOTE1: Address 0FFBH - 0FFFH are used by Micro64 or Micro128 Utilities

3.0 Micro64 & Micro128 Utilities

Micromint has included utilities with it's built in boot loader. Not only do you have the power of a boot loader, but you also have extra functions to help make your application extremely easy to produce. These functions are loaded into the upper 4k of the program space along with the bootloader at the factory. The added functions include reading the 12-bit ADC, reading and writing the real-time clock colander registers, and using the I^{2}C Bus.

3.1 Micro64 Utilities Function Calls

SRAM Address					Functions	SRAM Address
0FFBH	**0FFCH**	**0FFDH**	**Feature**	**Function**	**Address**	**0FFEH - 0FFFH**
			12- bit ADC	Single-ended CH0	07C04H	Result
				Single-ended CH1	07BFDH	Result
				Differential +/- (CH0-CH1)	07C0BH	Result
				Differential -/+ (CH1-CH0)	07C12H	Result
			RTC	Read tenth of a second	07CF3H	Result/Error
				Read seconds	07CF7H	Result/Error
				Read minutes	07CFCH	Result/Error
				Read hours	07D00H	Result/Error

Micro64 Utilities Continued

SRAM Address					Functions	SRAM Address
0FFBH	**0FFCH**	**0FFDH**	**Feature**	**Function**	**Address**	**0FFEH - 0FFFH**
			RTC	Read the day of the week	07D05H	Result/Error
				Read the day of the month	07D09H	Result/Error
				Read the month	07D0DH	Result/Error
				Read the year	07D11H	Result/Error
		Data		Write seconds	07D15H	Result/Error
		Data		Write minutes	07D1DH	Result/Error
		Data		Write hours	07D21H	Result/Error
		Data		Write the day of the week	07D29H	Result/Error
		Data		Write the day of the month	07D2DH	Result/Error
		Data		Write the month	07D31H	Result/Error
		Data		Write the year	07D35H	Result/Error
				Read the alarm seconds	07D39H	Result/Error
				Read the alarm minutes	07D3EH	Result/Error
				Read the alarm hour	07D43H	Result/Error
				Read the alarm day of the month	07D48H	Result/Error
				Read the alarm month	07D4DH	Result/Error
		Data		Write the alarm seconds	07D58H	Result/Error
		Data		Write the alarm minutes	07D6BH	Result/Error
		Data		Write the alarm hour	07D7EH	Result/Error
		Data		Write the alarm day of the month	07D91H	Result/Error
		Data		Write the alarm month	07DA4H	Result/Error
				Enable the alarm	07DB9H	Result/Error
				Disable the alarm	07DC7H	Result/Error
				Set the alarm to repeat every second	07DD2H	Result/Error
				Set the alarm to repeat every minute	07DFBH	Result/Error
				Set the alarm to repeat every hour	07E24H	Result/Error
				Set the alarm to repeat every day	07E4DH	Result/Error
				Set the alarm to repeat every month	07E76H	Result/Error
				Set the alarm to repeat every year	07EA2H	Result/Error
				Read the alarm flags	07ECBH	Result/Error
			I^2C Byte Transfer	Initialize the I^2C at 400kHz	07C20H	
				Initialize the I^2C at 100kHz	07C23H	
Slave Address	Register			Read registered byte	07C4DH	Result/Error
Slave Address	Register	Data		Send register byte	07C8AH	Result/Error
Slave Address		Data		Send Byte	07CB8H	Result/Error
Slave Address				Read Byte	07CDDH	Result/Error
				Disable the I^2C	07C37H	
			Utilities Version Number		07C19H	Result

3.2 Micro128 Utilities Function Calls

SRAM Address					Functions	SRAM Address
0FFBH	0FFCH	0FFDH	Feature	Function	Address	0FFEH - 0FFFH
			12- bit ADC	Single-ended CH0	0FC09H	Result
				Single-ended CH1	0FC02H	Result
				Differential +/- (CH0-CH1)	0FC10H	Result
				Differential -/+ (CH1-CH0)	0FC17H	Result
			RTC	Read tenth of a second	0FCF8H	Result/Error
				Read seconds	0FCFCH	Result/Error
				Read minutes	0FD01H	Result/Error
				Read hours	0FD05H	Result/Error
				Read the day of the week	0FD0AH	Result/Error
				Read the day of the month	0FD0EH	Result/Error
				Read the month	0FD12H	Result/Error
				Read the year	0FD16H	Result/Error
		Data		Write seconds	0FD1AH	Result/Error
		Data		Write minutes	0FD22H	Result/Error
		Data		Write hours	0FD26H	Result/Error
		Data		Write the day of the week	0FD2EH	Result/Error
		Data		Write the day of the month	0FD32H	Result/Error
		Data		Write the month	0FD36H	Result/Error
		Data		Write the year	0FD3AH	Result/Error
				Read the alarm seconds	0FD3EH	Result/Error
				Read the alarm minutes	0FD43H	Result/Error
				Read the alarm hour	0FD48H	Result/Error
				Read the alarm day of the month	0FD4DH	Result/Error
				Read the alarm month	0FD52H	Result/Error
		Data		Write the alarm seconds	0FD5DH	Result/Error
		Data		Write the alarm minutes	0FD70H	Result/Error
		Data		Write the alarm hour	0FD83H	Result/Error
		Data		Write the alarm day of the month	0FD96H	Result/Error
		Data		Write the alarm month	0FDA9H	Result/Error
				Enable the alarm	0FDBEH	Result/Error
				Disable the alarm	0FDCCH	Result/Error
				Set the alarm to repeat every second	0FDD7H	Result/Error
				Set the alarm to repeat every minute	0FE00H	Result/Error
				Set the alarm to repeat every hour	0FE29H	Result/Error
				Set the alarm to repeat every day	0FE52H	Result/Error
				Set the alarm to repeat every month	0FE7BH	Result/Error
				Set the alarm to repeat every year	0FEA7H	Result/Error
				Read the alarm flags	0FED0H	Result/Error

Micro128 Utilities Continued

SRAM Address						SRAM Address
0FFBH	**0FFCH**	**0FFDH**	**Feature**	**Function**	**Functions Address**	**0FFEH - 0FFFH**
			I^2C Byte Transfer	Initialize the I^2C at 400kHz	0FC25H	
				Initialize the I^2C at 100kHz	0FC28H	
Slave Address	Register			Read registered byte	0FC52H	Result/Error
Slave Address	Register	Data		Send register byte	0FC8FH	Result/Error
Slave Address		Data		Send Byte	0FCBDH	Result/Error
Slave Address				Read Byte	0FCE2H	Result/Error
				Disable the I^2C	0FC3CH	
			Utilities Version Number		0FC1EH	Result

4.0 Micro64/128 Software

When it comes from the factory, the Micro64/128 has no software on the board itself. Programs are developed using cross-development tools running on a desktop PC and is programmed into the Micro64/128 for execution. There are several development environments from which to choose.

4.1 Assembly

Any cross assembler capable of creating programs for Atmel's Atmega64 or Atmega128 microcontroller can be used to write assembly language programs for the Micro64/128. Atmel's assembler is available for free by downloading it at www.atmel.com/products/avr/

Other cross assemblers may be found by looking under "Third Party Support" on the website above.

4.2 BASCOM-AVR BASIC Compiler

BASCOM-AVR has a complete Windows IDE (Integrated Development Environment) with a Terminal Emulator. It is a structured BASIC with labels, that have statements highly compatible with Microsoft's Quick BASIC and Visual BASIC. BASCOM AVR supports IF-THEN-ELSE-END IF, DO-LOOP, WHILE-WEND, SELECT- CASE and in line assembly. It has a large set of Trig Floating point functions. Variables and labels can be 32 characters. Bit, Byte, Integer, Word, Long, Single, and Strings are supported for variable types.

A bit is 1/8 of a byte and can only be a 1 or a 0. A byte is stored as an unsigned 8-bit binary number ranging in value from 0 to 255. An integer is stored as a signed sixteen-bit binary number ranging in value from -32,768 to +32,767. A word is stored as an unsigned sixteen-bit binary number ranging in value from 0 to 65535. A long is stored as signed 32-bit binary number ranging in value from -2147483648 to 2147483647. A single is stored as a signed 32 bit binary number. Ranging in value from 1.5 x 10^–45 to 3.4 x 10^38. A string is stored as bytes and is terminated with a 0-byte. A string dimensioned with a length of 10 bytes will occupy 11 bytes of memory. For more information please visit www.mcselec.com .

4.3 CodeVisionAVR C Compiler

CodeVisionAVR runs under Windows 95, 98, Me, NT 4.0, 2000 and XP. It is an easy to use integrated development Environment with a built-in serial communication terminal for debugging and has an editor with auto indentation and keywords highlighting. The C Compiler supports bit, char, int, short, long, and float data types.

It also has supplementary libraries for Alphanumeric LCD modules for up to 4x40 characters, Philips I²C Bus, National Semiconductor LM75 Temperature Sensor, Dallas DS1621 Thermometer/Thermostat, Philips PCF8563 and PCF8583 Real Time Clocks, Dallas DS1302 and DS1307 Real Time Clocks, Dallas 1 Wire protocol, Dallas DS1820/DS1822 1 Wire Temperature sensors, Dallas DS2430/DS2433 1 Wire EEPROMs, SPI, Power management, Delays, BCD and Gray code conversion.

The compiler comes with a built-in CodeWizardAVR Automatic Program Generator, that allows you to write in a matter of minutes all the code needed for implementing the following functions: External memory access setup, Chip reset source identification, Input/Output Port initialization, External Interrupts initialization, Timers/Counters initialization, Watchdog Timer initialization, UART initialization and interrupt driven buffered serial communication with the following parameters: 7N2, 7E1, 7O1, 8N1, 8N2, 8E1 and 8O1, Analog Comparator initialization, ADC initialization, SPI Interface initialization, I²C Bus, LM75 Temperature Sensor, DS1621 Thermometer/Thermostat, PCF8563, PCF8583, DS1302 and DS1307 Real Time Clocks initialization, 1 Wire Bus and DS1820/DS1822 Temperature Sensors initialization, and LCD module initialization. For further information please visit http://www.hpinfotech.ro .

5.0 Programming the Micro64/128

The user-programmable part of the Micro64/128 uses Atmel's Atmega64 or Atmega128 FLASH micro-controller that can be reprogrammed thousands of times. These programs can be created using a number of resources, as described above.

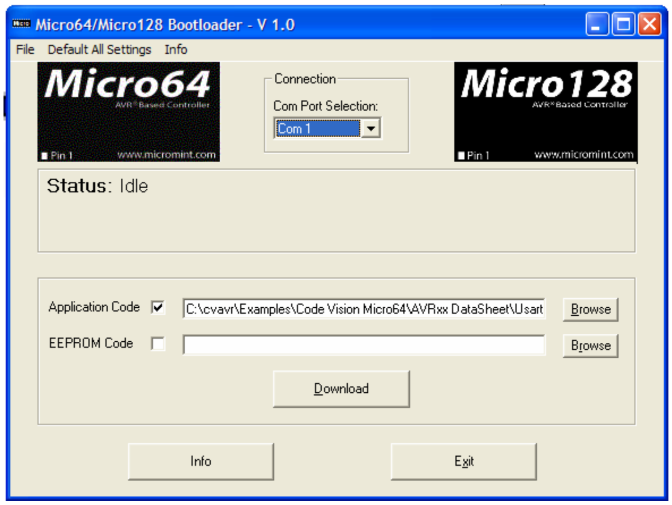
Programming the Micro64/128 is done through USART1, the RS232a/RS422/RS485 serial port, by using the Micro64/128 Boot Loader IDE. The Micro64/128 **cannot** be programmed over a RS485 network.

5.1 Using the Boot Loader IDE

The Boot Loader IDE comes free with the Micro64 and the Micro128. It is used to program a program and data EEPROM files into the module. It is very easy to use. There are only six steps to send a program to the Micro64/128.

<u>**Step**</u>

1. Open the Boot Loader IDE.
2. Select the appropriate COM Port.
3. Click on the top "Browse" button to find the program you want to send to the Micro64/128.
4. Optional Step: Click on the bottom "Browse" button to find the EEPROM file you want to send to the Micro64/128.
5. Click on the "Download" button to send the files.
6. Apply power to the board or reset the module.



Com Port Selection: Com 1

Status: Idle

Application Code ☑ C:\cvavr\Examples\Code Vision Micro64\AVRxx DataSheet\Usart

EEPROM Code □

5.2 Boot Loader Programming Protocol

After the Boot Loader is started (via a reset or a power-up), the following protocol must be observed:

1. 1. Upon power-up or reset Boot Loader sends a '^' (BOOTLOADER_ACTIVE_CHAR) at 115,200 bits per second (bps) using X-ON/X-OFF handshaking.

2. The host is then required to send the three-character entry sequence of '@&$'. This is used to prevent an inadvertent attempt of reprogramming from taking place. If the Boot Loader does not receive these characters within the timeout period of 5 seconds, the Boot Loader tests to see if there is code located in the main application area of flash. If there is, then the Boot Loader jumps to it, otherwise, execution stays within the Boot Loader indefinitely, waiting for the entry sequence.

3. Once the three-character entry sequence has been sent, the Boot Loader sends the version string (Vx.xx) followed by a '?' (READY_CHAR).

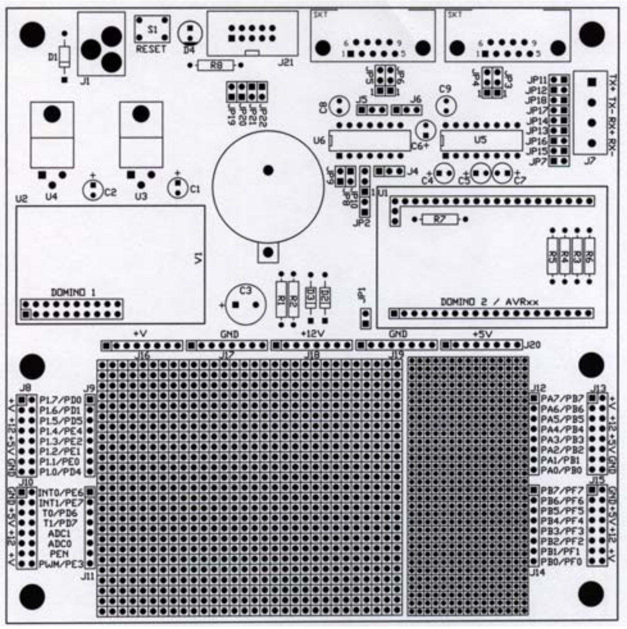
4. Upon receipt of the READY_CHAR, the host application should send the hex file for the new/updated application program observing an X-ON / X_OFF handshaking protocol to control data flow. The handshaking is very important as the flash memory area writes much more slowly than the serial port can send data. The programming software continues sending the hex file until it is all sent. After each line of ".hex" file is received by the Boot Loader, one of three characters is transmitted by the Boot Loader:
   - '~' Line received with no errors.
   - '%'Line received with no error, but an error occurred while flashing.
   - '-'Checksum error detected while receiving the line.

5. After the programming is complete, the Boot Loader sends either a '#', meaning the programming is all right, or and@' indicating that an error has occurred and the program did not load successfully. In most cases an error during programming means that the main application program is corrupted and will need to be resent.

6. The Micro64 Boot Loader then starts the newly programmed application software. As stated in step 2, the Micro64 Boot Loader tests to see if there is code located in the main application area of flash. If there is, the Micro64 Boot Loader jumps to it, otherwise, execution stays within the Micro64 Boot Loader indefinitely, waiting for the entry sequence

6.0 Micro64/128 Development Board

The Micro64/128 Development Board was designed, for use as an evaluation platform, for prototyping additional circuitry around applications using the Micro64/128.

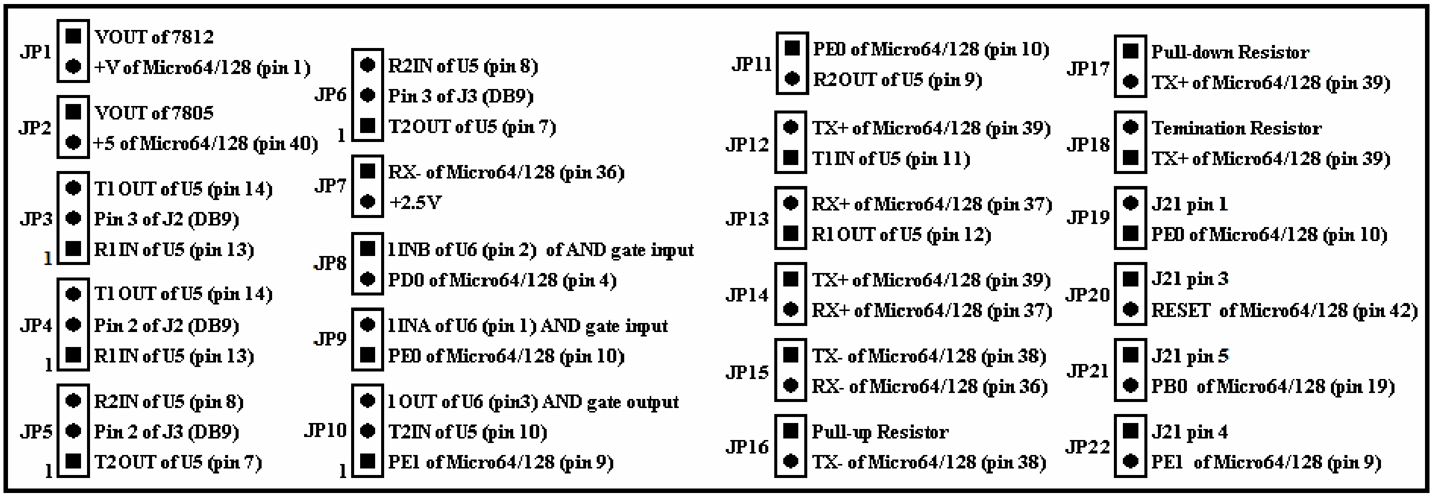


6.1 Development Board Power Supply

An unregulated 12 VDC wall transformer with a 2.5mm power plug is supplied to power the board. The plugs center tap should be negative. A diode (D1) will protect the regulator if a power supply with he wrong polarity is accidentally plugged in. The development board can power the Micro64/128 with a regulated +12VDC by placing a jumper on JP1 or with a regulated +5V by placing a jumper on JP2. **NOTE: Inserting jumpers on both JP1 and JP2 will damage the module.**

6.2 Jumper Descriptions

There are twenty-two jumpers on the development board. Figure 4-1 list each jumper and what they are connected to.



Jumper	Pin	Connection
JP1	■	VOUT of 7812
	●	+V of Micro64/128 (pin 1)
JP2	■	VOUT of 7805
	●	+5 of Micro64/128 (pin 40)
JP3	●	T1 OUT of U5 (pin 14)
	●	Pin 3 of J2 (DB9)
	1 ■	R1 IN of U5 (pin 13)
JP4	●	T1 OUT of U5 (pin 14)
	●	Pin 2 of J2 (DB9)
	1 ■	R1 IN of U5 (pin 13)
JP5	●	R2IN of U5 (pin 8)
	●	Pin 2 of J3 (DB9)
	1 ■	T2OUT of U5 (pin 7)
JP6	●	R2IN of U5 (pin 8)
	●	Pin 3 of J3 (DB9)
	1 ■	T2OUT of U5 (pin 7)
JP7	■	RX- of Micro64/128 (pin 36)
	●	+2.5V
JP8	■	1INB of U6 (pin 2) of AND gate input
	●	PD0 of Micro64/128 (pin 4)
JP9	●	1INA of U6 (pin 1) AND gate input
	■	PE0 of Micro64/128 (pin 10)
JP10	●	1 OUT of U6 (pin3) AND gate output
	●	T2IN of U5 (pin 10)
	1 ■	PE1 of Micro64/128 (pin 9)
JP11	■	PE0 of Micro64/128 (pin 10)
	●	R2OUT of U5 (pin 9)
JP12	●	TX+ of Micro64/128 (pin 39)
	■	T1IN of U5 (pin 11)
JP13	●	RX+ of Micro64/128 (pin 37)
	■	R1 OUT of U5 (pin 12)
JP14	■	TX+ of Micro64/128 (pin 39)
	●	RX+ of Micro64/128 (pin 37)
JP15	■	TX- of Micro64/128 (pin 38)
	●	RX- of Micro64/128 (pin 36)
JP16	■	Pull-up Resistor
	●	TX- of Micro64/128 (pin 38)
JP17	■	Pull-down Resistor
	●	TX+ of Micro64/128 (pin 39)
JP18	●	Temination Resistor
	■	TX+ of Micro64/128 (pin 39)
JP19	●	J21 pin 1
	■	PE0 of Micro64/128 (pin 10)
JP20	■	J21 pin 3
	●	RESET of Micro64/128 (pin 42)
JP21	■	J21 pin 5
	●	PB0 of Micro64/128 (pin 19)
JP22	■	J21 pin 4
	●	PE1 of Micro64/128 (pin 9)

Figure 4-1 ***Development Board Jumper Descriptions***

6.3 Serial Communication

The development board can be configured to have the Micro64/128's USART1 communicate in RS-232, RS-422, and RS-485. It also has the ability to configure USART0 to communicate in RS-232.

6.3.1 USART1 as RS-232.

In order to use USART1 for RS-232 communications the user must take into account the following jumpers JP3, JP4, JP7, JP12, and JP13. The RS-232 connection is made using a DB-9M and connecting it to the development boards, DB-9F, J2 connector. JP12 connects TX+ (pin 39 of Micro64/128) to the RS-232 driver chips T1IN signal (pin 11 of U5). JP13 connects RX+ (pin 37 of Micro64/128) to the RS-232 driver chips R1OUT signal (pin 12 of U5). Since the Micro64/128 uses a differential receiver, 2.5 VDC must be connected to RX- (pin 36 of Micro64/128). By placing a jumper on JP7, 2.5VDC is applied to RX-. JP3 is used to connect either the receiving or transmitting signal to pin 3 of J2's DB-9F. JP4 is used to connect either the receiving or transmitting signal to pin 2 of J2's DB-9F. If the user were going to communicate to a Personal Computer they would need to put jumpers on JP7, JP12, JP13, JP3 pins 1&2, and JP4 pins 2&3. Figure 4-2 illustrates what the jumpers might look like.

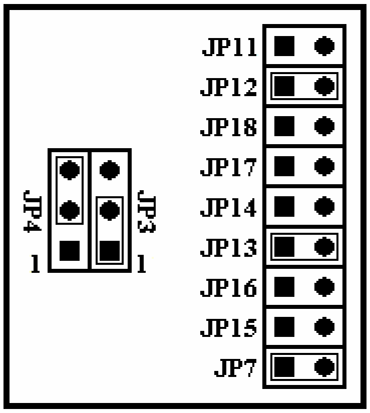


Figure 4-2 *USART1 Jumper Configurations for Connecting to a PC*

6.3.2 USART1 as RS-422

RS-422 communications require two twisted pairs. One pair connects the console transmitter to Micro64/128's receiver while the second pair connects the console receiver to Micro64/128's transmitter.

RS-422 uses two unidirectional data paths – one path for each direction. The data transmission is differential, enabling the noise picked-up on the pairs to cancel itself out. Each twisted pair should have termination enabled at each end of the line. Pull-up and pull-down termination may be required, but only at one end of each pair.

The only jumpers that are associated with RS-422 communications are JP16, JP17, and JP18 if termination is necessary. The twisted wire pairs should be connected to the screw terminals J7. J7 pin 1 is for TX+, pin 2 is for TX- , pin 3 is for RX+ and pin 4 is for RX-. JP16, JP17 and JP18 are the jumpers used for terminating the network. Since termination is needed to match the impedance of a node to the impedance of the transmission line the termination (R7), pull-up (R5) and pull-down (R6) resistors are not populated by the factory.

6.3.3 USART1 as RS-485

RS-485 communication requires one twisted pair to connect the console to Micro64/128. RS-485 uses one data path, so the drives at each end must NOT be enabled at the same time. The user is responsible for NOT breaking this rule.

The easiest protocol to follow is a master/slave(s) relationship, were the slaves DO NOT enable their transmitter (respond) unless the master asks them to. The data transmission is differential allowing picked up noise to cancel itself out. The twisted pair should have termination enabled at each end of the line. Pull-up and pull-down termination may be required, but only at one end of the pair.

The only jumpers that are associated with RS-485 communications are JP14, JP15, JP16, JP17, and JP18 if termination is necessary. JP14 is used to connect the positive side (TX+ & RX+) of the network together and JP15 is used to connect the negative (TX- & RX-) side together. The twisted wire pair should be connected to the screw terminal J7 pin 1 or 3 for the positive side and pin 2 or 4 for negative side. JP16, JP17 and JP18 are the jumpers used for terminating the network. Since termination is needed to match the impedance of a node to the impedance of the transmission line the termination (R7), pull-up (R5) and pull-down (R6) resistors are not populated by the factory.

6.3.4 USART0 as RS-232

In order to use USART0 for RS-232 communications the user must take into account the following jumpers JP5, JP6, JP10, and JP11. The RS-232 connection is made using a DB-9M and connecting it to the development boards, DB-9F, J3 connector. JP10 connects PE1 (USART0's transmit pin and pin 9 of Micro64/128) to the RS-232 driver chips T2IN signal (pin 10 of U5). JP13 connects PE0 (USART0's receive pin and pin 10 of Micro64/128) to the RS-232 driver chips R2OUT signal (pin 9 of U5). JP6 is used to connect either the receiving or transmitting signal to pin 3 of J3's DB-9F. JP5 is used to connect either the receiving or transmitting signal to pin 2 of J3's DB-9F. If the user were going to communicate to a Personal Computer they would need to put jumpers on JP7, JP12, JP13, JP3 pins 1&2, and JP4 pins 2&3. Figure 4-3 illustrates what the jumpers might look like.

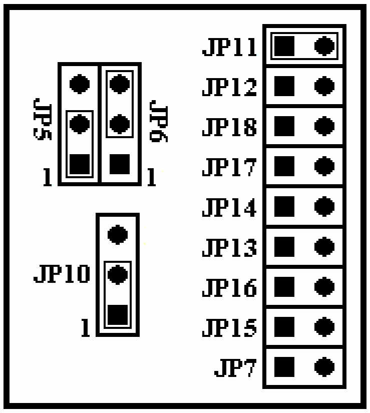


Figure 4-3 *USART0 Jumper Configurations for Connecting to a PC*

6.4 Micro64/128 Connections

All of the Micro64/128's signals except for VBAT, *RESET, TX+, TX-, RX+, and RX- are brought out to four 2x8 headers (J8, J10, J13, & J15) and four 1x8 plated through solder holes (J9, J11, J12, & J14). Please refer to figure 8 for their pin out. J16, J17, J18, J19 and J20 are connected to the development boards power supply and can be used to power your external circuitry and are located above the prototyping area. VBAT can be connected to either a coin cell battery or a Super Capacitor. If the user chooses to backup the real-time clock with a battery, a coin cell battery holder (V1) and a 0 ohm resistor (D2) would have to be installed on the board. The 0 ohm resistor instead of a diode would need to be installed because the Micro64/128 already has a diode built in. To back-up the real-time clock with a super capacitor, two 1k resistors need to be installed in R1 and R2, a BAT85 Diode needs to be installed in D3 and the 1 Farad Super Capacitor needs to be installed in C3. The RESET pin for Micro64/128 is connected to a pushbutton located near the power connector. The user can press and release the pushbutton to reset the Micro64/128.

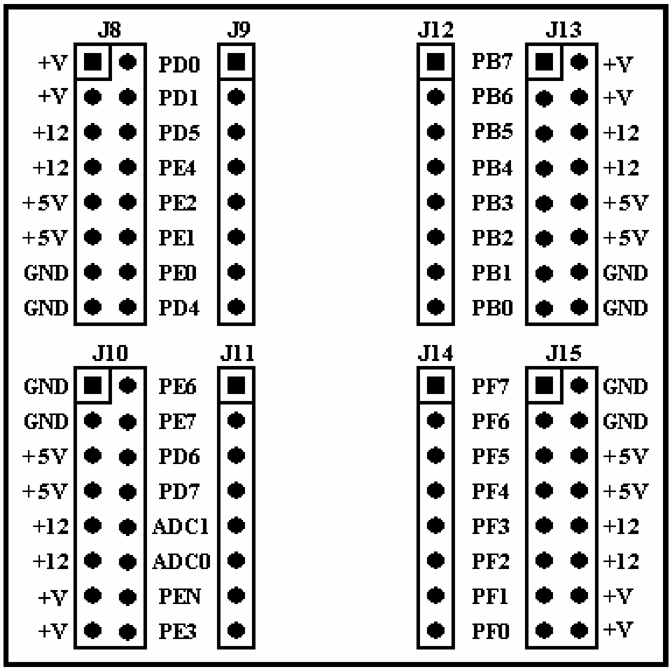
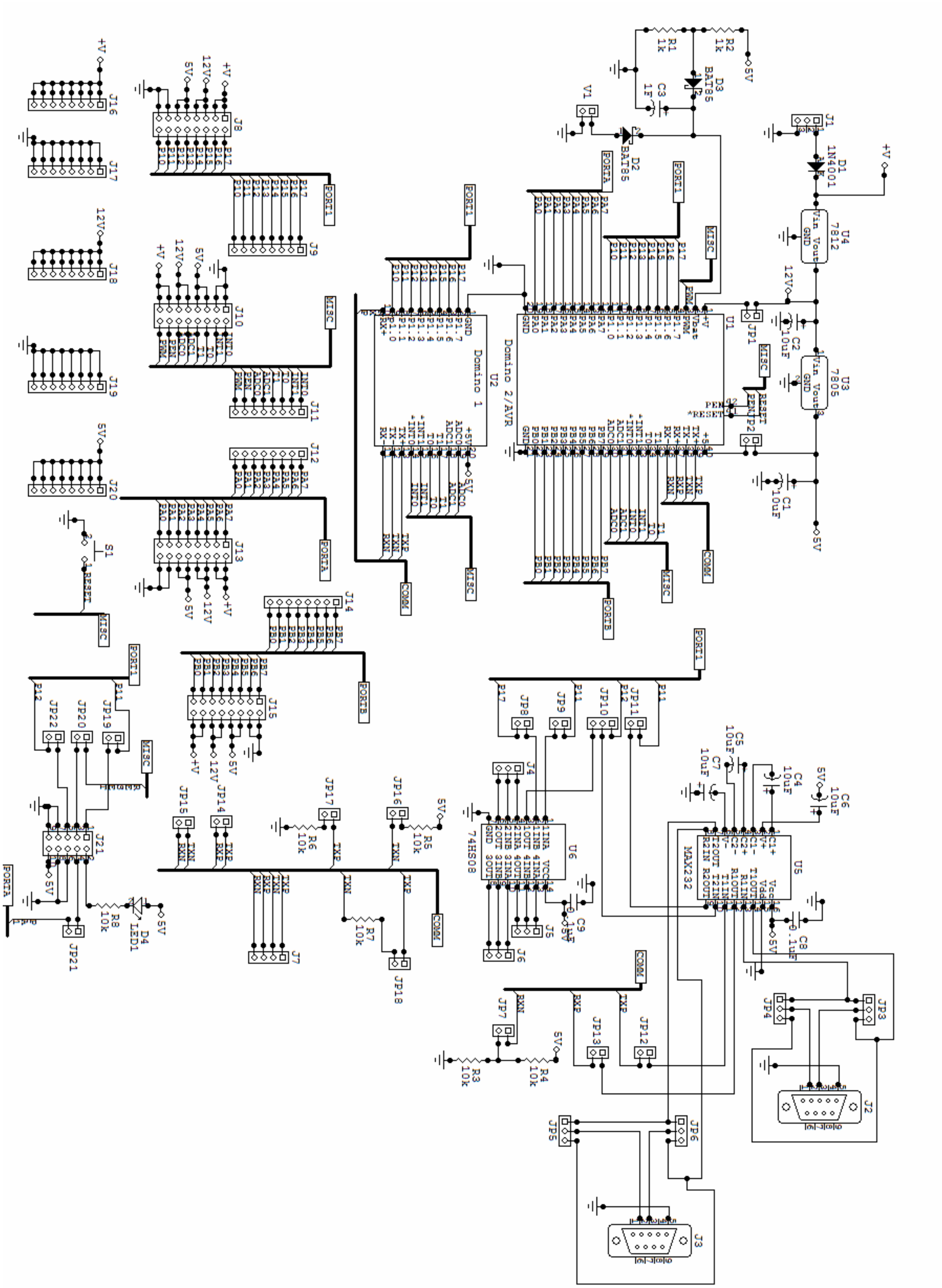


Figure 4-4 *Micro64/128 Connections*

6.5 Development Board Schematic



Appendix 1.0

1.1 Sample Application: Communications

Micro64/128 can communicate with other serial devices at up to 230.4 kbps. Usart1 can be connected in one of three configurations: RS-232A, RS422, and RS-485. Micro64/128’s RS-232A output can be used with most full duplex PC-type serial devices which normally handle RS-232C provided they can reconcile receiving the lower-voltage transmit level of RS-232A. This three wire (Tx/Rx/GND) using the RS-422 input receivers as simple level-shifting inverters as shown in Figure 1 creates RS-232A connection. If the user needs the full voltage levels of RS-232C an additional chip is need. This is demonstrated in Figure 2.

RS-422 is an alternate full-duplex connection which uses two twisted-pair transmission lines (i.e., Tx+/Tx-/Rx+/Rx-) offering long transmission paths and noise-canceling techniques. This distance is typically 4000 feet. This connection is shown in Figure 3. RS-485 is similar to RS-422 with the exception that it uses a single twisted pair in a half-duplex arrangement (i.e., +/-). This means data transmissions must use the same twisted-pair path to travel in both directions, requiring a simple protocol of only one unit seizing the transmission pair at a time while all others listen. This connection is shown in Figure 4.

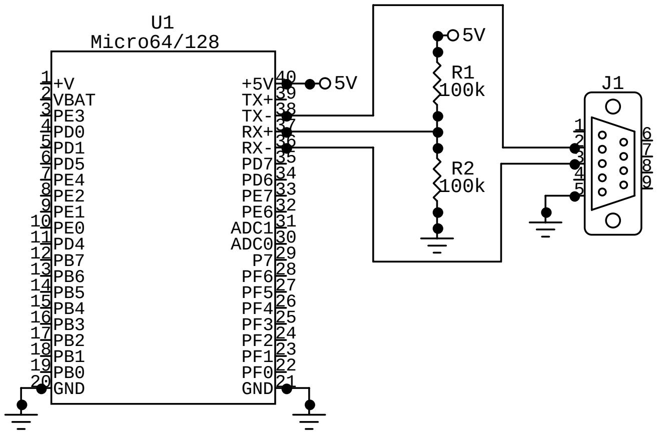


Figure 1: Typical RS-232A Connections

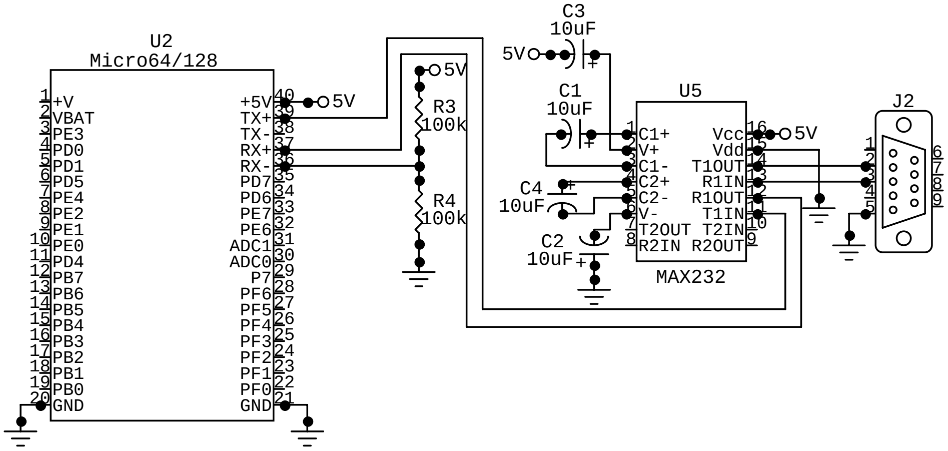


Figure 2: Typical RS-232C Connections

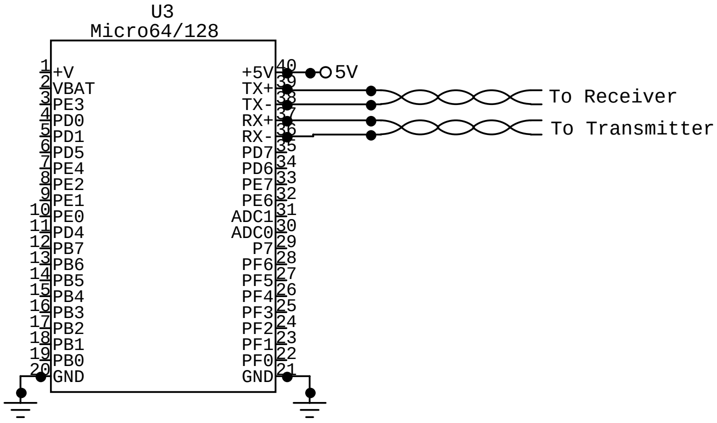


Figure 3: Typical RS-422 Connections

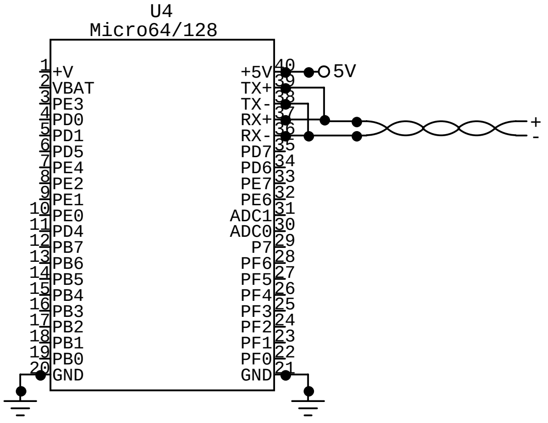


Figure 4: Typical RS-485 Connection

1.2 Sample Application: Networking Micro64/128

Multiple Micro64/128s can be used in a networked multi-drop configuration. Network protocol requires that only one unit is allowed to transmit on the line at a time. All other units are listening in receive mode.

This is accomplished by requiring one Micro64/128 or a device like a PC to be the net master. The master talks to any slave unit either passing information to it or requesting information to it or requesting information from it. The slaves must never answer the master until a response is requested. The master then relinquishes the net to that slave for the response and regains the net when the slave is finished. This arrangement enables multiple controllers to work together gathering numerous inputs and controlling innumerable outputs, independent of the system size. Figure 5 demonstrates a network.

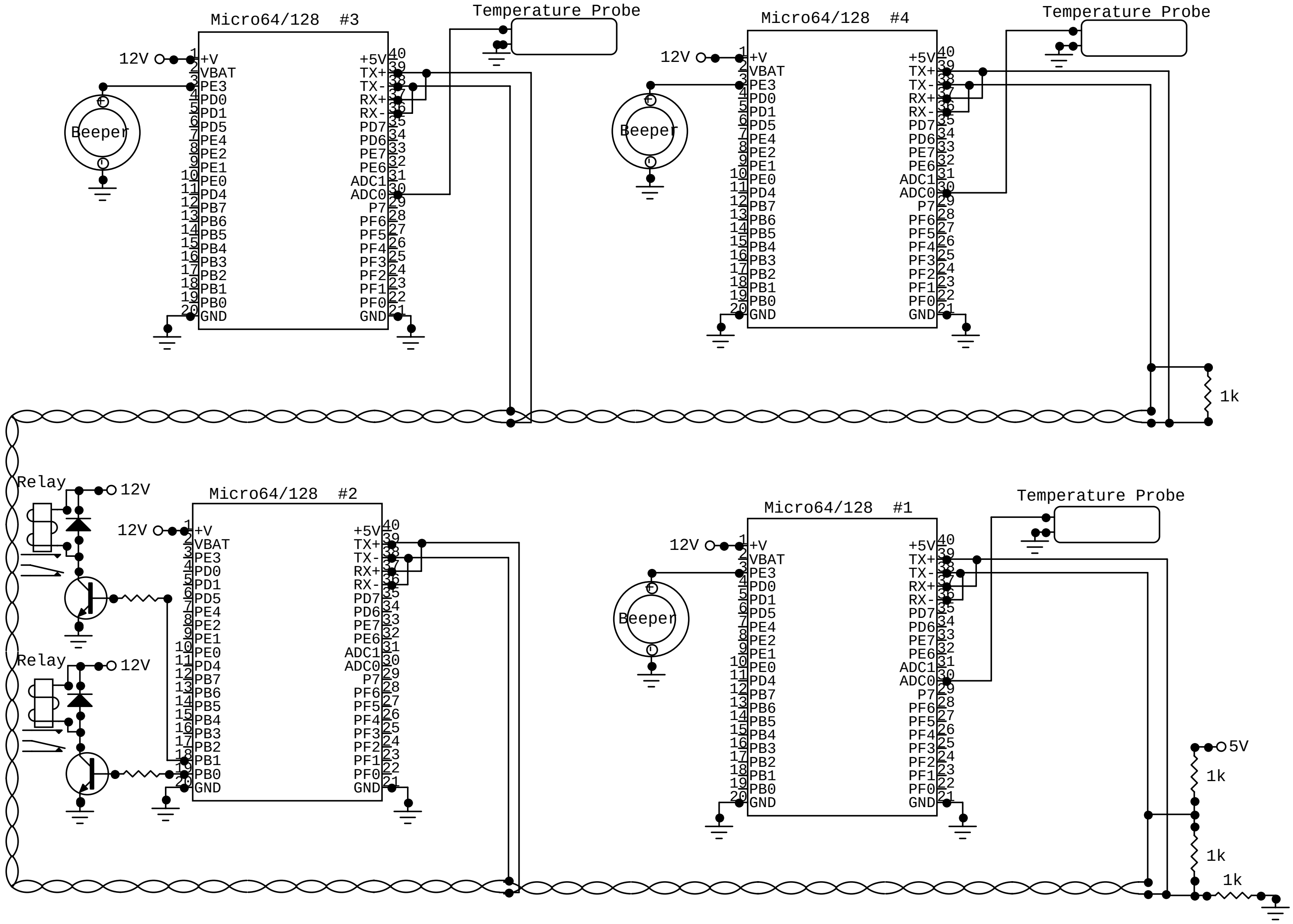


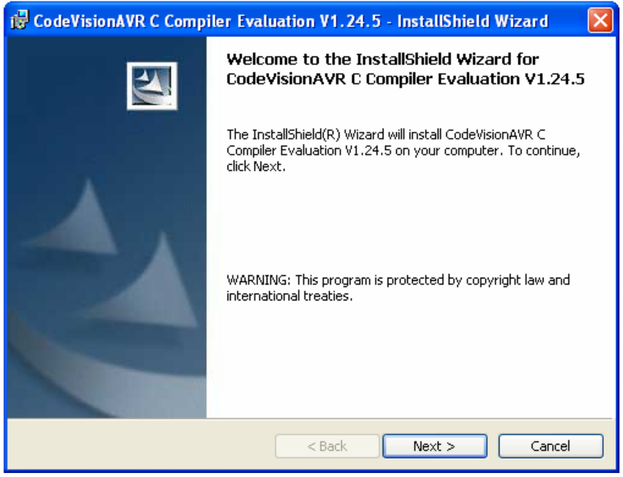
Figure 5: RS-485 Network

Appendix 2.0 Getting Started With the Micro64

2.1 Software Installation

2.10 Installing the CodeVisionAVR C Compiler

1. Open the CodeVisionAVR Demo folder on the CD.
2. Click on setup icon and the window similar to the one below should open.

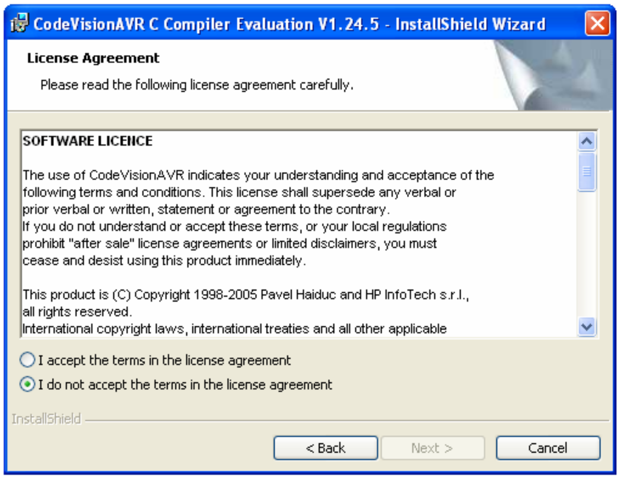

CodeVisionAVR C Compiler Evaluation V1.24.5 - InstallShield Wizard

Welcome to the InstallShield Wizard for CodeVisionAVR C Compiler Evaluation V1.24.5

The InstallShield(R) Wizard will install CodeVisionAVR C Compiler Evaluation V1.24.5 on your computer. To continue, click Next.

WARNING: This program is protected by copyright law and international treaties.

3. Click the Next button and the following window should open.


CodeVisionAVR C Compiler Evaluation V1.24.5 - InstallShield Wizard

License Agreement

Please read the following license agreement carefully.

SOFTWARE LICENCE

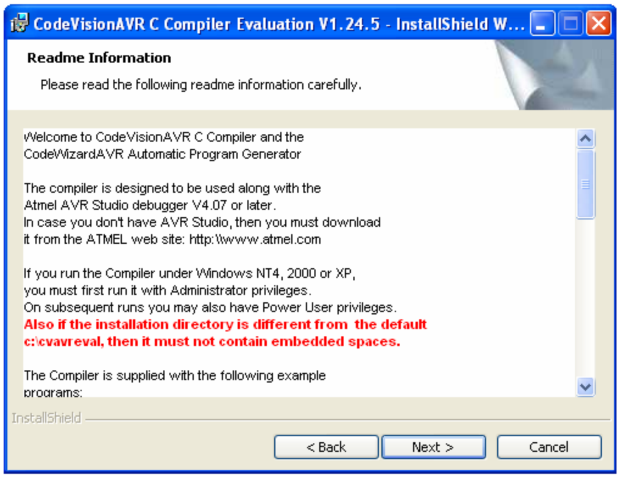
The use of CodeVisionAVR indicates your understanding and acceptance of the following terms and conditions. This license shall supersede any verbal or prior verbal or written, statement or agreement to the contrary.
If you do not understand or accept these terms, or your local regulations prohibit "after sale" license agreements or limited disclaimers, you must cease and desist using this product immediately.

This product is (C) Copyright 1998-2005 Pavel Haiduc and HP InfoTech s.r.l., all rights reserved.
International copyright laws, international treaties and all other applicable

○ I accept the terms in the license agreement
● I do not accept the terms in the license agreement

4. Accept the terms in the license agreement.

5. Click the Next button and the following window should open.


CodeVisionAVR C Compiler Evaluation V1.24.5 - InstallShield W...

Readme Information

Please read the following readme information carefully.

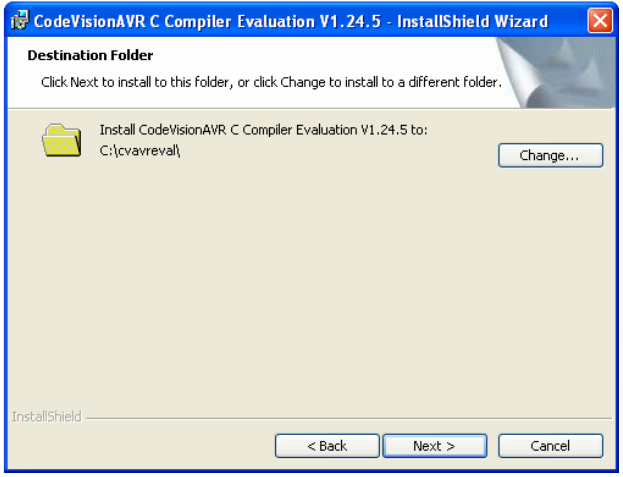
Welcome to CodeVisionAVR C Compiler and the
CodeWizardAVR Automatic Program Generator

The compiler is designed to be used along with the
Atmel AVR Studio debugger V4.07 or later.
In case you don't have AVR Studio, then you must download
it from the ATMEL web site: http:\\www.atmel.com

If you run the Compiler under Windows NT4, 2000 or XP,
you must first run it with Administrator privileges.
On subsequent runs you may also have Power User privileges.
Also if the installation directory is different from the default c:\cvavreval, then it must not contain embedded spaces.

The Compiler is supplied with the following example
programs:

6. Click the Next button and the following window should open.

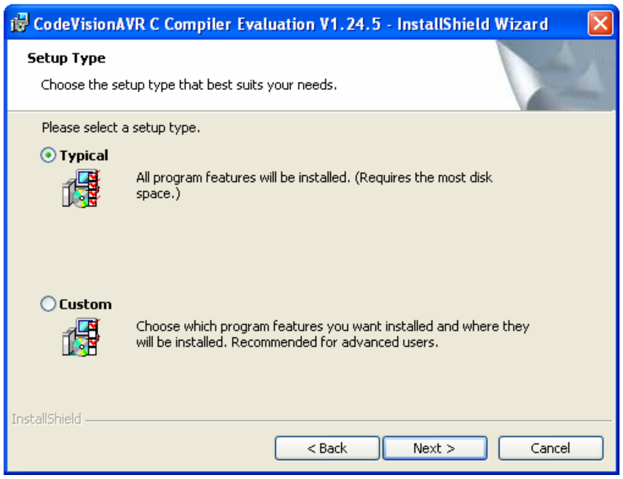

CodeVisionAVR C Compiler Evaluation V1.24.5 - InstallShield Wizard

Destination Folder

Click Next to install to this folder, or click Change to install to a different folder.

Install CodeVisionAVR C Compiler Evaluation V1.24.5 to:
C:\cvavreval\

7. Click the Next button and the following window should open.



CodeVisionAVR C Compiler Evaluation V1.24.5 - InstallShield Wizard

Setup Type

Choose the setup type that best suits your needs.

Please select a setup type.

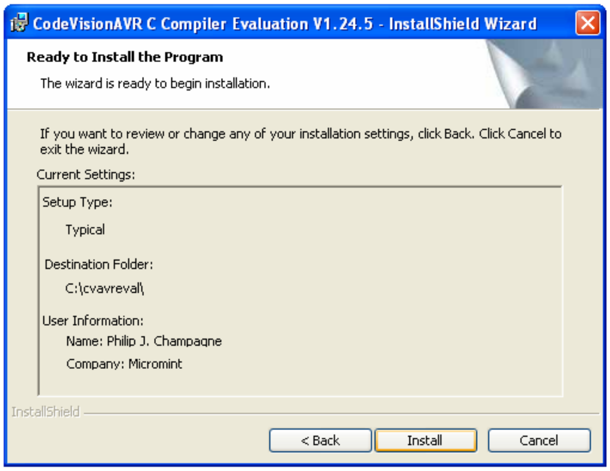
● **Typical**

All program features will be installed. (Requires the most disk space.)

○ **Custom**

Choose which program features you want installed and where they will be installed. Recommended for advanced users.

8. Click the Next button and the following window should open.



CodeVisionAVR C Compiler Evaluation V1.24.5 - InstallShield Wizard

Ready to Install the Program

The wizard is ready to begin installation.

If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.

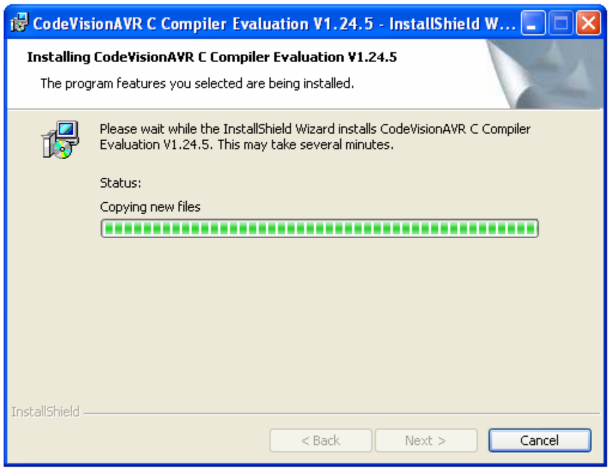
Current Settings:

Setup Type:
Typical

Destination Folder:
C:\cvavreval\

User Information:
Name: Philip J. Champagne
Company: Micromint

9. Click the Install button and the following window should open.



CodeVisionAVR C Compiler Evaluation V1.24.5 - InstallShield W...

Installing CodeVisionAVR C Compiler Evaluation V1.24.5

The program features you selected are being installed.

Please wait while the InstallShield Wizard installs CodeVisionAVR C Compiler Evaluation V1.24.5. This may take several minutes.

Status:

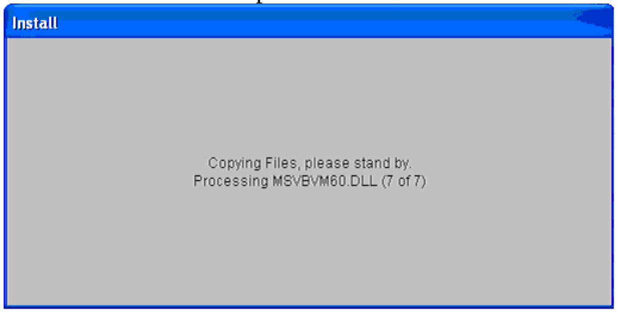
Copying new files

10. Wait for the program to install then press the Finish button.

2.11 Installing the Micro64's Boot Loader Software

1. Open the Boot Loader folder on the CD.

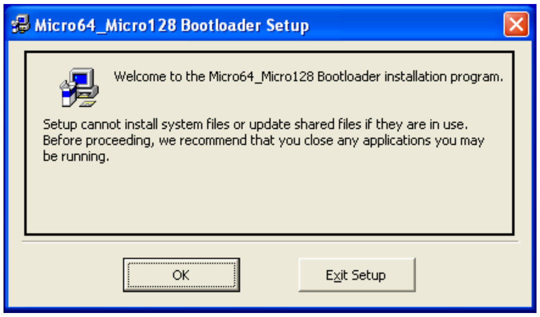
2. Click on setup icon and the window similar to the one below should open.



Install

Copying Files, please stand by.
Processing MSVBVM60.DLL (7 of 7)

3. Wait for the files to finish loading and the following window should appear.

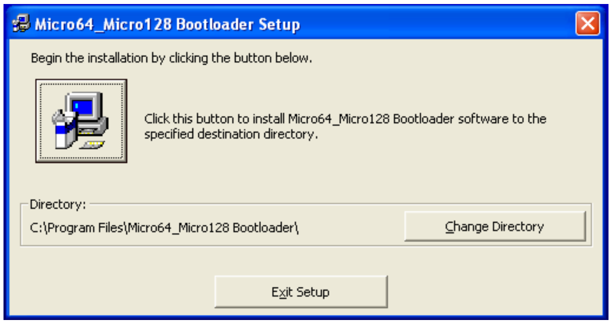


Micro64_Micro128 Bootloader Setup

Welcome to the Micro64_Micro128 Bootloader installation program.

Setup cannot install system files or update shared files if they are in use. Before proceeding, we recommend that you close any applications you may be running.

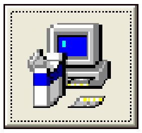
4. Click on the OK button to continue with the install and the following window should open.

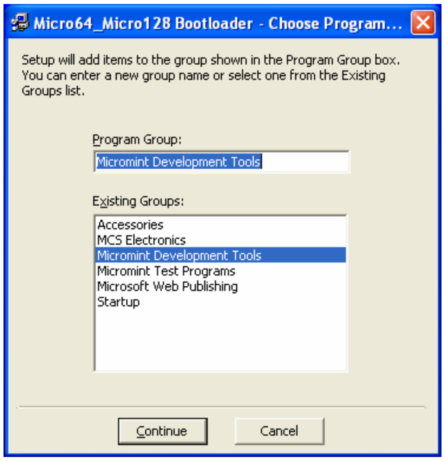


Begin the installation by clicking the button below.

Click this button to install Micro64_Micro128 Bootloader software to the specified destination directory.

Directory: C:\Program Files\Micro64_Micro128 Bootloader\

5. Click on the  button and the following window should open.



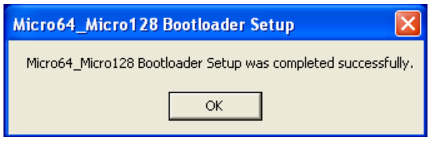
Setup will add items to the group shown in the Program Group box. You can enter a new group name or select one from the Existing Groups list.

Program Group: Micromint Development Tools

Existing Groups:

Accessories
MCS Electronics
Micromint Development Tools
Micromint Test Programs
Microsoft Web Publishing
Startup

6. Click on the Continue button and files should start loading into your computer. After the files are done loading the following window should open.

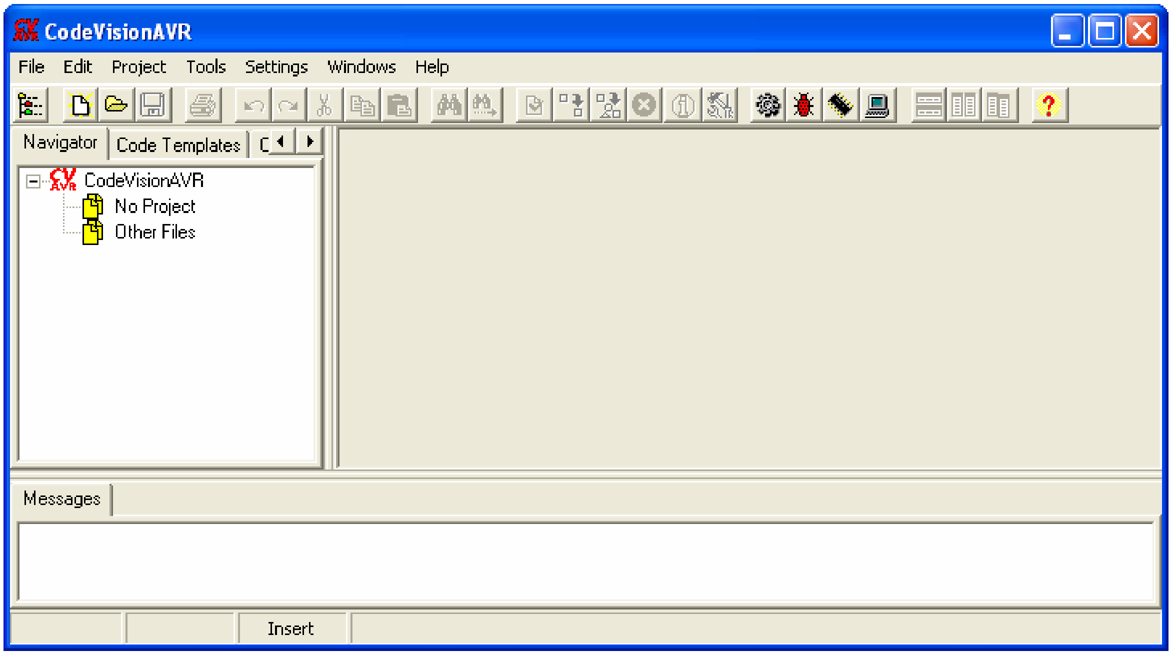


Micro64_Micro128 Bootloader Setup was completed successfully.

7. Click OK to finish the installation.

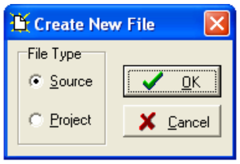
2.2 Compiling and running HelloWorld.c

When the compiler is opened for the first time the GUI (Graphical User Interface) should look like the following image.

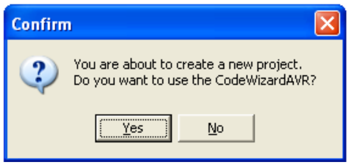


Please follow the following steps to setup the compiler for a Micro64/128.

1. Click on “File|New” menu option or click the toolbar button and the following window will be displayed.

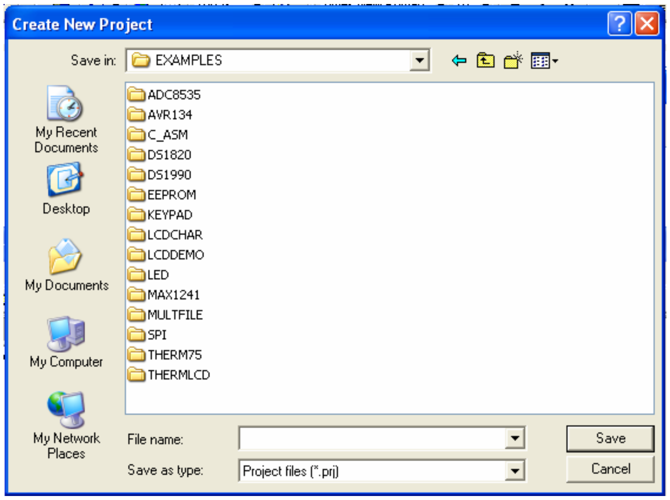


2. Select “Project” and press “OK” and the following window will be displayed.



You are about to create a new project.
Do you want to use the CodeWizardAVR?

3. For simplicity we will not use the CodeWizardAVR so click “No” and the following window should open.



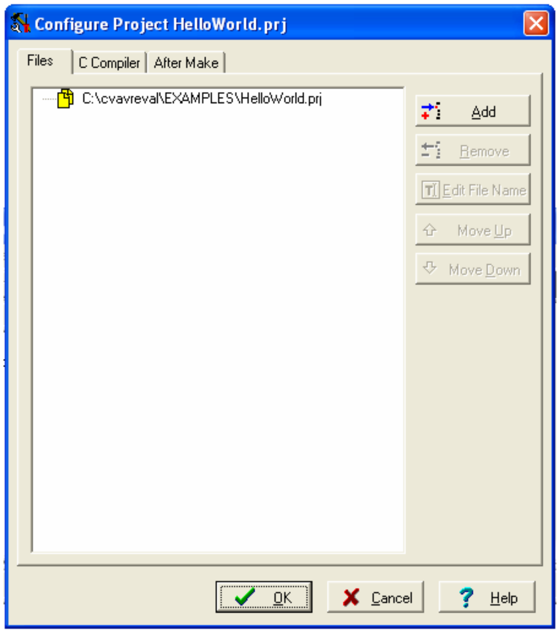
Save in: EXAMPLES

ADC8535
AVR134
C_ASM
DS1820
DS1990
EEPROM
KEYPAD
LCDCHAR
LCDDEMO
LED
MAX1241
MULTFILE
SPI
THERM75
THERMLCD

File name:

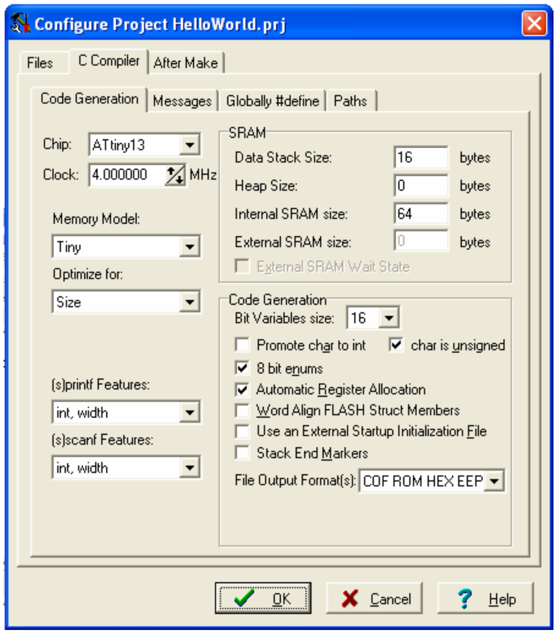
Save as type: Project files (*.prj)

4. Type in a file name of your choice and click “Save”. For demonstration purposes HelloWorld was chosen for the File name. The following screen should appear.



C:\cvavreval\EXAMPLES\HelloWorld.prj

5. Click on the “C Compiler” tab and the window will look similar to the following.



Chip: ATtiny13

Clock: 4.000000 MHz

Memory Model: Tiny

Optimize for: Size

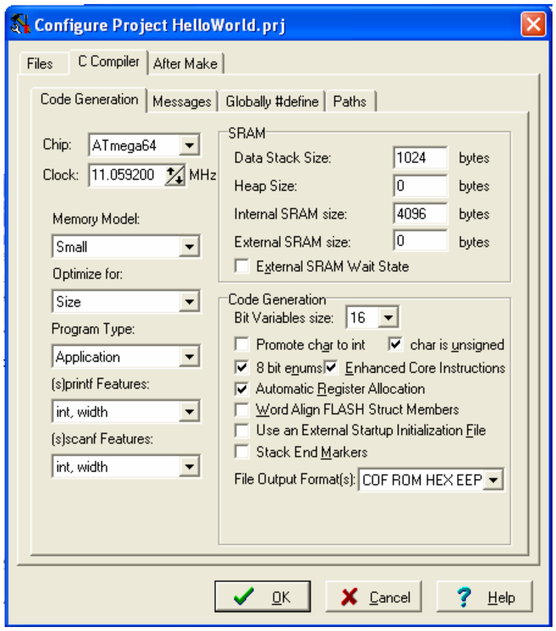
(s)printf Features: int, width

(s)scanf Features: int, width

SRAM
Data Stack Size: 16 bytes
Heap Size: 0 bytes
Internal SRAM size: 64 bytes
External SRAM size: 0 bytes
□ External SRAM Wait State

Code Generation
Bit Variables size: 16
□ Promote char to int ☑ char is unsigned
☑ 8 bit enums
☑ Automatic Register Allocation
□ Word Align FLASH Struct Members
□ Use an External Startup Initialization File
□ Stack End Markers
File Output Format(s): COF ROM HEX EEP

6. Click the arrow on Chip: ATtiny13 ▾ and select Atmega64 for Micro64 or Atmega128 for Micro128.
7. Click next to the first digit in the following Clock: 4.000000 MHz and change it to 11.0592.
8. After you complete the changes the window should look like the following for the Micro64/128.



Chip: ATmega64

Clock: 11.059200 MHz

Memory Model: Small

Optimize for: Size

Program Type: Application

(s)printf Features: int, width

(s)scanf Features: int, width

SRAM

Data Stack Size:	1024	bytes
Heap Size:	0	bytes
Internal SRAM size:	4096	bytes
External SRAM size:	0	bytes

☐ External SRAM Wait State

Code Generation

Bit Variables size: 16

☐ Promote char to int ☑ char is unsigned

☑ 8 bit enums ☑ Enhanced Core Instructions

☑ Automatic Register Allocation

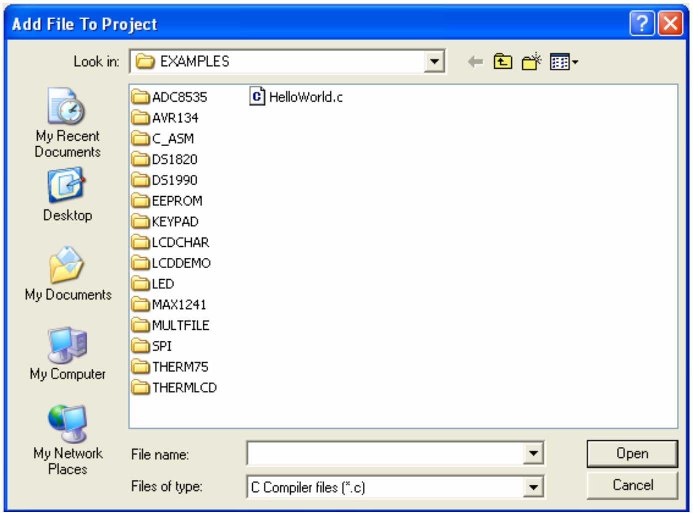
☐ Word Align FLASH Struct Members

☐ Use an External Startup Initialization File

☐ Stack End Markers

File Output Format(s): COF ROM HEX EEP

9. All other parameters do not need to be changed in order to compile a simple program. Please refer to the CodeVisionAVR's help section for further details on the other parameters. Click on the "Files" tab to continue.
10. The next step would be to add a source file. To add a source file click the Add button and the following window will appear.



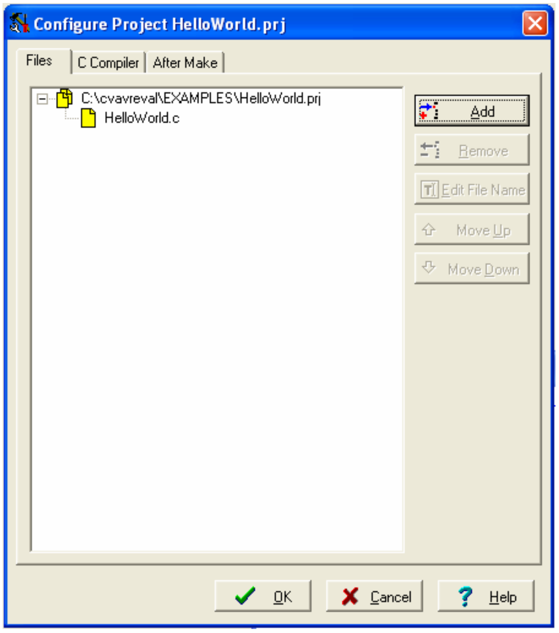
Look in: EXAMPLES

ADC8535
AVR134
C_ASM
DS1820
DS1990
EEPROM
KEYPAD
LCDCHAR
LCDDEMO
LED
MAX1241
MULTFILE
SPI
THERM75
THERMLCD
HelloWorld.c

File name:

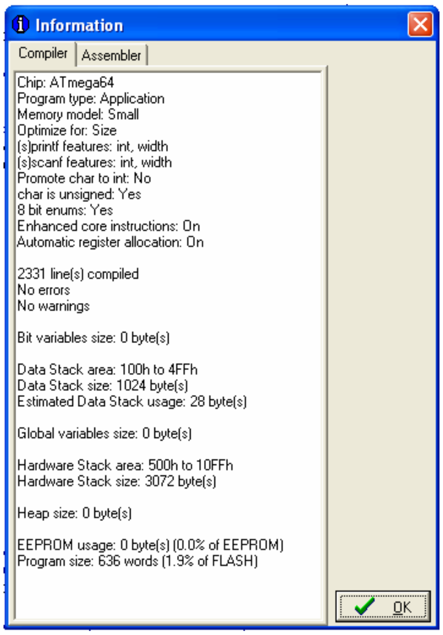
Files of type: C Compiler files (*.c)

11. Find and Select "HelloWorld.c" and click "Open". HelloWorld.C can be found on the CD or on Micromint's Micro64/128 Datasheet & Application Notes webpage. http://www.micromint.com/app_notes/micro64_128.htm The following window should appear.



C:\cvavreval\EXAMPLES\HelloWorld.prj
HelloWorld.c

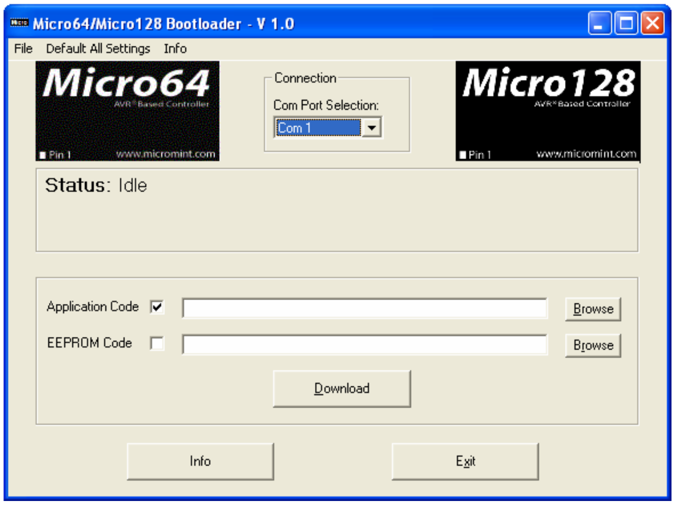
12. Click the "OK" button and then you will be ready to compile the program.
13. To make the hex file for the program click on "Project|Make" menu option or click the tool bar button. The following window should open.



```
Chip: ATmega64
Program type: Application
Memory model: Small
Optimize for: Size
(s)printf features: int, width
(s)scanf features: int, width
Promote char to int: No
char is unsigned: Yes
8 bit enums: Yes
Enhanced core instructions: On
Automatic register allocation: On

2331 line(s) compiled
No errors
No warnings

Bit variables size: 0 byte(s)

Data Stack area: 100h to 4FFh
Data Stack size: 1024 byte(s)
Estimated Data Stack usage: 28 byte(s)

Global variables size: 0 byte(s)

Hardware Stack area: 500h to 10FFh
Hardware Stack size: 3072 byte(s)

Heap size: 0 byte(s)

EEPROM usage: 0 byte(s) (0.0% of EEPROM)
Program size: 636 words (1.9% of FLASH)
```

14. Click "OK". Now it is time to program the Micro64. using the Boot Loader software. If you already installed the Boot Loader software in section 2.11 then click on "START|Programs| Micromint Development Tools| Micro64_Micro128 Bootloader" and the following window will open.

The boot loader uses USART1 to download programs to the Micro64/128. If you are using the development board then please refer to section 6.31 of the Micro64/128 datasheet to set-up the jumpers properly.


Micro64/Micro128 Bootloader - V 1.0

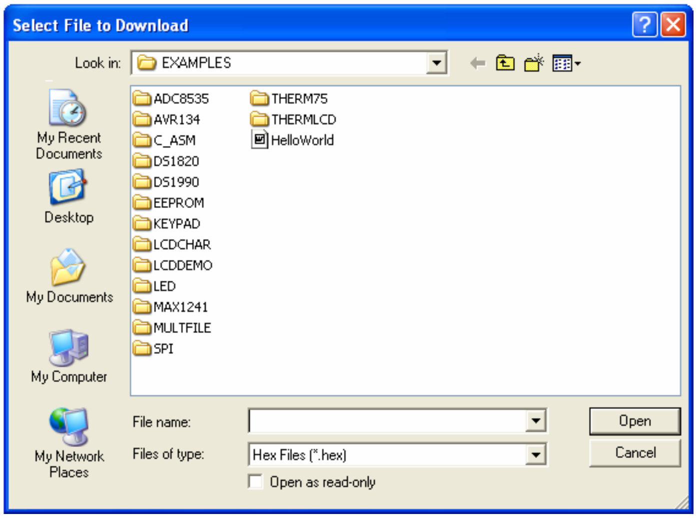
Com Port Selection: Com 1

Status: Idle

Application Code ☑

EEPROM Code □

15. Click the Browse button next to “Application Code” to select the hex file that was created by the compiler. A similar window like the following window will open.


Select File to Download

Look in: EXAMPLES

ADC8535, AVR134, C_ASM, DS1820, DS1990, EEPROM, KEYPAD, LCDCHAR, LCDDEMO, LED, MAX1241, MULTFILE, SPI, THERM75, THERMLCD, HelloWorld

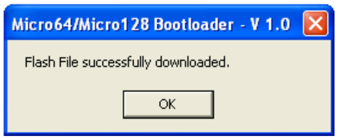
File name:

Files of type: Hex Files (*.hex)

□ Open as read-only

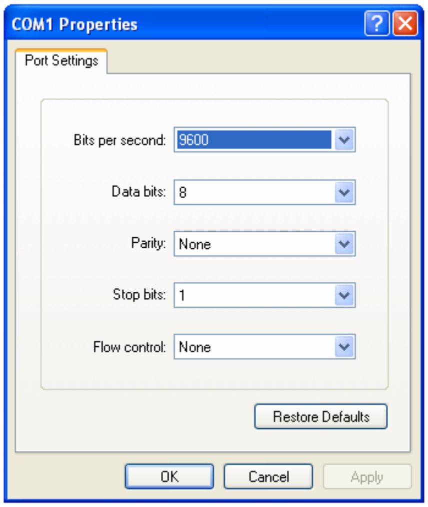
16. Find and select the file “HelloWorld.hex” and click “Open”.

17. Click on the Download button to send the file through COM1 to the Micro64.
NOTE: The Boot Loader software uses the COM Port when it is sending a file to the Micro64. If the software says it is unable to open the COM port there can be a few reasons why.
    1. The COM port selected is already in use. Close the application using the COM port or select another one.
    2. The Micro64/128 is constantly sending information to the closed COM port. Hold the RESET button or disconnect the power the Micro64/128 then click on the Download button.
18. Press and release the RESET button on the Micro64 development board or cycle the power to the module. After the program is done downloading the file the following message box will open.


Micro64/Micro128 Bootloader - V 1.0

Flash File successfully downloaded.

19. Click “OK” and open HyperTerminal with the following settings.


COM1 Properties

Port Settings

Bits per second: 9600

Data bits: 8

Parity: None

Stop bits: 1

Flow control: None

20. Press and release the RESET button or cycle the power. Wait for a moment and you should see “Hello World” constantly displaying on the screen. That is all there is to getting a program up and running.

2.3 HelloWorld.c Listing.

```
/******************************************
Program : HelloWorld Example for Micro64 or Micro128
Company : Micromint, Inc
*********************************************/

#include <mega64.h>                    // Comment this line out for Micro128
// For Micro128 make sure that you goto "Project|Configure", to the
// "C Compiler" tab and change the chip to ATmega128
//#include <mega128.h>                  // Uncomment this line for Micro128
#include <stdio.h>                      // Standard I/O library

// Declare your global variables here
#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
#define UDRE 5
#define RXC 7
#define TXC 6

#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)
#define TX_COMPETE (1<<TXC)
int COM;         // if COM = 0 then use USART0 if it = 1 then use USART1
//****************************************************************************************************
// The program lines between the ** lines, like the above line, is needed in order to use the
// printf command for USART1.

/* inform the compiler that an alternate version
   of the getchar function will be used for USART1 */

#define _ALTERNATE_GETCHAR_

/* now define the new getchar function */
char getchar(void)
{
/* write your code here */
char status,data;
    switch(COM)
    {
      case 0:
          while (1)
               {
               while (((status=UCSR0A) & RX_COMPLETE)==0);
               data=UDR0;
               if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
               return data;
               };
      case 1:
          while (1)
```

```
                {
                while (((status=UCSR1A) & RX_COMPLETE)==0);
                data=UDR1;
                if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
                return data;
                };
     };
}

/* inform the compiler that an alternate version
   of the putchar function will be used */
#define _ALTERNATE_PUTCHAR_

/* now define the new putchar function */
void putchar(char c)
{
/* write your code here */
     switch(COM)
     {
       case 0:
           while ((UCSR0A & DATA_REGISTER_EMPTY)==0);
           UDR0=c;
           break;
       case 1:
           while ((UCSR1A & DATA_REGISTER_EMPTY)==0);
           UDR1=c;
           while ((UCSR1A & TX_COMPETE)==0);
     };
}
//*******************************************************************************************
void main(void)
{
// Declare your local variables here

// Set up USART1's Baud rate at 9600 bps with a 11.0592 MHz Crystal
UCSR1A=0x00;      // RX EN, TX EN
UCSR1B=0x18;     // RX EN, TX EN
UCSR1C=0x06;     // 8N1
UBRR1H=0x00;    // Baud rate high - 9600
UBRR1L=0x47;   // Baud rate low

COM = 1;                          // Use USART1
DDRD.6 = 0;                                                                          // Make
PORTD.6 an output
PORTD.6 = 1;                                                                         // Enable the
RS485 control line


while (1)
      {
        printf("Hello World\r\n");
      };
}
```