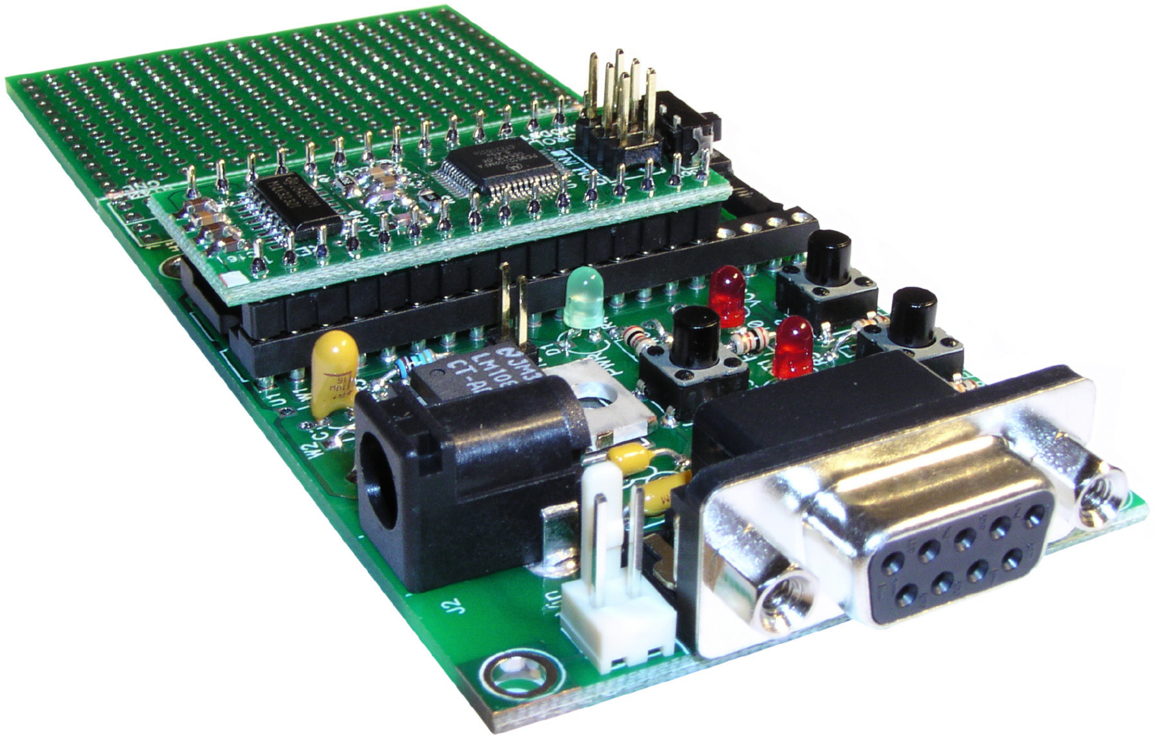
Using Your NanoCore12 Microcontroller Module



Technological Arts

www.technologicalarts.com

Revison 0c

DISCLAIMER

E S D W A R N I N G

This product, like all microcontroller products, uses semiconductors that can be damaged by electrostatic discharge (ESD). When handling, care must be taken so that the devices are not damaged. Damage due to inappropriate handling is not covered by the warranty.

The following precautions must be taken:
• Do not open the protective conductive packaging until you have read the following, and are at an approved anti-static work station.
• Use a conductive wrist strap attached to a good earth ground.
• If working on a prototyping board, use a soldering iron or station that is marked as ESD-safe. Always disconnect the microcontroller from the prototyping board when it is being worked on.
• Always discharge yourself by touching a grounded bare metal surface or approved anti-static mat before picking up an ESD-sensitive electronic component.
• Use an approved anti-static mat to cover your work surface.

1 INTRODUCTION

1.1 WELCOME!

With **NanoCore12**, you are now ready to explore the power and versatility of Freescale's advanced 16-bit microcontroller family! Whether you're new to Freescale microcontrollers or you've used some of the earlier ones (such as 68HC05, 68HC11, or 68HC12), you'll be impressed with the well thought-out design and implementation of the HCS12 family. **NanoCore12** gives you the opportunity to explore the 9S12C family's potential at a very affordable price! Add to that the proven advantages of it's popular DIP format, and you'll see why you picked a real winner!

1.2 SUPPORT

To help you get the most out of this product, and to make the experience as enjoyable and productive as possible, we've put together a comprehensive website, loaded with resources, support, and applications information. If you experience any difficulties, or need help with your application, the World Wide Web is arguably the most valuable resource available to you. There you'll find the latest information, software, and trouble-shooting help, as well as discussion groups where you can network with people around the globe to get the answers you need. So if you still need help, or have questions after reading this manual and perusing the contents of the included CD, visit our Support Forum: **www.technologicalarts.net** and tap into the collective! Also, be sure to join Freescale's **16-bit Microcontroller** discussion group, at http://forums.freescale.com/freescale/board?board.id=16BITCOMM

1.3 PRODUCT CONFIGURATION

NanoCore12 was designed as a versatile evaluation and application tool for the Freescale HCS12 "C" family of microcontrollers. It is suitable for developing applications for any of the derivatives (e.g. 9S12C32, 9S12C128). All of the subsystems are identical, with the differentiating factor among the chips being the amounts of Flash and RAM included on the chip. NanoCore modules are completely self-contained, including RS232 interface and 5V regulator circuitry right on the module. You can simply plug it into any solderless breadboard to use it.

For added convenience, though, a Docking Module is offered, which provides additional support for working with the microcontroller. It includes a bigger voltage regulator which has extra available current to spare for your applications. The voltage regulator also has a jumper option for 3-Volt operation. Other features include a standard 9-pin D-sub connector for the RS232 serial interface, a socket for the MCU module, a couple of user LEDs and pushbuttons, and a snap-off prototyping area. A similar product, the School Board, has a section of solderless breadboard instead of the snap-off prototyping area. This configuration is ideal for quick experimentation and training purposes. Product bundles are offered for both, and include a power supply, a serial cable, and a resource CD containing this manual, example programs, software tools, and a collection of MCU documentation from Freescale. An optional CD containing the free 32K Special Edition of Metrowerk's CodeWarrior C compiler is available upon request.

NanoCore12 modules are offered in 24-pin, 32-pin, and 40-pin DIP footprints that plug into the DIP socket on the Docking Module or the socket strips on a School Board. While the standard configuration incorporates the 9S12C32 MCU, a 9S12C128 version is also available on the 40-pin "MAX", offering four times the program space (128K vs. 32K) and double the RAM (4K vs. 2K).

1.4 NC12 VS. TRADITIONAL EVALUATION BOARDS

Most available evaluation and development systems tend to be too expensive and bulky for embedding into a real application, so they lie on a shelf gathering dust once you've reached a certain point in the learning curve. Or maybe you think up some clever way to hack it apart and make it fit inside your robot or product prototype. Even then, the prototyping area provided is often limited, and does not lend itself to re-usability. And what if you burn out a chip the night before the contest or product demo? What a mess to repair or re-design!

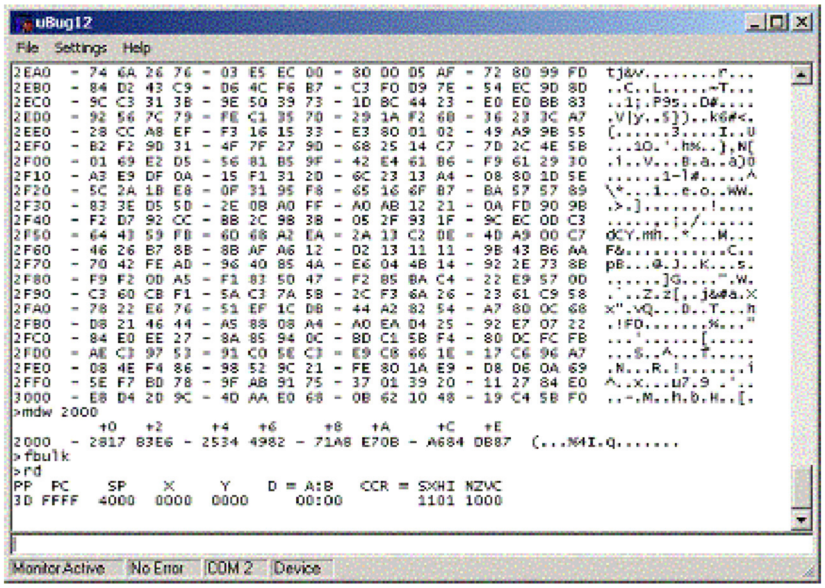
NanoCore12 solves all of these problems and more! Since the MCU modules are packaged in standard DIP footprints, they are modular and re-usable. They can simply be plugged into your application, or moved from one application board to another, allowing your whole system to be re-configured at the last minute. The prototyping area on the Docking Module offers you a platform for your initial application, which can be snapped off and discarded when you no longer need it. Two 40-pin connector footprints remain, and can be used with various types of connectors, ribbon cables, backplanes, prototyping cards, etc. that are available now or are under development. Make a point to visit our website from time to time, just to see what's new. There you'll find that various 40-pin connectors are offered for sale, if you have trouble locating them from your local electronics store.

The detachable nature of the prototyping card means that you can easily replace it with other cards, including some application cards available from Technological Arts. You can build up a collection of different application circuits, and use them all with the same microcontroller board. Of course, if the prototyping card approach doesn't suit you, the standard DIP format of **NanoCore12** makes it easy for you to incorporate it into your own designs. In fact, it's pinout is a superset of the popular 24-pin DIP MCU modules available from other vendors. You can plug it into a DIP socket or directly into a solderless bread-board. If you've already invested in an application board from another vendor, in most cases you can plug a NanoCore12 DIP module in and accomplish some serious applications.

The modular nature of this product is especially advantageous in an educational environment, where the student can progress from simple to more complex applications throughout a semester, or from one course to the next-- even incorporating the board into a final project. In fact, where budgets are tight, different students can share the same microcontroller module, and plug in their own interface cards when it's their turn to use it.

1.5 RESIDENT DEBUG/MONITOR

Residing in a 2K protected block of on-chip flash memory is Freescale's versatile Serial Monitor program. When used with uBug12 (a free Windows application created by Technological Arts), you can display and edit memory and registers, erase and program flash, set breakpoints, and do instruction tracing. Flip a switch on NanoCore to Run mode, and your program runs automatically from Flash, following reset or powerup. See Chapter 2 and Appendix A for details on uBug12 and the resident monitor.



```
2EA0  - 74 6A 26 76 - 03 E5 EC 00 - 80 00 D5 AF - 72 80 99 FD   tj&v........r...
2EB0  - 84 D2 43 C9 - D6 4C F6 B7 - C3 F0 D9 7E - 54 EC 9D 8D   ..C..L.....~T...
2EC0  - 9C C3 31 3B - 9E 50 39 73 - 1D 8C 44 23 - E0 E0 BB 83   ..1;.P9s..D#....
2ED0  - 92 56 7C 79 - FE C1 35 70 - 29 1A F2 68 - 36 23 3C A7   .V|y..5p)..h6#<.
2EE0  - 28 CC A8 EF - F3 16 15 33 - E3 80 01 02 - 49 A9 9B 55   (.......3....I..U
2EF0  - B2 F2 9D 31 - 4F 7F 27 9D - 68 25 14 C7 - 7D 2C 4E 5B   ...1O.'.h%..},N[
2F00  - 01 69 E2 D5 - 56 81 B5 9F - 42 E4 61 B6 - F9 61 29 30   .i..V...B.a..a)0
2F10  - A3 E9 DF 0A - 15 F1 31 2D - 6C 23 13 A4 - 08 80 1D 5E   ......1-l#.....^
2F20  - 5C 2A 1B E8 - 0F 31 95 F8 - 65 16 6F 87 - BA 57 57 89   \*...1..e.o..WW.
2F30  - 83 3E D5 5D - 2E 08 A0 FF - A0 AB 12 21 - 0A FD 90 9B   .>.]........!....
2F40  - F2 D7 92 CC - 8B 2C 98 38 - 05 2F 93 1F - 9C EC 0D C3   .......;./......
2F50  - 64 43 59 FB - 6D 68 A2 EA - 2A 13 C2 DE - 4D A9 00 C7   dCY.mh..*...M...
2F60  - 46 26 B7 8B - 8B AF A6 12 - D2 13 11 11 - 9B 43 B6 AA   F&...........C..
2F70  - 70 42 FE AD - 96 40 85 4A - E6 04 4B 14 - 92 2E 73 8B   pB...@.J..K...s.
2F80  - F9 F2 0D A5 - F1 83 5D 47 - F2 85 BA C4 - 22 E9 57 0D   .......]G....".W.
2F90  - C3 60 CB F1 - 5A C3 7A 5B - 2C F3 6A 26 - 23 61 C9 58   .`..Z.z[,.j&#a.X
2FA0  - 78 22 E6 76 - 51 EF 1C DB - 44 A2 82 54 - A7 8D 0C 68   x".vQ...D..T...h
2FB0  - D8 21 46 44 - A5 88 08 A4 - A0 EA D4 25 - 92 E7 07 22   .!FD........%..."
2FC0  - 84 E0 EE 27 - 8A 85 94 0C - 8D C1 5B F4 - 80 DC FC FB   ...'........[....
2FD0  - AE C3 97 53 - 91 C0 5E C3 - E9 C8 66 1E - 17 C6 96 A7   ...S..^...f.....
2FE0  - 08 4E F4 86 - 98 52 9C 21 - FE 8D 1A E9 - D8 D6 0A 69   .N...R.!.......i
2FF0  - 5E F7 BD 78 - 9F AB 91 75 - 37 01 39 20 - 11 27 84 E0   ^..x...u7.9 .'..
3000  - E8 D4 2D 9C - 4D AA E0 68 - 0B 62 10 48 - 19 C4 5B F0   ..-.M..h.b.H..[.
>mdw 2000
          +0   +2      +4   +6      +8   +A      +C   +E
2000  - 2817 83E6 - 2534 4982 - 71A8 E70B - A684 DB87   (...%4I.q.......
>fbulk
>rd
PP  PC     SP    X     Y      D = A:B    CCR = SXHI NZVC
3D FFFF   4000  0000  0000       00:00          1101 1000
```

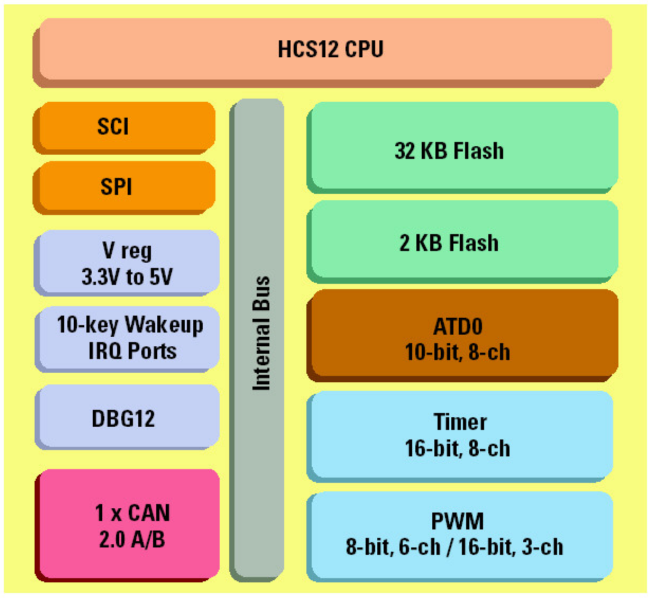
1.6 COMMUNICATIONS

An RS-232C serial interface port connector (RX & TX only) is included on both the Docking Module and School Board, allowing communication with a PC com port, or any other device which has an RS-232 serial port, via a standard 9-pin serial port extension cable. The RS-232 channel is implemented via the SCI of the MCU, and when NanoCore is reset in Load mode, the resident Serial Monitor uses this port to communicate with an appropriate program running on your PC (e.g. uBug12). In Run mode, the RS-232 port is available for your application.

While the MCU does support Controller Area Network (CAN), no physical layer circuit is provided on the 24-pin and 32-pin versions of NanoCore12. If you would like to use CAN on either of these modules, you'll need to add the appropriate transceiver circuit to PM0 and PM1 of the MCU module. However, the 40-pin module (NanoCore12MAX) includes an on-board CAN transceiver circuit, so the CAN interface can be used directly, without the need for additional circuitry.

In passing, it should be mentioned that the MCU also supports Serial Peripheral Interface (SPI). Since this is a logic-level protocol, meant for local communications among peripheral chips, no transceivers are required nor are they provided. Commonly used SPI chips and modules include: serial memory (e.g. EEPROM, Flash), temperature controllers, clock/calendar chips, DACs, MP3 decoders, etc.

See the 9S12C datasheets for details on all of these subsystems.



2 GETTING STARTED

- ***Be sure to read and follow the Safe Handling Procedures outlined inside the manual's front cover***
- Perform a visual check of the hardware for any damage during transit
- Connect the Docking Module's (or School Board's) RS232 port (J3) to a com port of a personal computer, using the cable supplied
- Set switch SW2 to the LOAD position
- Connect the supplied power source to J1
- Locate and install the Windows application called uBug12, included on the CD (also on our website)
- Launch uBug12
- uBug12 will display a short message, followed by its command prompt
- Activate the serial port connection by entering CON x (where x is the comport you are using; usually 1 or 2)
- The Module is now ready for your commands (see Appendix B for a full list of uBug12 commands)

To download one of the supplied example programs into flash and execute it, follow these steps:
- Type **fbulk** <enter> at the uBug12 prompt to erase any existing program
- Type **fload** <enter>
- From the displayed file browser, select one of the example program's output files (.s2 or .s28 file)
- After loading has finished, move switch SW2 to the RUN postion, and press the Reset button (SW1)
- The program will run
- If you wish to debug the program, move SW2 back to the LOAD position and press Reset
- The uBug12 prompt will appear
- Use uBug12 commands to debug your code (see Appendix B for a full list of uBug12 commands)

2.1 POWER OPTIONS

A DC power supply is included with most bundle configurations. It is the recommended power source when you're starting out. If for some reason it's not convenient (eg. you don't want an extension cord trailing around behind your robot :>), there are a couple of alternatives:

Option 1: connect a DC voltage of 6 Volts or more (absolute maximum: 24 VDC) via the external power connector, J1, on the Docking Module or School Board. (If you aren't using a Docking Module or School Board, apply a voltage of 6 to 12 V between the Vin (positive) pin and the Vss (Ground) pin on the NanoCore module. Refer to the NanoCore12 pinout diagrams on the back page of this manual to determine the correct connections.) Your DC supply does not need to be regulated, but it should be capable of supplying at least 100 mA (more if you will be using a BDM pod, or you're driving other circuits as well). If your supply will also be driving motors, make sure to isolate it before feeding it into the module (to protect it from electrical noise generated by the motor coils). You can do this by putting inductors (10uH, nominal) in series with both the + and - leads. Preferably, use the red & black power wire provided (order code: PCJ1-8). Red is positive, and black is negative (ground). *CAUTION! Make sure you have the polarity correct!*

Option 2: supply regulated 5VDC (or 3VDC if using 3-Volt mode) via the Vcc (+3V) pin and Vss (Ground) pin on the module. See module pinout diagram in Appendix C for the Vcc and GROUND pins to use. *CAUTION! Double-check your connections before applying power! If you are applying 3V, be sure to read the notes on 3-Volt operation in chapter 3.*

2.2 DEMO PROGRAMS

We've included a few demo programs on the CD-ROM (also available from the product webpage) to give you a starting point. There are some examples in C and some in assembler. The source code is included, so you're free to modify them all you want!

2.3 USING THE DEBUG/MONITOR

uBug12 is a Windows-based graphical user interface (GUI) for Freescale's HCS12 Serial Monitor program. It aims to emulate the most common debug/monitor commands, and to provide an easy-to-use interface. The following paragraphs will help you get started with uBug12.

One nice feature you'll discover is command line history. Use the UP and DOWN arrows on your keyboard to recall previously typed commands. You can then edit them, as needed, before hitting <ENTER>.

2.4 NEW TO FREESCALE MICROCONTROLLERS?

If you've come from an 8051, PIC, or other background (or have never used micro-controllers before), you should get up to speed on Freescale MCUs by reading *Understand-ing Small Microcontrollers,* found on the CD-ROM. Written by Freescale's Jim Sibigtroth, principal design engineer of the HC12 family, this excellent book uses an earlier MCU (68HC05) to introduce you to the basic concepts and design philosophy upon which the 9S12 was built. You should also make sure to have a copy of the MC68HC11 Reference Manual, since it contains detailed descriptions and examples for many of the hardware subsystems.

2.6 MIGRATING FROM 68HC11

If you are already experienced with the 68HC11 family of microcontrollers, writing programs for the HCS12 family will not present a big challenge (don't throw away your HC11 Reference Manual-- the trusty "pink book"). In fact, you can use your existing 68HC11 assembly code and re-assemble it to run on the CPU12 core, but there are a few things to keep in mind.

Assembler syntax. You may need to edit your source file to conform to the syntax and directives of the HC12 assembler you'll be using. There are several assemblers available (e.g. AS12, MiniIDE, IASM12, MCUez), and each has its own syntax to be aware of.

Register Block. Instead of $1000, the register block default location is $0000 through $03FF, and there are a few hundred registers! You'll need to locate the relevant registers for the subsystems you plan to use, and make sure they are properly configured. The good news is that you can safely ignore registers for the subsystems you are not using-- the reset defaults are at safe settings.

RAM location. Following reset, the memory map configuration has the register block overlapping RAM, starting at $0000, with registers taking priority, so the first 1K bytes of RAM are not usable. In order to free up all of the RAM. the monitor program re-maps RAM to start at $3800 and go to $3FFF (via the RAMinit register). This means you'll need to initialize the Stack Pointer to $4000 (on the HCS12, the stack pointer points to the address following the top of the stack).

High-speed Bus. The default bus speed is half the crystal frequency of 8 MHz, so it is 4 MHz. If you enable the PLL, it will be even higher (up to 24 MHz). This will mean changing some initialization values for control registers and revising delay constants if you are using any software timing loops in your old 68HC11 code.

I/O Ports. The digital I/O ports on the HCS12 are more flexible than ever. Besides selecting the direction of each port pin via a Data Direction Register, there are registers controlling output drive level (standard and reduced), internal pullup and pulldown resistors, and output logic polarity (ie. true or inverted logic).

COP Watchdog. On most flavours of HC11, this could be enabled via a bit in the non-volatile CONFIG register. On the HC12, it is dynamic, and automatically enabled following reset. Therefore you have to choose whether you're going to service it, or disable it.

Write-Once Registers. On the HCS12, there is no 64-cycle startup window in which you have to write all the protected registers. Instead, the HC12 implements a WriteOnce rule on sensitive registers. What this means is that, following reset, you have one chance to write them, then they become "Read Only". The advantage of this is that you have more control of when you alter these register values. To take advantage of this safeguard, you should initialize all the registers that are crucial, even if the default values are what you want. That way, if your code runs amok, or there are any glitches which try to change register values, they will be protected.

There are many more differences, and you should make sure to read through the Freescale App Note (AN1284) that details the new instructions and addressing modes of the 68HC12, explaining differences from the 68HC11.

2.7 MIGRATING FROM THE 68HC912

You gain a lot more speed, memory, and flexibility, but you have a lot more registers to think about, and many of their addresses have changed. Gone are the Vfp generator and flash voltage switch, since the new flash technology uses 5V, and has built-in self-timed algorithms for program and erase functions. But your s-records must contain an even number of bytes, and begin on an even address boundary if you're going to "burn" them into flash. Some assemblers will generate this format for you but others (such as AS12) don't. In the latter case, you'll need to use the Freescale utility called SRECCVT.

3 HARDWARE DESIGN FEATURES

3.1 3-VOLT OPERATION

One of the nice features of the MCU is that it can operate on 3 to 5 V, while maintaining full bus speed capability. To support 3 Volt operation, the Docking Module incorporates an adjustable regulator whose output voltage is set by a resistive voltage divider. The circuit has been designed such that simply inserting a shorting jumper causes the regulator's output to shift from 5V to 3.3V. (Note: If you aren't using a Docking Module (e.g. you're just plugging the DIP module into a breadboard), you'll need to supply your own 3 Volt regulated supply to the Vcc pin to take advantage of 3-Volt mode.) When operated at 3V, there are a few precautions that should be noted, however.

The logic pins are not 5V-tolerant, so you will need to take the necessary steps to prevent damage to the I/O pins of the MCU. Also, the maximum VRH voltage is limited to 3.3V, so any external voltage or precision voltage reference you supply for use with the Analog-to-Digital converter subsystem should be scaled accordingly. One last point is that some BDM pods (eg. MicroBDM12SX) will not work with 3V targets, so you should check the specs of the BDM pod you intend to use. A good choice is MicroBDM12LX, which works with both 3V and 5V targets.

3.2 RESET

Unlike previous HC11 and HC12 designs, the 9S12C MCU has an on-chip low-voltage inhibit (LVI) reset circuit, so it is not necessary to provide such a circuit externally. A momentary tact switch is provided for manual reset, and the LVI circuit will provide a clean reset signal upon power-up.

3.3 ABOUT THE DOCKING MODULE VOLTAGE REGULATOR

The NanoCore12 Docking Module (also, the School Board) includes an LM1086CT-ADJ voltage regulator. Housed in a TO-220 package, it is capable of dissipating about 500 mW at room temperature. Other nice features are: reasonably low quiescent current, and low dropout voltage-- it will work with an input voltage down to about 5 Volts (or 3 Volts, in 3-Volt Mode), making it quite well-suited to battery operation. It is also designed to withstand reverse polarity. One drawback, however, is that it can become unstable and start to oscillate at low temperatures, especially if the input voltage source is connected to J1 via long wires. If low-temperature operation is anticipated, the on-board 10uF tantalum capacitor can be replaced with a higher value (47uF or 100uF). To compensate for long lead-in wires, add capacitance of about 100uF at, or close to, the J1 connector.

3.4 ABOUT THE ON-BOARD VOLTAGE REGULATOR

NanoCore12 modules include a tiny LP2981AIM-5 voltage regulator mounted beneath the module. It is a low-dropout 5V regulator capable of supplying the on-board circuitry with the required current, with as much as 50 mA to spare for user applications, if the supplied input voltage to the regulator is limited to 6V. The higher the input voltage, the less current will be available for the user application before thermal shutdown occurs.

3.5 PLL

While the supplied crystal is only 8MHz, the MCU is capable of running at a much higher speed. The phase-locked loop feature of the MCU allows you to boost the bus speed by an integer multiple of the crystal frequency, so by enabling the PLL, you can actually run the MCU at 24Mhz.

4 WRITING SOFTWARE

4.1 IMPACT OF THE SERIAL MONTIOR

When you are working without a BDM pod, the Serial Monitor program is the only method available to load and erase flash. It is in a protected block of flash, so there's no way to accidentally erase it. There are two modes, controlled by switch SW2: Run and Load. The monitor mode is determined immediately following reset by checking the position of switch SW2. When working with the monitor program in place, there are a few points to be noted:

1) while the user vectors are implemented by the monitor at 0xF780 to 0xF7FF, you don't really have to worry about it, because the monitor program will automatically adjust them when your s-record is loaded.

2) the monitor relocates RAM to the address range 0x3800 to 0x4000 from the default location after MCU reset of 0x0000 to 0x07FF. (On NanoCore12MAXC128, the RAM is remapped to the range 0x3000 to 0x4000).

3) the monitor program enables the phase-locked loop (PLL), so the module runs at 24Mhz instead of the MCU's default startup speed of 4Mhz.

4) the user code must clear the CCR I-Bit (e.g. via the CLI instruction in assembler or the INTR_ON() directive (in ICC12).

5) the SCI cannot be used by the user program when in LOAD mode, since it is dedicated to the monitor program.

6) COP cannot be disabled in LOAD mode.

4.2 WRITING A SIMPLE C PROGRAM IN ICC12

Before starting, you'll need to set up your compiler settings, as follows:
Program Memory = 0x4000.0x7FFF:0xC000.0xFFFF
Data Memory = 0x3800 (use 0x3000 if using a 'C128 module)
Stack Pointer = 0x3FC0

Note that the Data Memory and Stack Pinter addresses shown are valid only for a device with a resident monitor, since the monitor remaps the RAM following reset. If you are writing software for a completely blank chip, and loading it in via a BDM pod, you'll need to change these values to work with the default RAM address range (see the MCU datasheet).

```
    //Flashes Docking Module or School Board LED D2 twice a second

    #include <hcs12c32.h>

    #define DUMMY_ENTRY  (void (*)(void))0xFFFF

    #pragma nonpaged_function _startextern void _start(void);/* entry point in crt12.s */
void main(){
        INTR_ON();              //needed for the SerialMonitor
        DDRT = 0x01;         //Enable LED port (use 0x04 for NC12MAX)

        RTICTL = 0x7F;       //Set RTI divider for 4Hz time base
        CRGFLG |= 0x80;      //Clear the RTI Flag
        CRGINT |= 0x80;      //Enable the RTI
}
    #pragma interrupt_handler rti_handler
        void rti_handler(){
                CRGFLG |= 0x80; // Clear the RTI Flag
                PTT ^= 0x01; //Toggle LED (use 0x04 for NC12MAX)

                INTR_ON(); //Enable Interrupts
}

    #pragma abs_address:0xFFF0

    void (*interrupt_vectors[])(void) =
    {
        rti_handler, /*Real Time Interrupt*/
        DUMMY_ENTRY, /*IRQ*/
        DUMMY_ENTRY, /*XIRQ*/
        DUMMY_ENTRY, /*SWI*/
        DUMMY_ENTRY, /*Unimplemented Instruction Trap*/
        DUMMY_ENTRY, /*COP failure reset*/
        DUMMY_ENTRY, /*Clock monitor fail reset*/
        _start, /*Reset*/
    };
```

4.3 OTHER ISSUES WITH ICC12

Because the register addresses have changed from what they were in HC12, meaning the header file is different for the 9S12C32, some library files in ICC12 will need to be re-compiled, using the new header file, if you want to use them. Of course, if you're not using library functions, or you are using functions that don't involve registers, then there won't be a problem with the existing versions. The modified functions are included on the CD-ROM to get you started.

To use the SCI, make sure to include ***C32_iochar.c*** and ***C32_serial.c***. Also, you'll need the complete vector file for the C32, which is called ***C32_vectors.c***. Unzip ***C32_C.zip*** and place ***hcs12c32.h*** in ***c:\icc\include\*** (or in your equivalent path). Make sure to place ***C32_Vectors.c*** in the same folder as your project, and add it to you project via the "add file menu item". (By the way, the same files work for the C128.)

4.4 "HELLO WORLD" PROGRAM

First of all, create a new project from the Project menu.

Then create a new file and save it as HelloWorld.c Add it to the Project by right clicking in the Project Panel and using Add Files to add it to the Files section.

Next type in the following code:

```
#define _SCI
#include <hcs12c32.h>

#pragma nonpaged_function _start
```

```
extern void _start(void);  /* entry point in crt12.s */
extern int _textmode;
int putchar(char c)
     {
     if (_textmode && c == '\n')
             putchar('\r');
     while ((SC0SR1 & TDRE) == 0)
             ;
     SC0DRL = c;
     return c;
     }

void main(){
     INTR_ON();             //need to enable interrupts for the SerialMonitor
     SCI0BD = 26; //9600 Baud
     SCI0CR2 = 0x0C; /* enable transmitter and receiver */

     puts("Hello, World!");
}
#pragma abs_address:0xFFFE
void (*interrupt_vectors[])(void) =
{
     _start, /*Reset*/
};
```

Since **puts** calls **putchar**, we define it before invoking it in **main**. **Main** has an implicit **_Start** entry point, which is called after the setup by **CRT12.o**, which is a module that the ICC12 linker links in as the starting point of the program. Besides initializing the stack and other system features, it initializes memory, initialized variables and constants before transferring control to the Main.

Compile and link the program, fixing any syntax errors that may have cropped up. Ensure that the Project Options | Device Configuration drop down box points to the 9S12C32 Flash Mode. This sets the link address to start the code section at 0x4000 and the stack at top of RAM (0x4000).

4.5 USING A BDM POD

If you have a BDM pod, you can erase the resident monitor program completely. This will free up all the MCU resources for your program (most importantly, the SCI). Without the monitor in place, the RAM will be at the default location following reset, so make sure to use the correct compiler/linker settings. Also, the PLL won't be enabled, so the bus speed will be 4 MHz.

4.6 AUTOMATING S-RECORD CONVERSION IN ICC12

You may have to convert the s-record file to get it into the proper format for your BDM pod to load correctly. ICC12 has a nice feature at *Project->Options->Compiler->ExecuteCommandAfterBuild* where you can add the SRECCVT command mentioned earlier. Use the following command line:

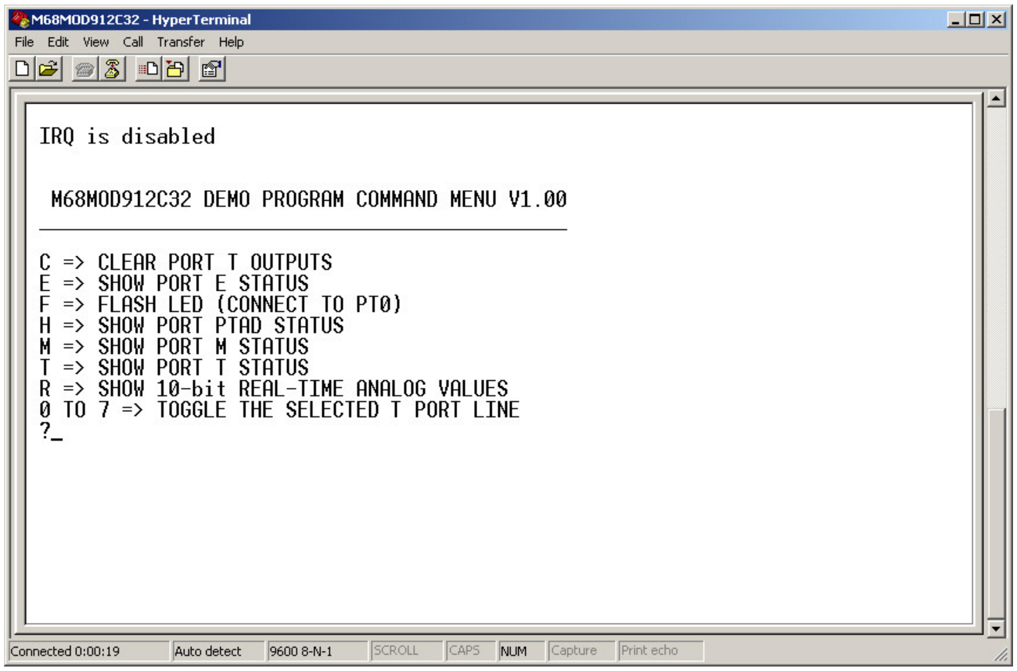
sreccvt.exe -m C0000 FFFFF 32 -lp -o *outfile*.s2 *infile*.s19

where *infile* refers to the file you are converting and *outfile* is the result.

5 TUTORIAL

NanoCore modules are shipped from the factory with a Demo program loaded into its memory. This program is useful for familiarizing yourself with the hardware features of the MCU. It was written in assembler, and the sourcecode can be found on our Product CD or online our Support Library, if you would like to examine or modify it.

If you've already erased the demo program, you may use uBug12 to re-load it. Then follow the relevant steps in the QuickStart Guide to open HyperTerminal and achieve the display shown below.



```
IRQ is disabled

 M68MOD912C32 DEMO PROGRAM COMMAND MENU V1.00
_____________________________________________
C => CLEAR PORT T OUTPUTS
E => SHOW PORT E STATUS
F => FLASH LED (CONNECT TO PT0)
H => SHOW PORT PTAD STATUS
M => SHOW PORT M STATUS
T => SHOW PORT T STATUS
R => SHOW 10-bit REAL-TIME ANALOG VALUES
0 TO 7 => TOGGLE THE SELECTED T PORT LINE
?_
```

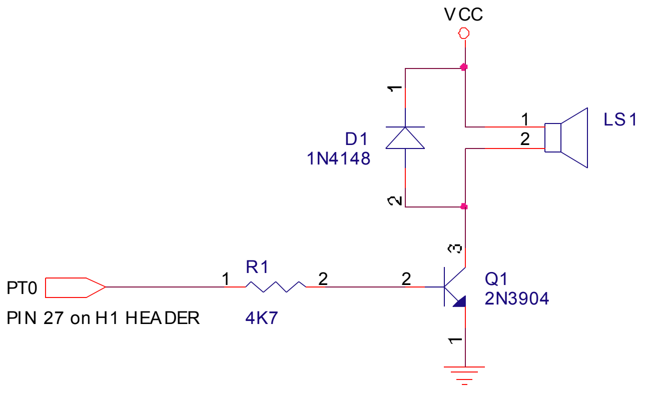
5.1 Executing Demo Commands

- •Type the digit 0 repeatedly and observe that LED D2 toggles on and off.
- •Type the digit 1 repeatedly and observe that LED D3 toggles on and off.
- •Type the letter F to cause LED D2 to flash twice.

Typing E, H, M, or T will cause a hex number to be displayed, whose bit values represent the logic levels of the respective pins. If the pins are unconnected to external circuitry, they will assume a logic 1 level, due to internal pullup resistors.

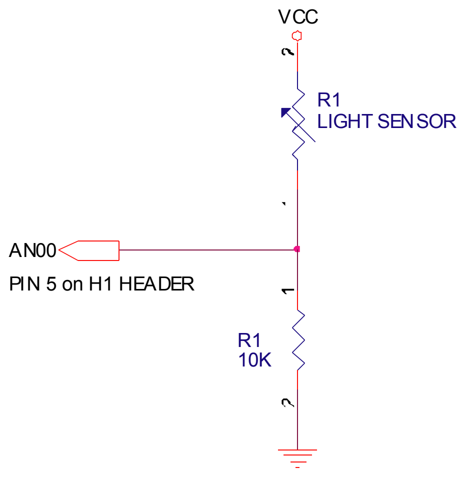
5.2 A Simple Audio Circuit

Connect a simple audio transducer circuit, as shown, and type the letter F. LED D2 will flash twice, and two audible "beep" tones will be emitted from the transducer. If you press and release the Reset button on the Docking module, LED D2 will flash four times, and you'll hear four "beep" sounds from the transducer.



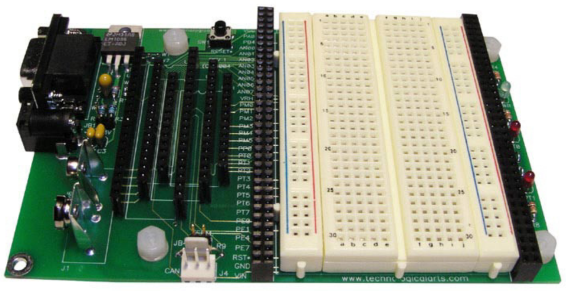
5.3 A Simple Analog Circuit

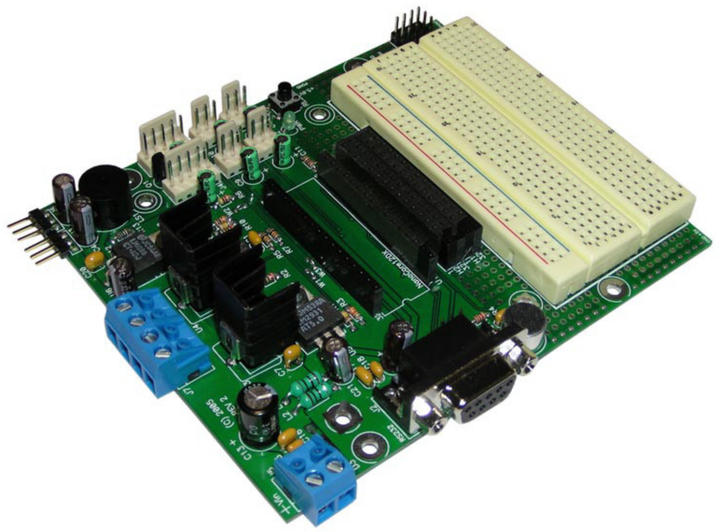
Connect a CdS light sensitive resistor (photocell) in the configuration shown. Then type the letter R to see the value of AN00 change as you vary the amount of light reaching the photocell. The value displayed is a 10-bit representation of the voltage level on the AN00 pin, with fullscale being approximately 5V (represented by the hexadecimal number 3FF).



6 GOING FURTHER

If you'd like to get started interfacing common electronic devices such as LEDs, switches, relays, etc., you may consider purchasing the optional NanoCore12 School Board (#NC12SB, shown). The School Board has socket strips to accomodate all three NanoCore12 module sizes (i.e. 24-pin, 32-pin, and 40-pin), and provides a solderless breadboard section, a reset button, a couple of user LEDs, a 5V/3V regulator (jumper selectable), and 9-pin serial connector. 9-V battery snaps are provided with robotics applications in mind. Mounting hole locations are standardized so as to line up with most hobby robot bases currently on the market.





If you're into robotics (or other mechatronics applications) you may be interested in the Servo/Sensor/Motor Interface board for NanoCore12DX (#NC12DXSSMI). It is similar to the School Board in that it includes a solderless breadboard, voltage regulator, and serial port connector. However, it also includes connectors for up to six hobby-style R/C servos, two Devantech sonar distance-measuring sensors (#SRF04), and up to four analog sensors (e.g. Sharp IR distance-measuring sensor GP2D120). But perhaps the highlight of the board is the set of two high-current H-bridges (#TLE5206) for driving two DC motors (up to 4 Amps and 24V). The H-bridge drivers are connected to the NanoCore12's PWM pins, for easy speed control implementation. Additional features on the board are a microphone and an audio transducer. An op amp circuit adds battery-voltage monitoring capability, via one of the NanoCore12's analog input pins.

New application cards are being developed from time to time. Check the subcategory called Application Cards on the NanoCore12 webpage to browse the currently available selection.

APPENDIX A - SERIAL MONITOR

INTRODUCTION

This appendix describes the Freescale 2 Kbyte monitor program for the HC9S12 series MCU. This program supports 23 primitive debug commands to allow FLASH / EEPROM programming and debug through an RS232 serial interface to a personal computer. These include commands to reset the target MCU, read or modify memory (including FLASH /EEPROM memory), read or modify CPU registers, go, halt, or trace single instructions. In order to allow a user to specify the address of each interrupt service routine, this monitor redirects interrupt vectors to an unprotected portion of FLASH just below the protected monitor program. This monitor is intended to be device unspecific, this single application with very slight modification should execute on any HC9S12 derivative. A user on a tight budget can evaluate the MCU by writing programs, programming them into the MCU, then debug using only a serial I/O cable and free software (uBug12) for their personal computer.

This monitor does not use any RAM other than the stack itself. The COP watchdog is utilized for a cold reset function; user code should not disable the COP (ie. by writing 0x00 to COPCTL). This development environment assumes you reset to the monitor when you are going to perform debug operations. If your code takes control directly from reset, and then an SCI0 interrupt or a SWI attempts to enter the monitor, the monitor may not function because SCI0, the phase locked loop (PLL), and memory initialization registers may not be initialized as they would be for a cold reset into the monitor. There is no error handling for the PLL. If the frequency source is missing or broken, the monitor will not function. The monitor sets the operating speed of the MCU to 24 MHz. Modification of the MCU speed by the user with out considerations for the monitor program will render the monitor nonfunctional. If the PLL loses lock during operation, the monitor will fail.

BLOCK PROTECTION

In order to prevent accidental changes to the monitor program itself, the 2 Kbyte block of FLASH memory where it resides ($F800-$FFFF), is block protected. Additionally all write commands are restricted from modifying the monitor memory space. The only way to change the contents of this protected block is to use a BDM-based development. In the lowest cost applications where the monitor is used with an SCI serial interface to the RS232 serial port on a personal computer, there is no way to accidentally erase or modify the monitor software.

COP CONFIGURATION

The monitor as written creates hard reset function by using the COP watchdog timer. It does so by enabling the COP and waiting for a COP timeout reset to occur. If the user application uses the COP two issues must be considered.

•If the COP is disabled in the user application, the monitor will be unable to perform a hard reset and will soft reset to the start of the monitor instead.

•The monitor does not service the COP timer. If the user application implements COP timer servicing, upon re-entry into the monitor a hard reset is likely to occur.

MEMORY CONFIGURATION

1) Register space is $0000-$03FF.

2) Flash memory is any address greater than $4000. All paged addresses are assumed to be Flash memory.

3) RAM ends at $3FFF and builds down to the limit of the device’s available RAM.

4) External devices attached to the multiplexed external bus interface are not supported.

SERIAL PORT USAGE

In order for this monitor to function the SCI0 serial interface is used. It is assumed that the monitor has exclusive use of this interface. User application code should not implement communications on this serial channel. This monitor accommodates RS232 serial communications through SCI0 at 115.2 kbaud. For applications requiring the use of SCI0, you should purchase a BDM pod which allows for more advanced debugging.

VECTOR REDIRECTION AND INTERRUPT USE

Access to the user vectors is accomplished via a jump table located within the monitor memory space. This table points all interrupt sources to a duplicate vector table located just below the monitor. ($F780-$F7FE). The monitor will automatically redirect vector programming operations to these user vectors. The user's code should therefore continue to implement the normal (non-monitor) vector locations ($FF80-$FFFE). If execution of an interrupt with an un-programmed vector is attempted, behavior is undefined. For this reason, the user is strongly encouraged to implement a software trace for all vectors, as is good programming practice. The monitor depends on interrupts being available for monitor re-entry after GO or TRACE commands. Therefore, it is important that the user application executes with interrupts enabled.

APPENDIX B - UBUG12 COMMAND LIST

------------------------------------- **REGISTERS** --

RD		- Register Display
RM	<RegisterName> <Data8/16>	- Register Modify
CCR	<Data8>	- Set CCR register
D	<Data16>	- Set D register
PC	<Data16>	- Set PC register
PP	<Data8>	- Set PP register
SP	<Data16>	- Set SP register
X	<Data16>	- Set X register
Y	<Data16>	- Set Y register

---------------------------------- **MEMORY MODIFY** ------------------------------------

BF	<StartAdd> <EndAdd> <Data8>	- Block fill
BFW	<StartAdd> <EndAdd> <Data16>	- Block fill word
MD	<StartAdd> [<EndAdd>]	- Memory display
MDW	<StartAdd> [<EndAdd>]	- Memory display word
MM	<Address> <Data8>	- Memory modify byte
MMW	<Address> <Data16>	- Memory modify word

-- **FLASH** ---

FBULK		- Flash bulk erase
FLOAD	[;B][;M]	- Flash load

-------------------------------------- **DEVICE INFO** --------------------------------------

DEVICE		- Get device name

--------------------------------------- **GO/HALT** ---

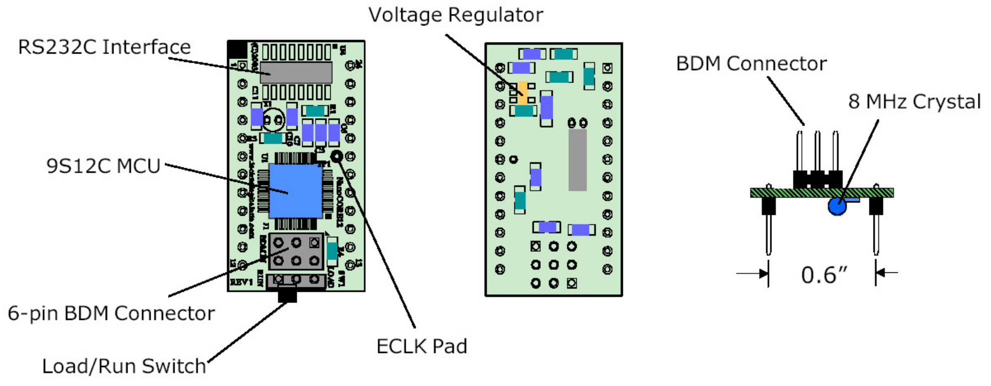
GO	[<StartAddress>]	- Start execution
HALT		- Halt execution
RESET		- Reset target

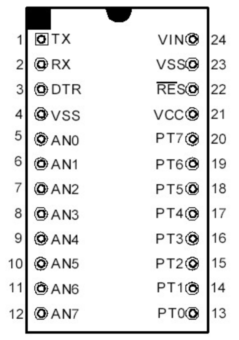
-- **GUI** --

CON	<Comport>	- Connect to target
DISCON		- Disconnect from target
EXIT		- Terminate GUI
HELP		- Display help
OP	<Opacity%>	- Set main GUI opacity

APPENDIX C - COMPONENT PLACEMENT AND PINOUT

NanoCore12™ MCU Module

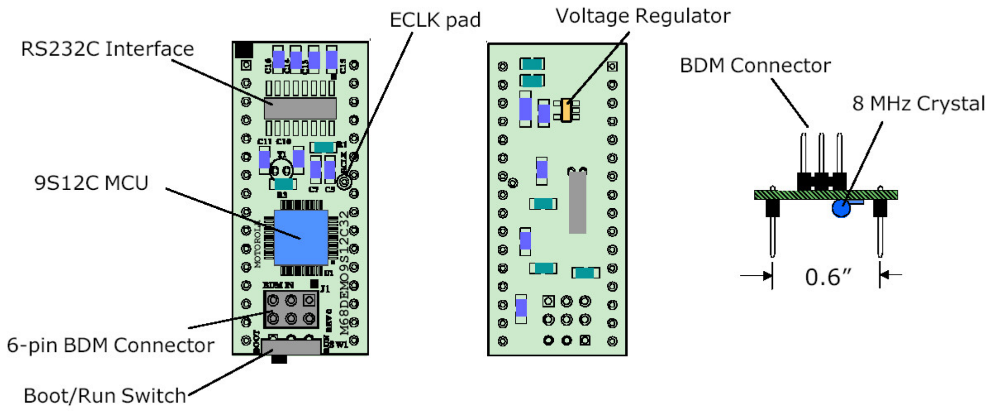


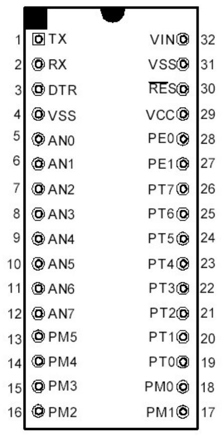


TOP VIEW BOTTOM VIEW END VIEW

PINOUT
(top view)

NanoCore12DX™ MCU Module

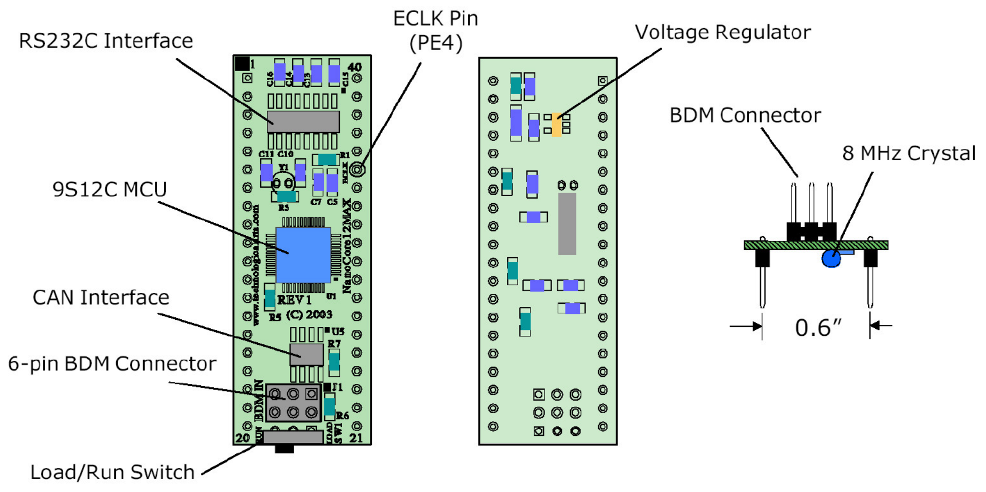


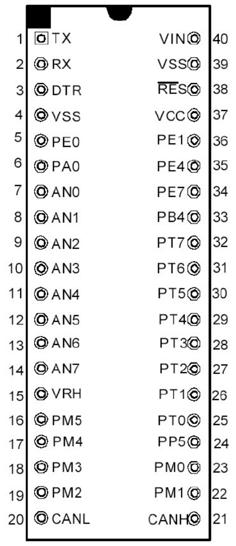


TOP VIEW BOTTOM VIEW END VIEW

PINOUT
(top view)

NanoCore12MAX™ MCU Module





TOP VIEW BOTTOM VIEW END VIEW

PINOUT
(top view)