Table of Contents

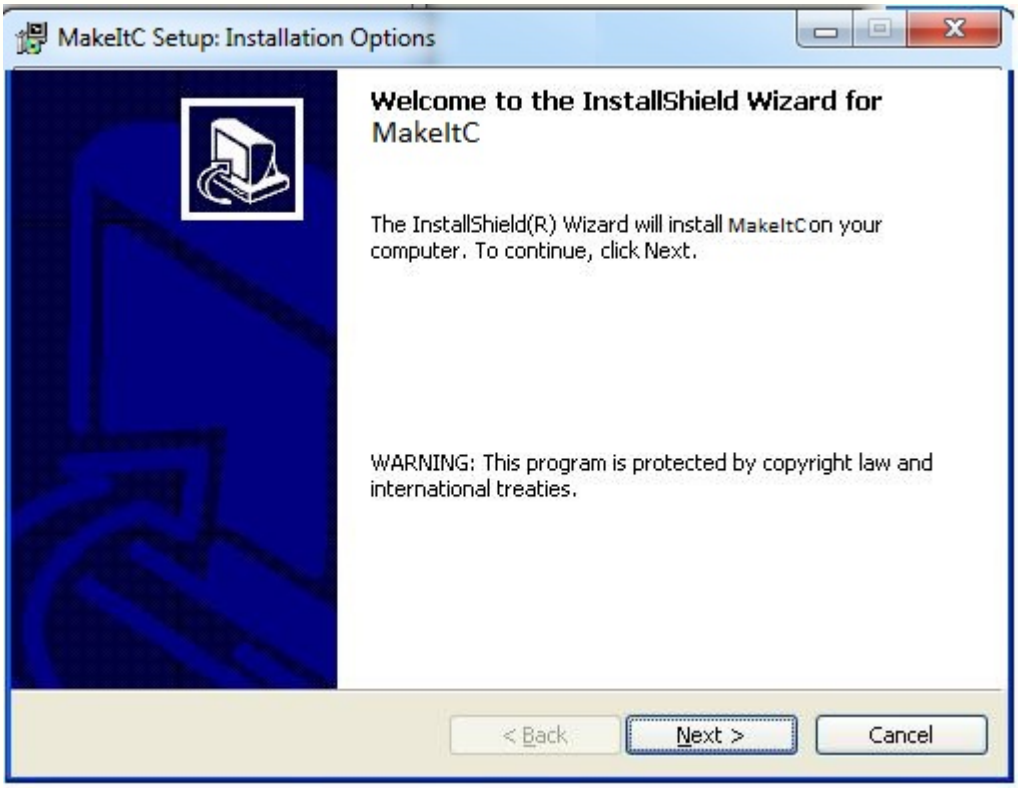
Getting Started

Step 1: Install Software

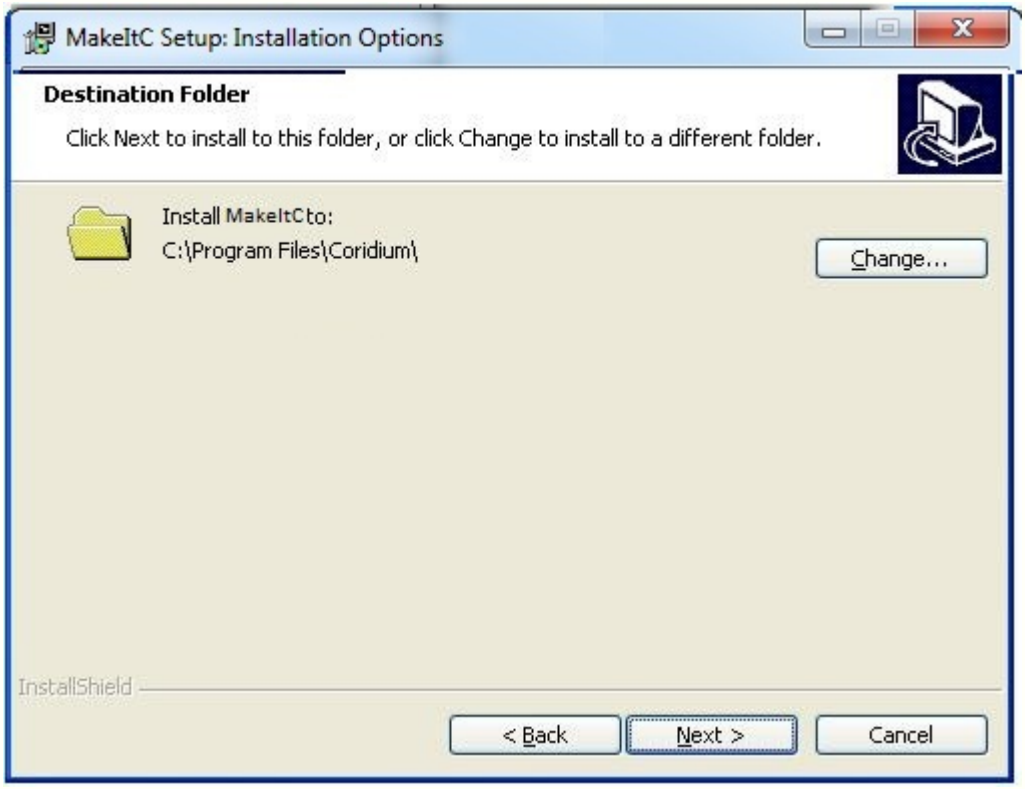
The **ARMexpress or ARMmite** have a built in BASIC Compiler or Library. When the C compiler is used to download a program it will overlay these BASIC components, so you **WILL NOT** be able to run BASIC after downloading a C program.

The GCC compiler runs on the PC and is controlled by the Tcl program **MakeItC**. **MakeItC** launches a text editor of your choice, analyses sources, compiles the C components, links them into a hex file, downloads the hex file to the hardware, and communicates with the ARMexpress or ARMmite with the TclTerm terminal emulation program.

Also, a number of help files and documents about the ARMexpress and ARMmite will be installed on the machine at this time.

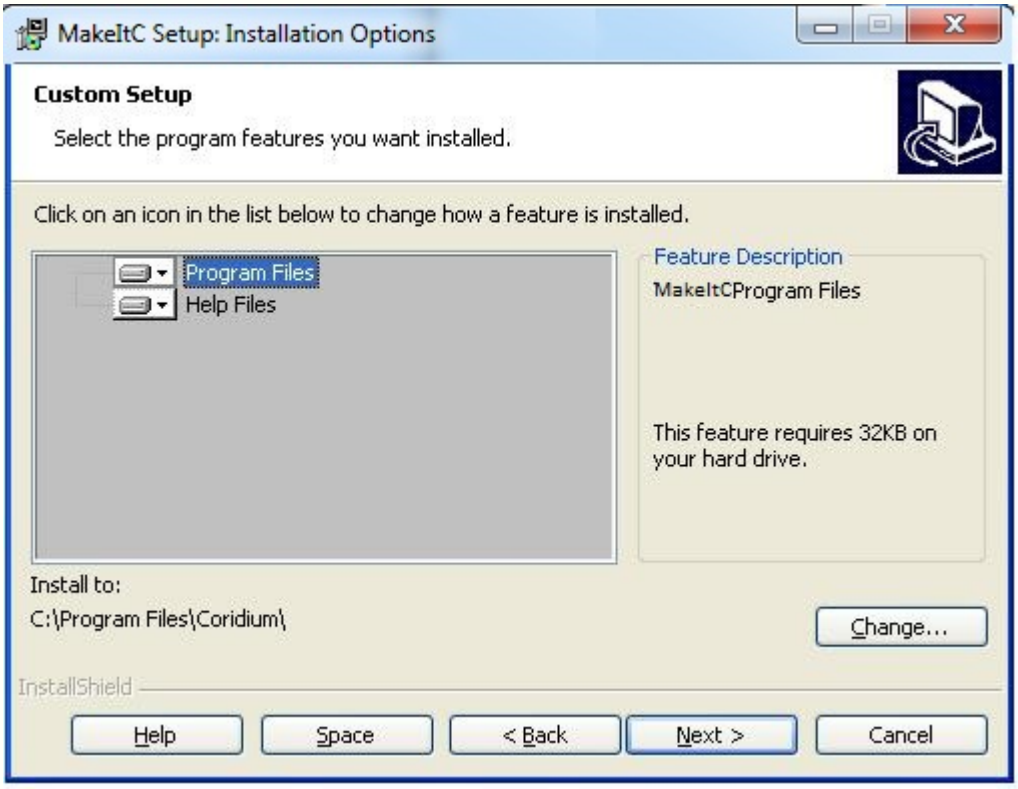


Click next to get started.

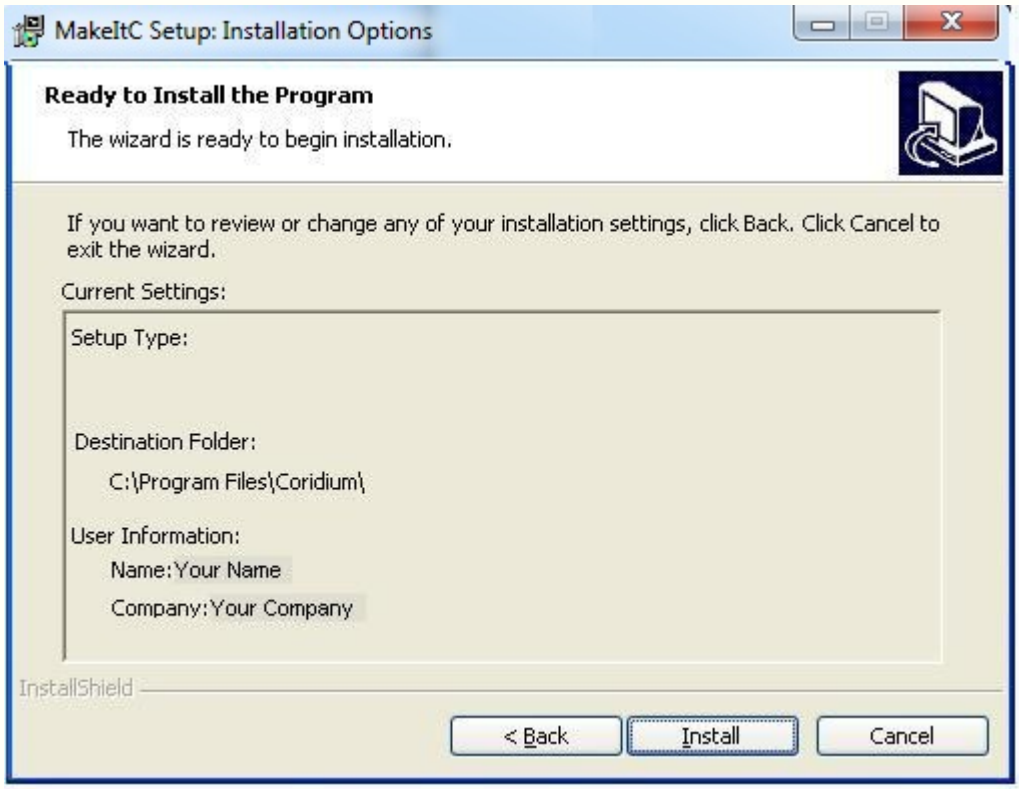


If you wish to change the directory into which MakeIt and its components will be installed you may do so here.

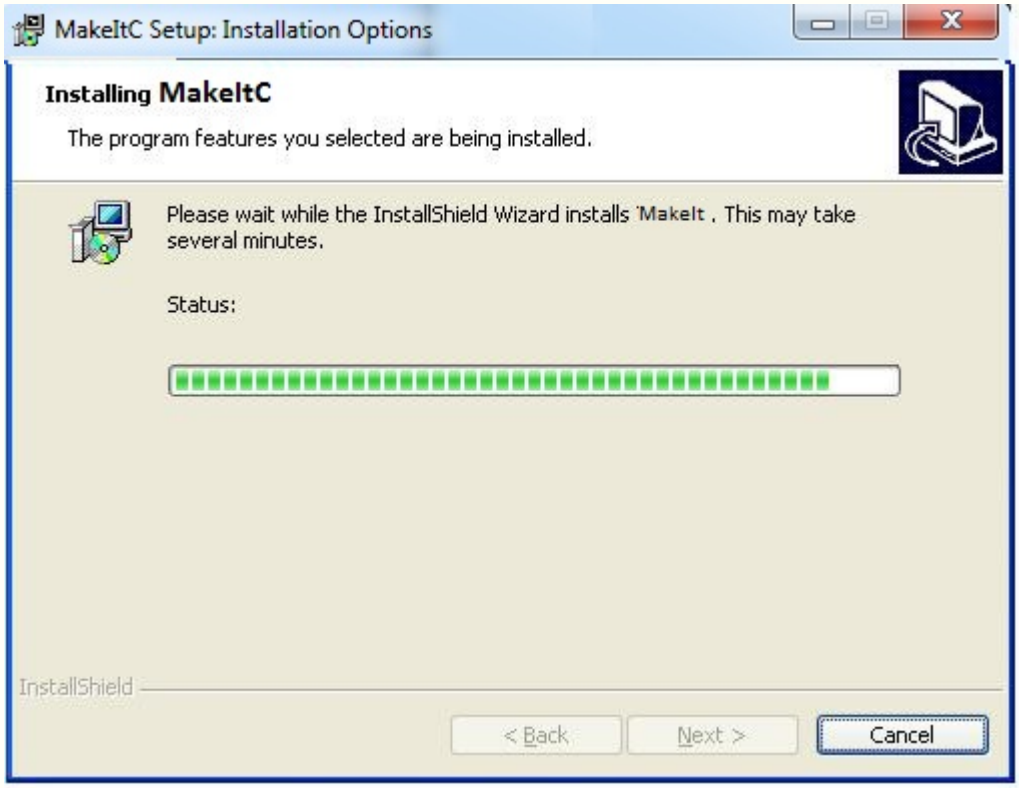
While the installer does give you a choice of where to install the C compiler, this version of Yagarto GCC requires both the gcc AND the sources you want to be compiled to be on the C: drive.



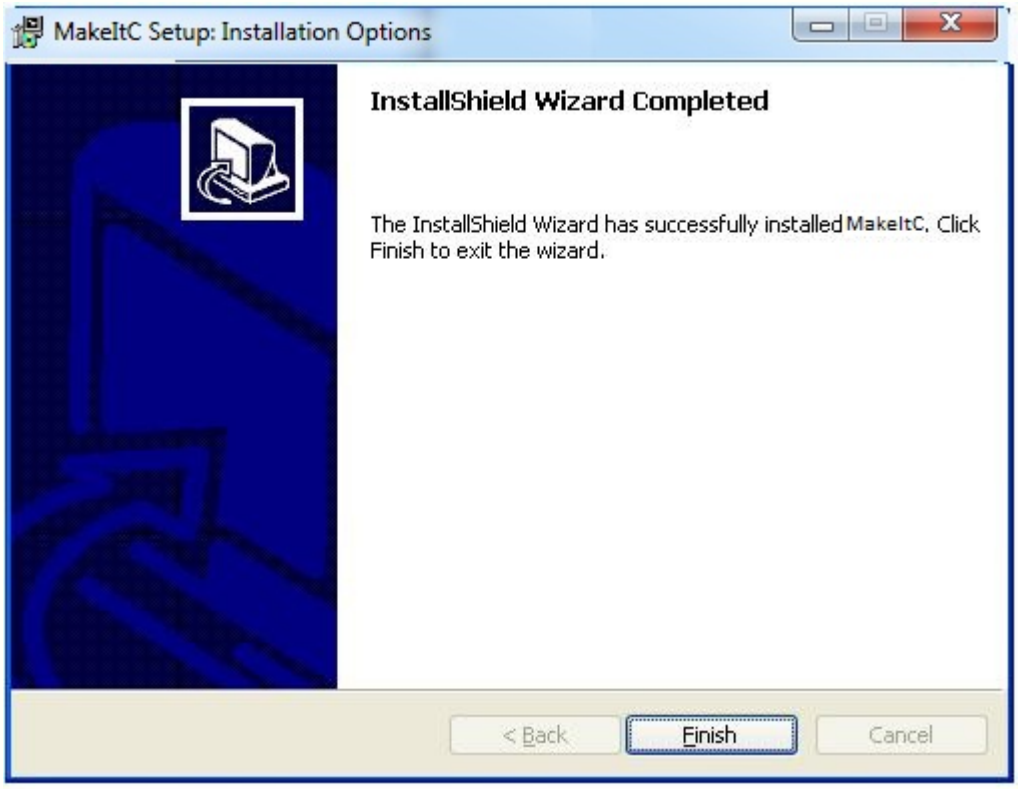
Select next to continue.



Click Install to begin installing files.



One moment please...

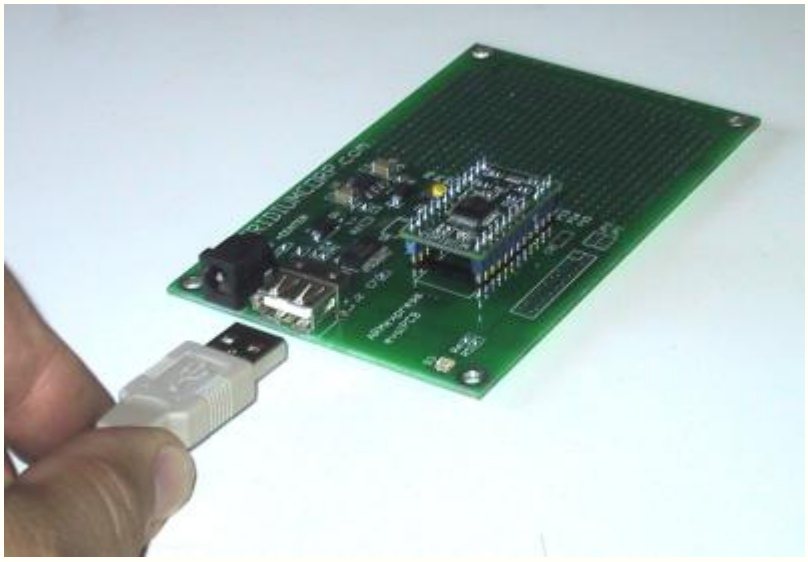


And its as easy as that.

On to Step 2

Step 2: Connect USB

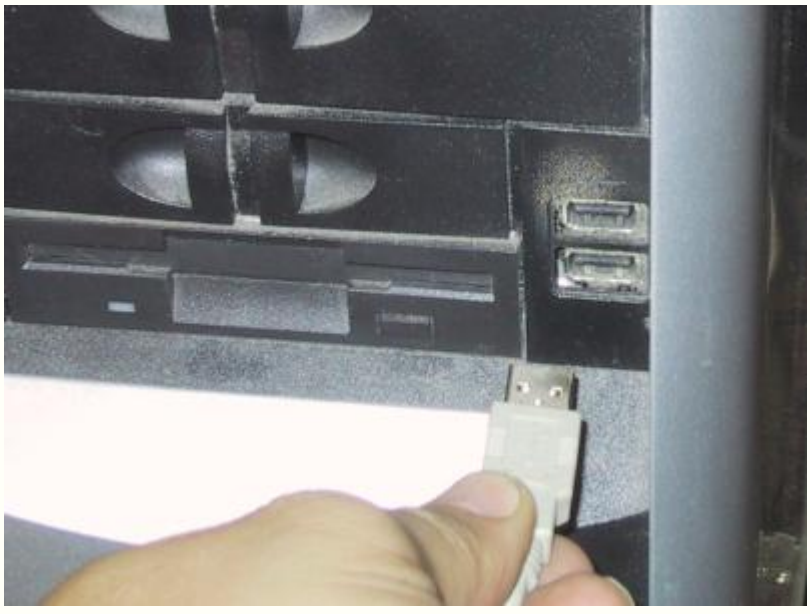
Connect USB Cable to ARMmite/ARMexpress Eval PCB



The **ARMmite** / **ARMexpress** Eval Kit comes with a USB cable. This cable allows you to connect the **ARMmite**/**ARMexpress** directly to a computer equipped with USB. Locate the USB jack on the side of the Eval PCB and plug one end of the USB cable into it. When connected to a PC power is supplied by the PC, the optional power connection is not required, but both may be safely connected.

To connect an ARMmite PRO refer to this page.

Connect USB Cable to Computer



Locate the USB jack on your computer and plug the other end of the cable into it.

Please Consult Installation Guides

Most PC's will sound a tone that indicates a new USB device has been connected. Most Windows Vista and 7 systems will either include the FTDI device driver or are able to download it automatically from the network.

If your system is unable to do that. Run the FTDI driver installation setup in the \Program Files\Coridium\Windows_drivers directory. This will install the proper drivers for the FTDI chips we use for interfacing to the USB.

Up to date details are at the **www.ftdichip.com** VCP drivers page.

From here it should be a simple matter of following the dialog instructions.

Driver Installation Complete, Confirm USB Connection

The Eval PCB or the ARMmite will be powered from the USB bus. It may also be connected to a 7-12V DC power source simultaneously.

To verify connection with the USB and PC the LED on the Eval PCB should light up.

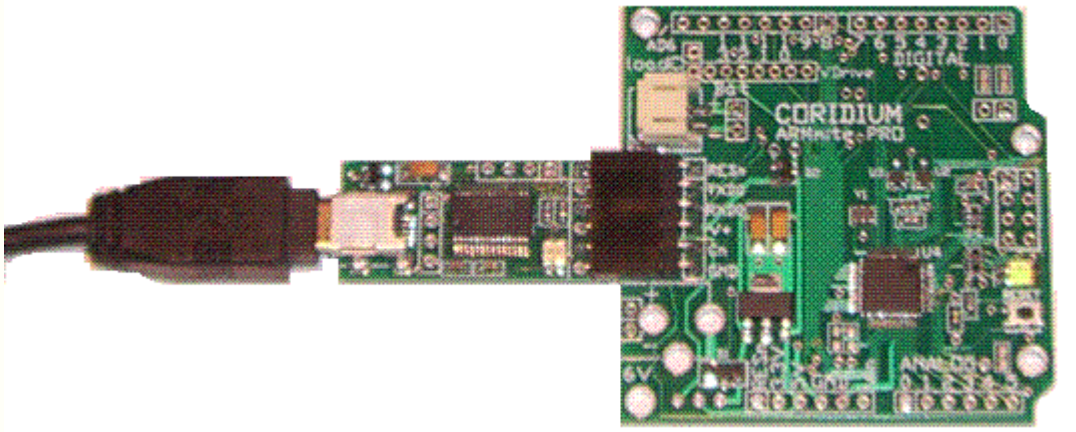
On to Step 3

Step 2: Connect USB on ARMmite PRO

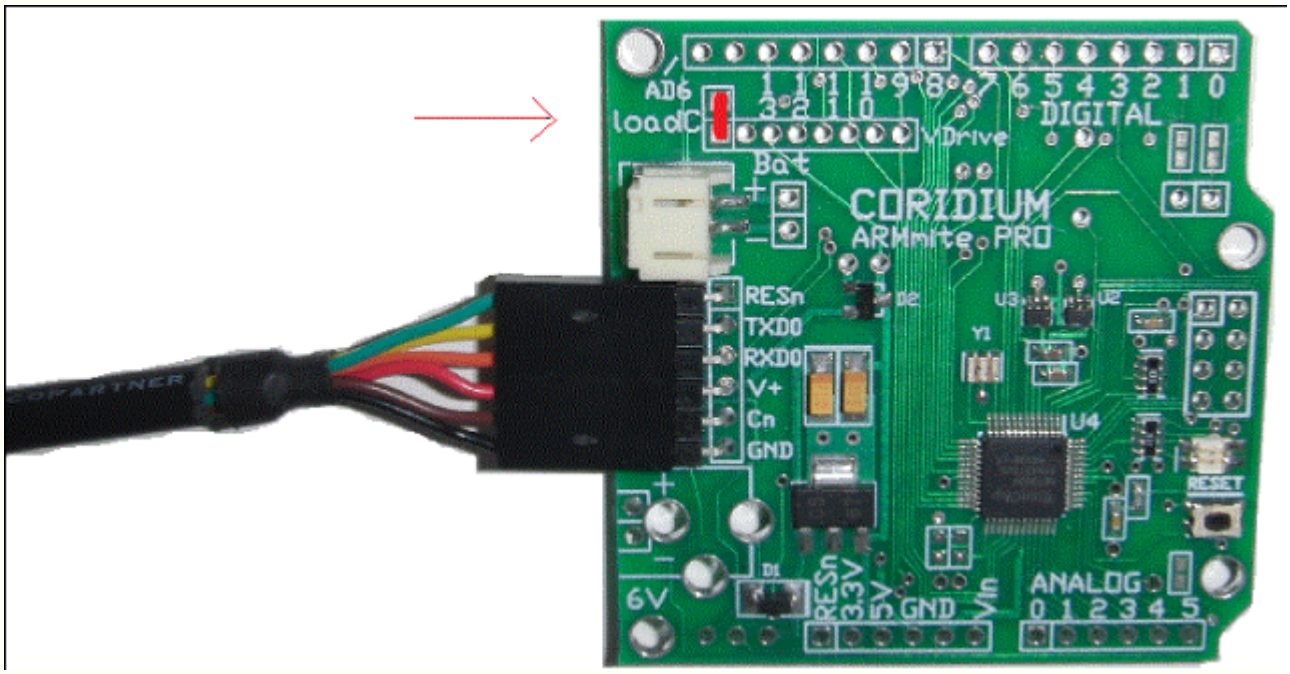
Connect Coridium USB Dongle to ARMmite PRO

The **ARMmite PRO** Eval Kit comes with a USB dongle and cable. This dongle and cable allows you to connect the **ARMmite PRO** directly to a computer equipped with USB. When connected to a PC power is supplied by the PC, the optional power connection is not required, but both may be safely connected.

The USB dongle from Cordium allows you to download C programs **WITHOUT** installing jumpers or pushing buttons.

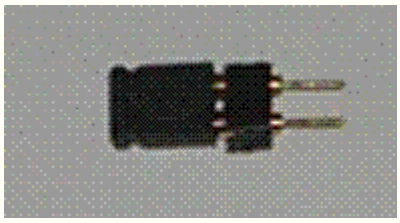


Connect FTDI cable to ARMmite PRO

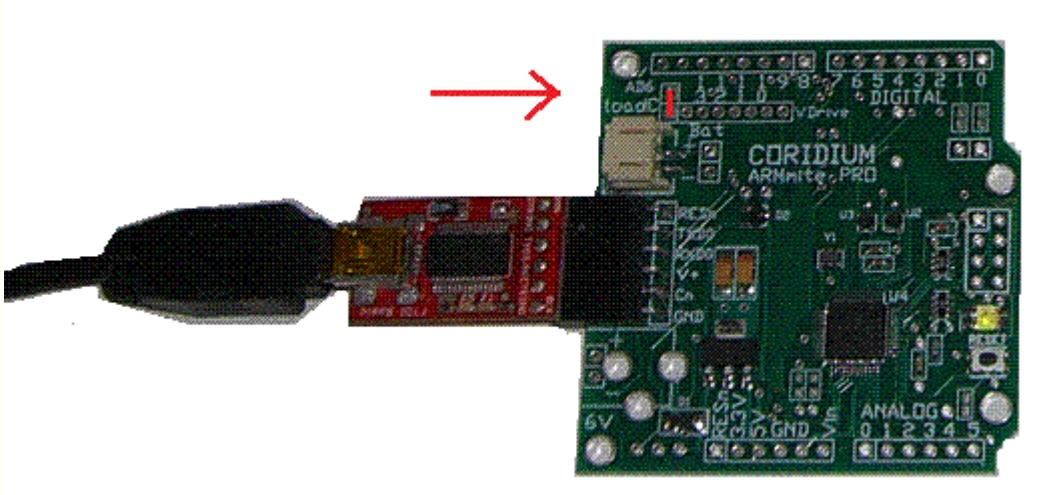


Connect the black wire to GND.

Picture shows the 2 pins that must be jumpered to load a C program, a 2 pin header and a jumper block can be used without soldering it into the PCB (illustrated below)



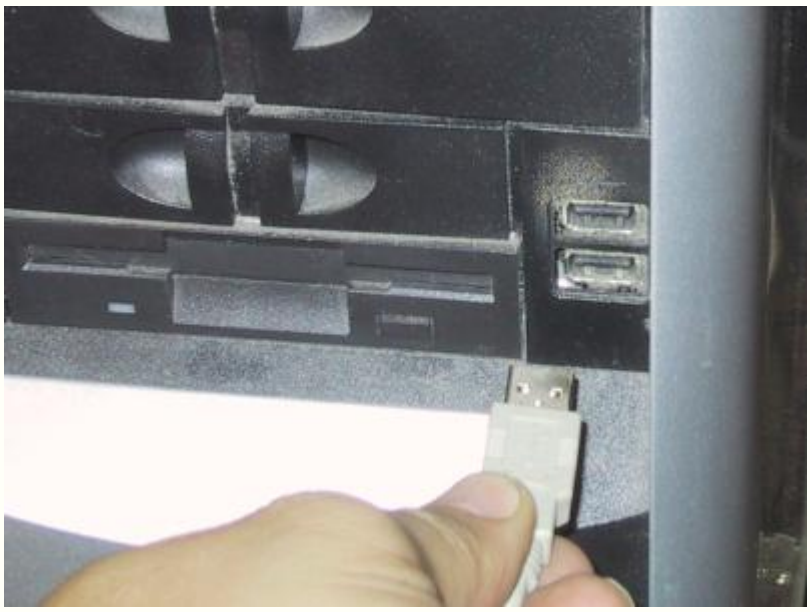
Connect SparkFun USB Dongle to ARMmite PRO



The SparkFun USB Dongle does not control the RTS line so the C jumper must also be installed to download a C program.

Connect USB Cable to Computer

-
-



Locate the USB jack on your computer and plug the other end of the cable into it.

Please Consult Installation Guides

Most PC's will sound a tone that indicates a new USB device has been connected. Most Windows Vista and 7 systems will either include the FTDI device driver or are able to download it automatically from the network.

If your system is unable to do that. Run the FTDI driver installation setup in the \Program Files\Coridium\Windows_drivers directory. This will install the proper drivers for the FTDI chips we use for interfacing to the USB.

Up to date details are at the **www.ftdichip.com** VCP drivers page.

Driver Installation Complete, Confirm USB Connection

The Eval PCB or the ARMmite will be powered from the USB bus. It may also be connected to a 5-12V DC power source simultaneously.

To verify connection with the USB and PC the LED on the Eval PCB should light up.

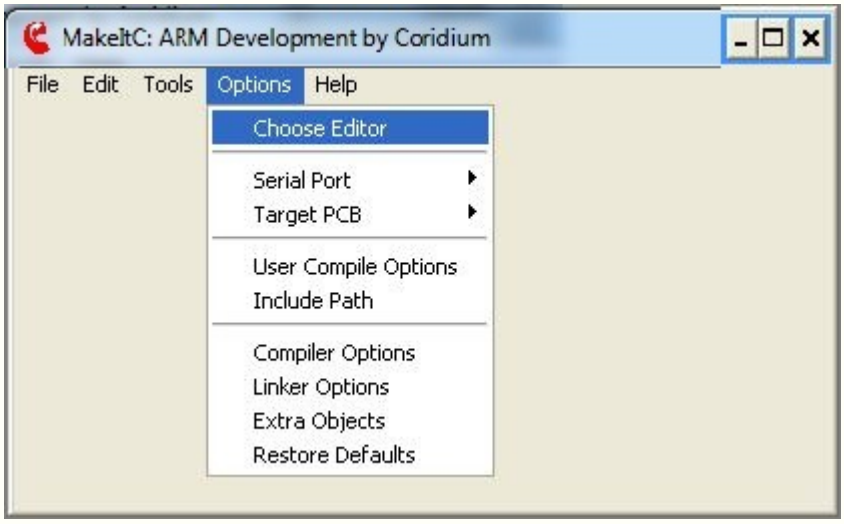
On to Step 3

Step 3: Setting up MakeItC

Setup Your Text Editor

Before you really get started you should probably set up the text editor. We assume you have a favorite one, we like the Crimson Editor these days. We also don't believe our tools should force you to learn a new text editor.

But if you don't choose one, the good old Notepad will be used.

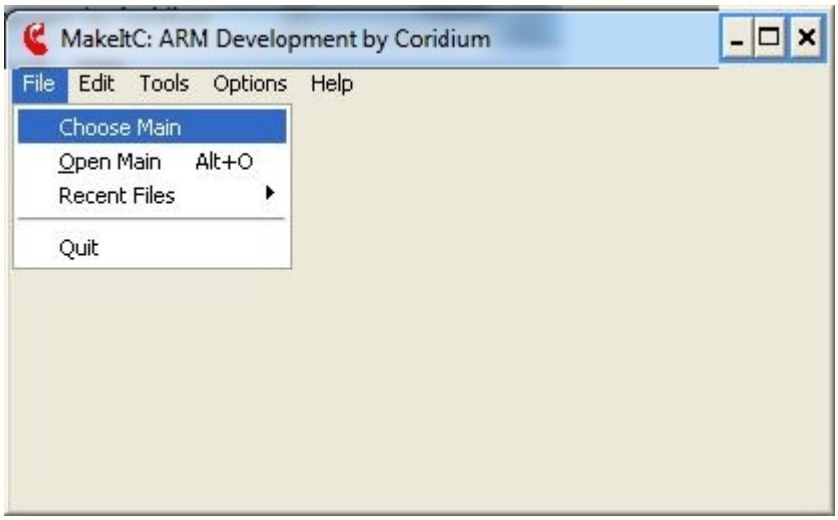


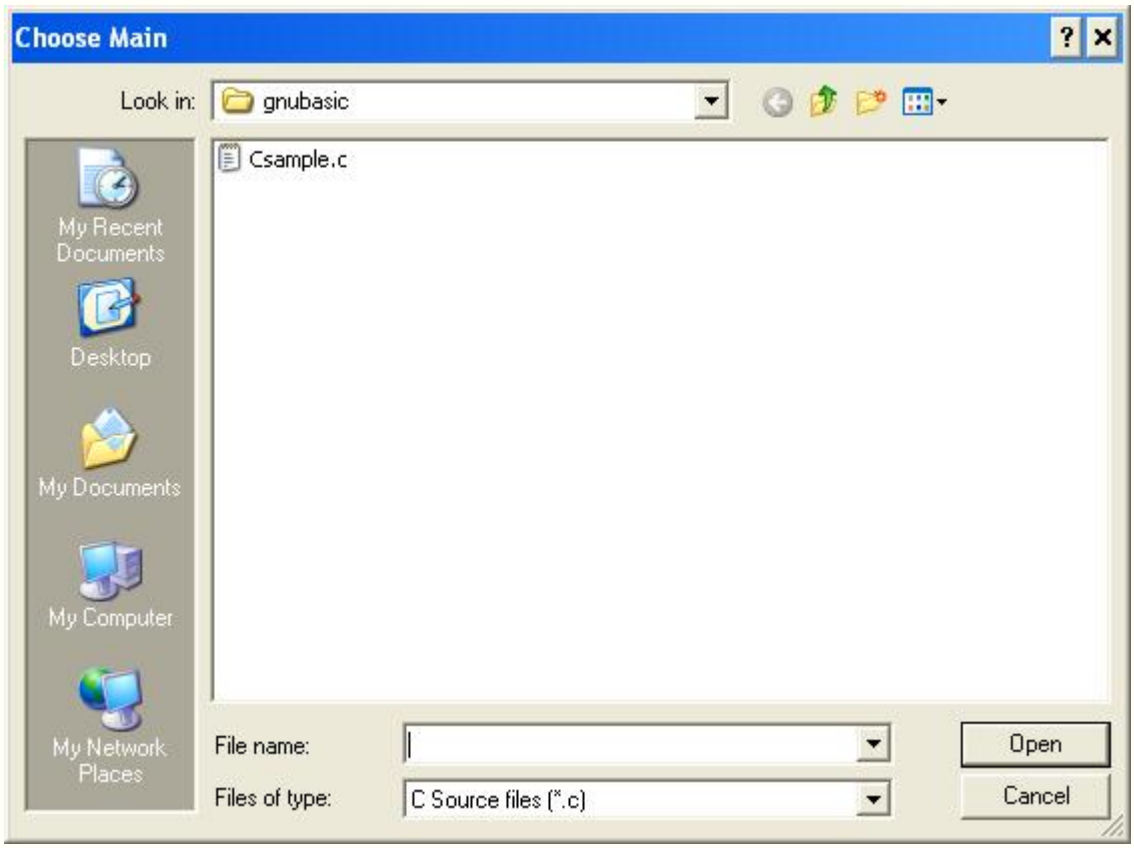
At this point MakeItC will ask to navigate to the .exe or .bat file that launches the text editor.

Choosing the main source file

Now its time to choose a source.c file. This source file will typically contain the main() function.

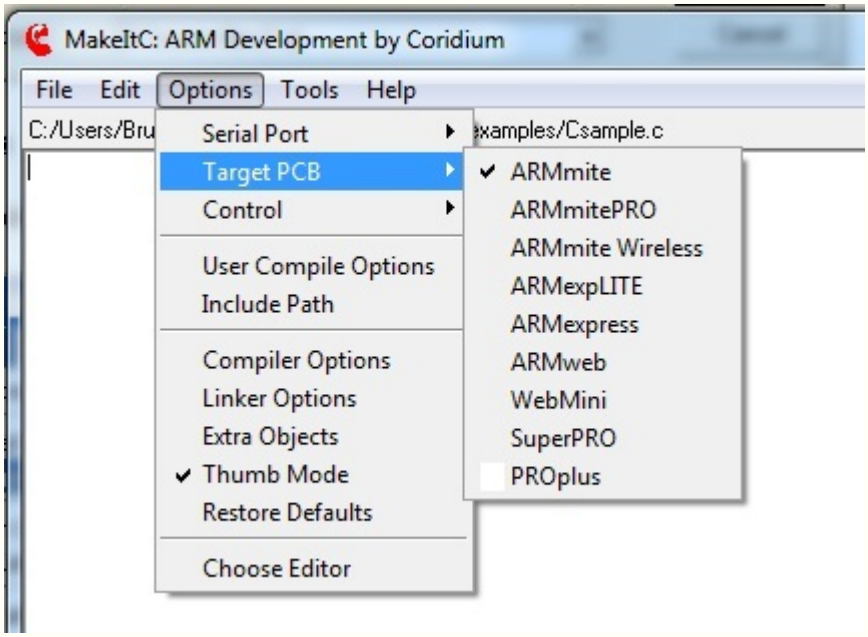
For example you could start with the Csample.c program in the examples directory





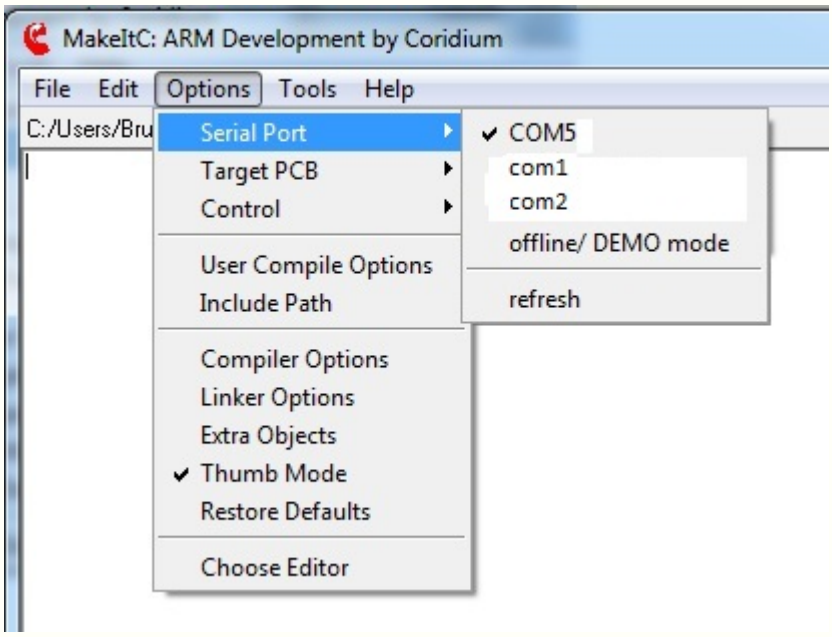
Setting the parameters

Before compiling and loading the program, a couple options need to be set. First is which type of board is to be loaded (ARMmite or ARMexpress).



Then if you want to download and run the program, check the serial port configuration. MakeItC will use the first FTDI-USB device found, or the last port you had assigned.

Ports that are in CAPS are recognized as USB-serial devices like the ARMmite. Ports listed as lower case are not USB-serial devices.

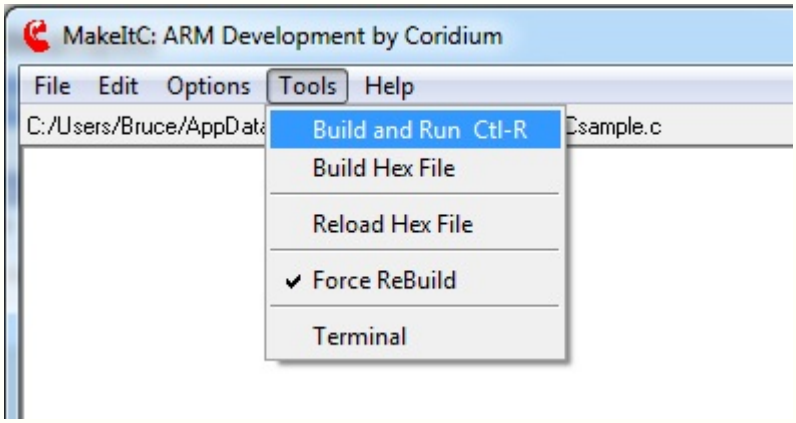


Now the MakeItC can compile and launch the sample program.

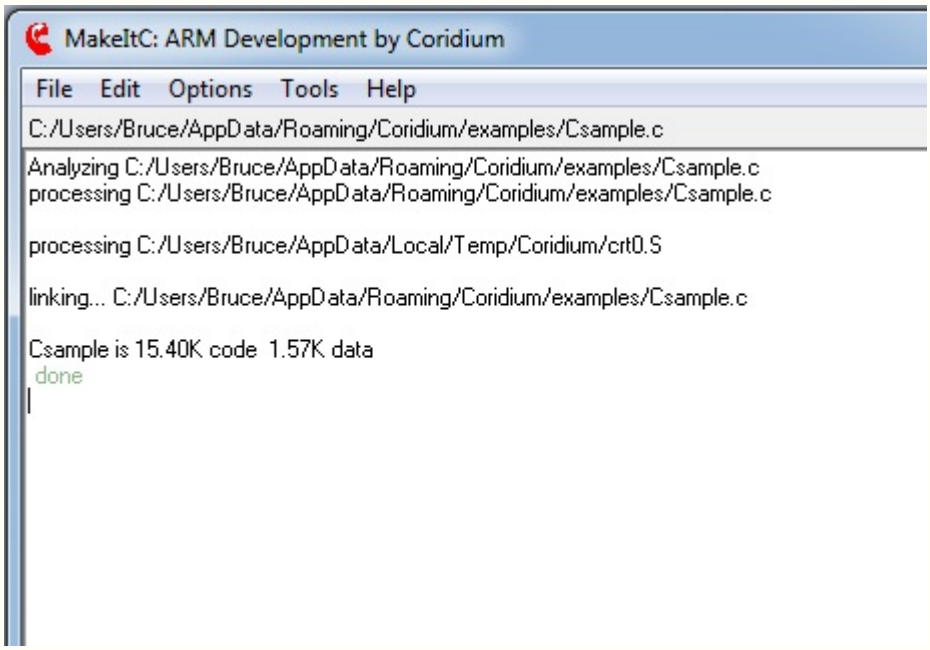
On to Step 4

Step 4: Running the Sample C Program

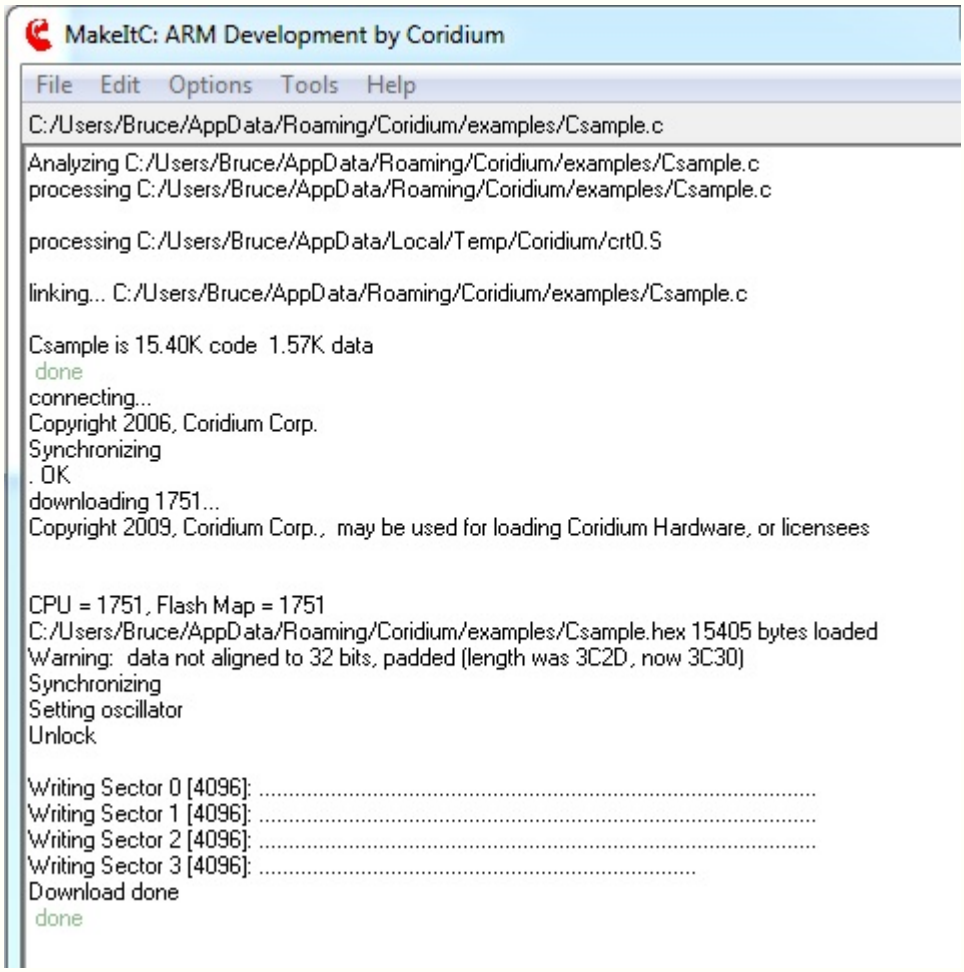
Compile load and run the program



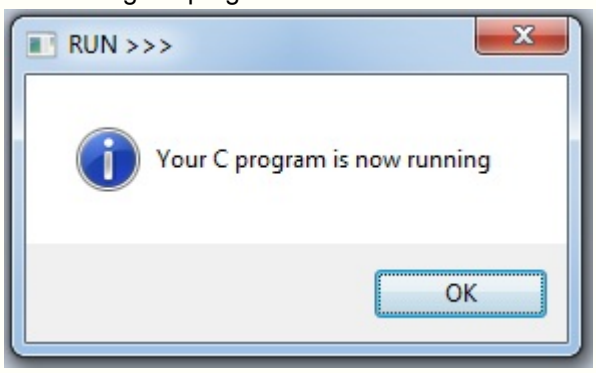
compiling the code



loading the code

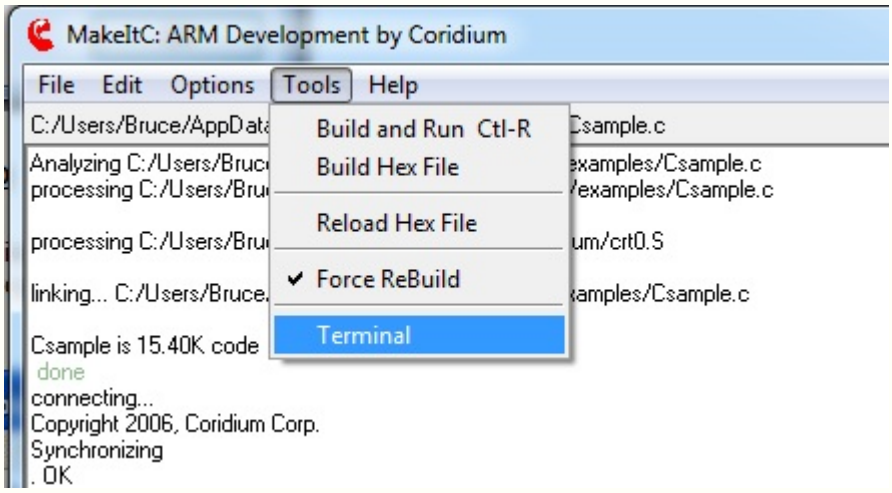


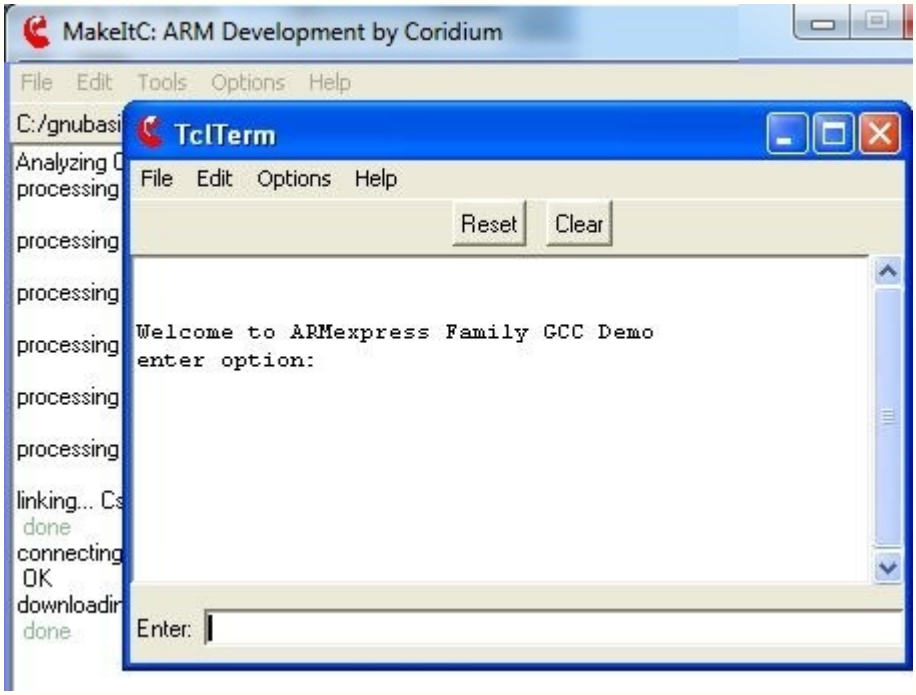
and starting the program



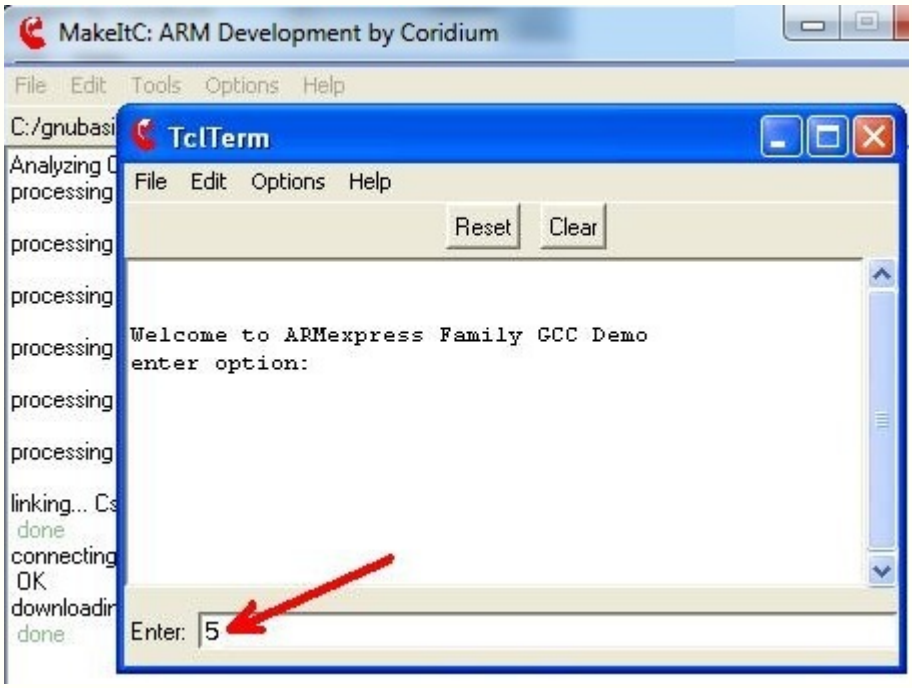
A program that uses IO

The program is now running. This example program communicates with the PC, so to really execute it open a TclTerm window using the tools>terminal menu to communicate with the program

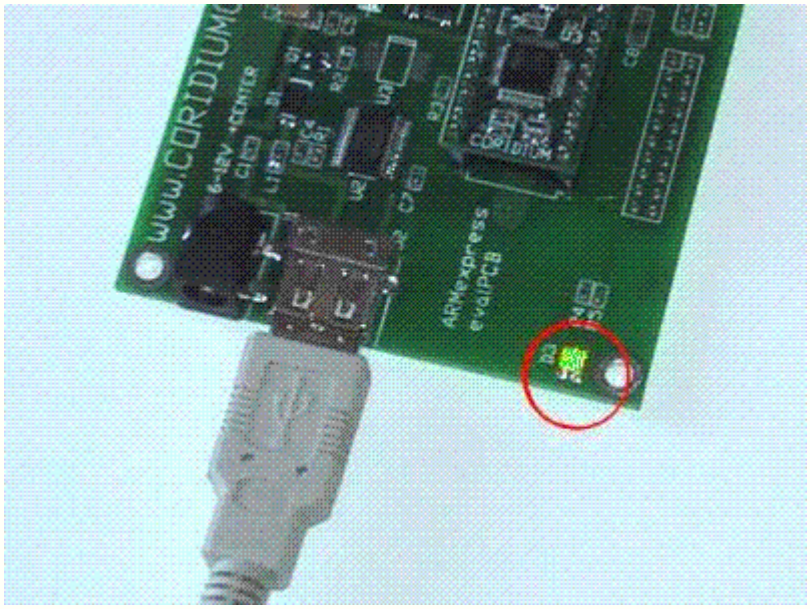




The Csample program will execute a number of routines, some of which require additional hardware (such as I2C or SPI devices). Some of the examples will modulate the LED on the PCB. One of those that flashes the LED is option 5, type that in the box to the right of **Enter:**



And see the results

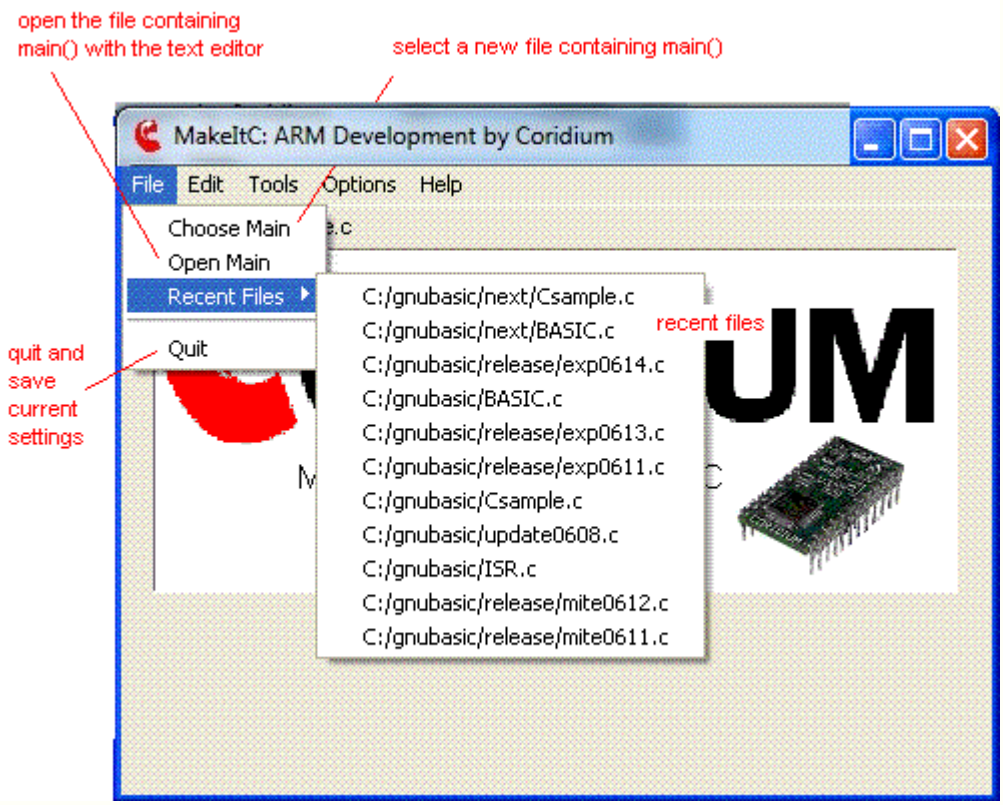


Try some of the other options, refer to the Csample source for details.

More Details on MakeItC

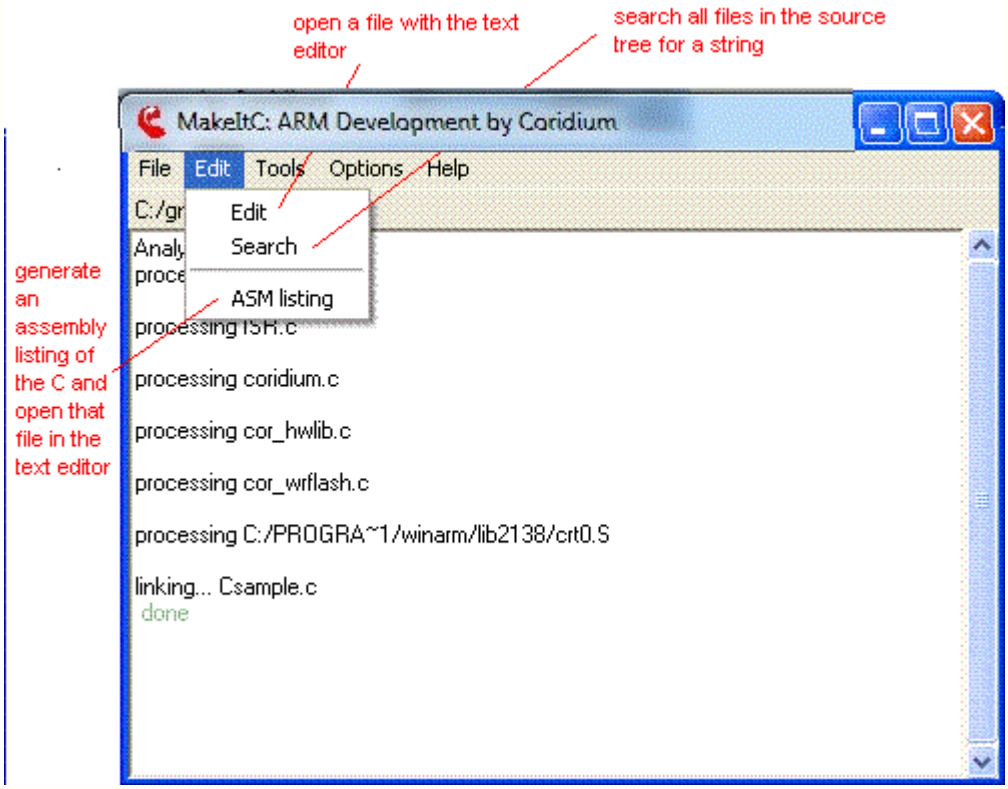
MakeItC Features

Choose a File



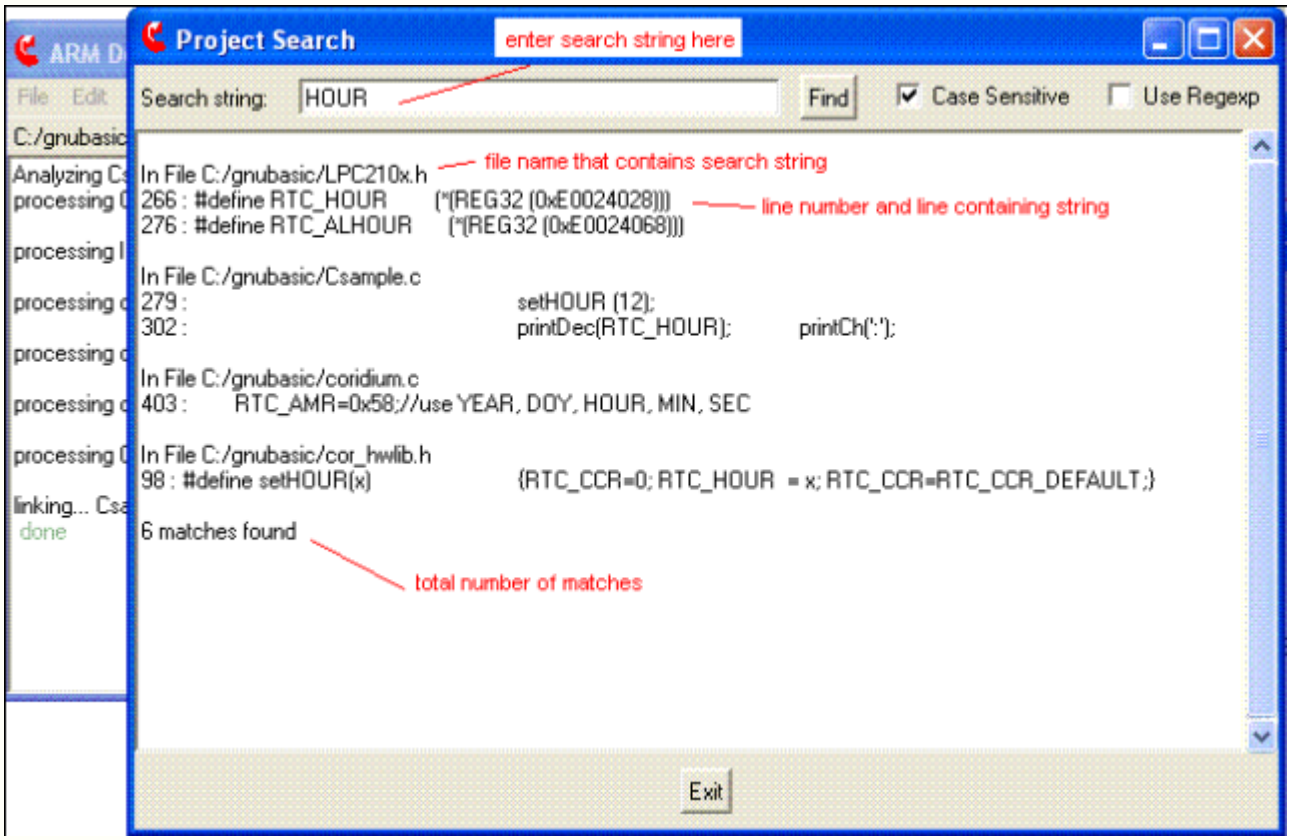
MakeItC assumes that your project can be built completely from your file that includes the main() routine. Any header files will opened and examined for other header files. Any header file that has a corresponding C source file will be added to the list of files to compile. If there is no C source, but an object .o file, that object will be added to list of files to link. In all cases the header and associated file need to be in the same directory, and that directory does not need to be in the directory of the file containing the main(), but it does need to be in the include path (see below). This process is similar to the make process, but is automated and simplified.

Edit Menu

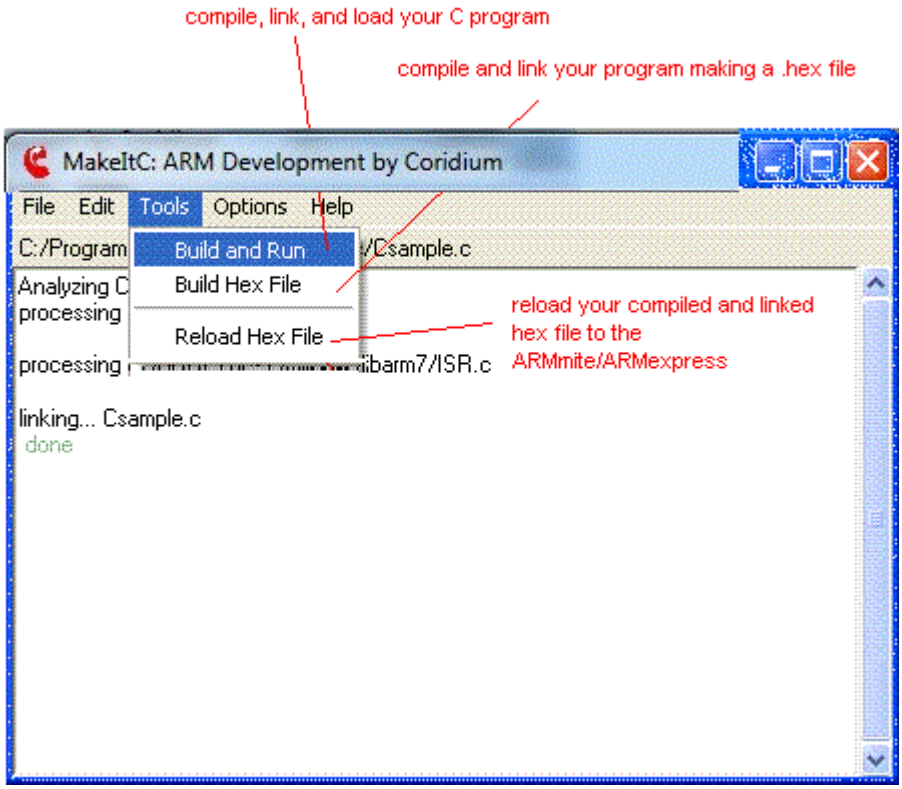


keyw ords: ASM assembly edit search

Search Window



Tools Menu

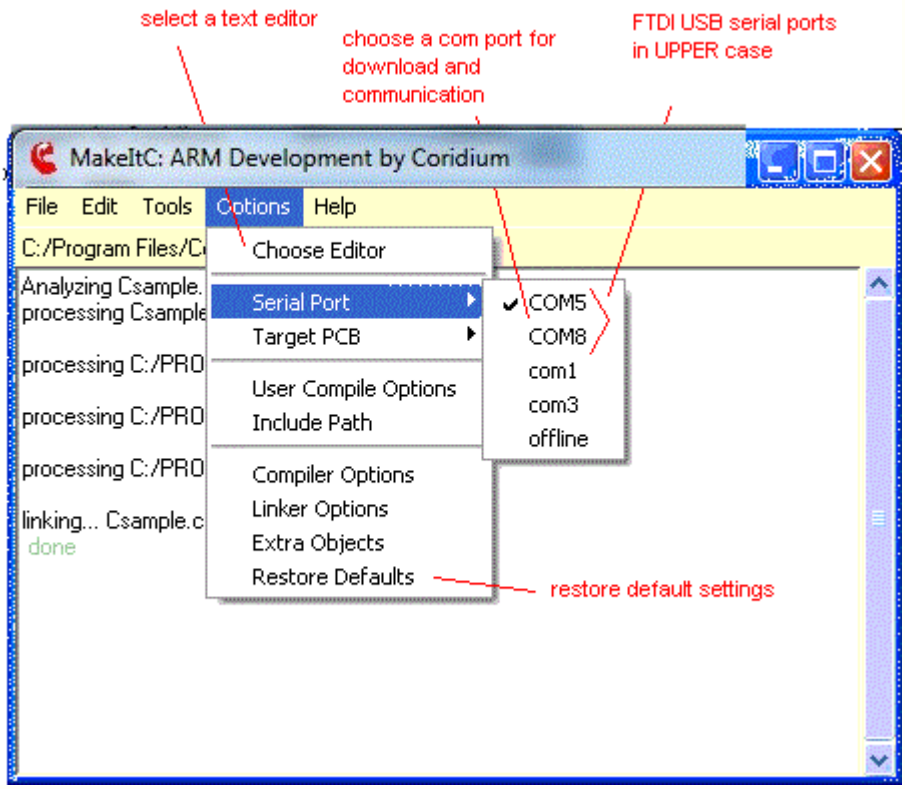


keyw ords: tools build reload terminal hex run

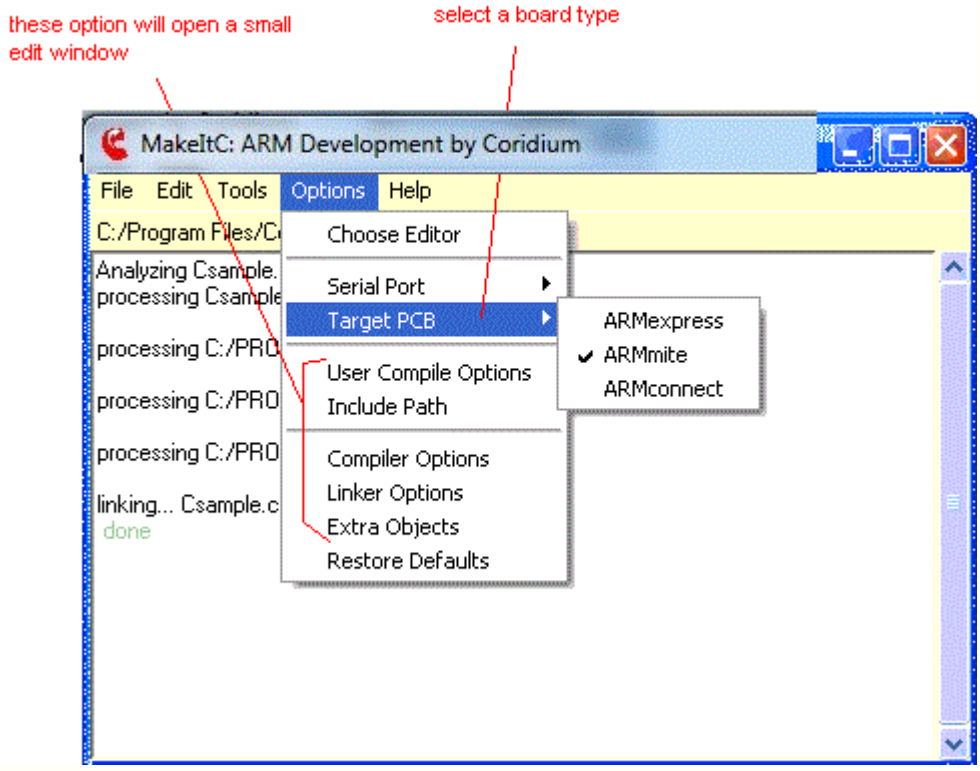
Any file compiled whose name contains ISR, will disable the compiler optimizations. The normal optimization level is -Os (see gcc doc's for details). Code is also compiled to allow links interworking with thumb code.

All compiled files will also generate an assembly listing including the C source, this .lst file may aid in debugging. To see a better version of all source and corresponding assembly language use the option in the Edit menu. The memory map will be saved in a file with the suffix .map.

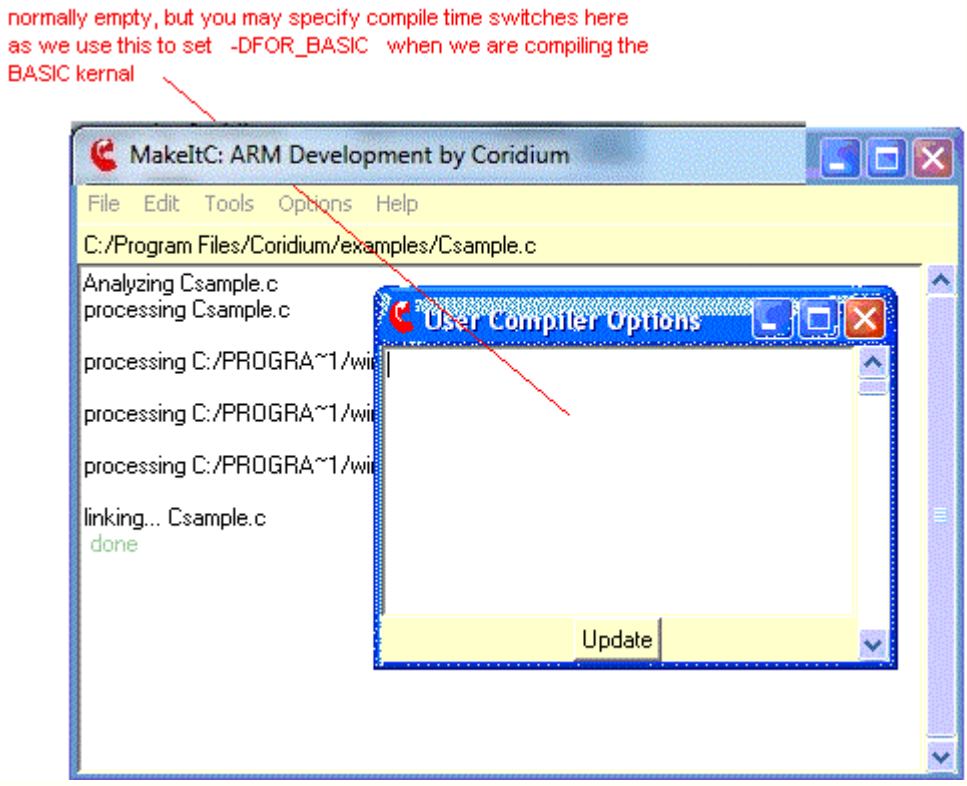
Options Menu

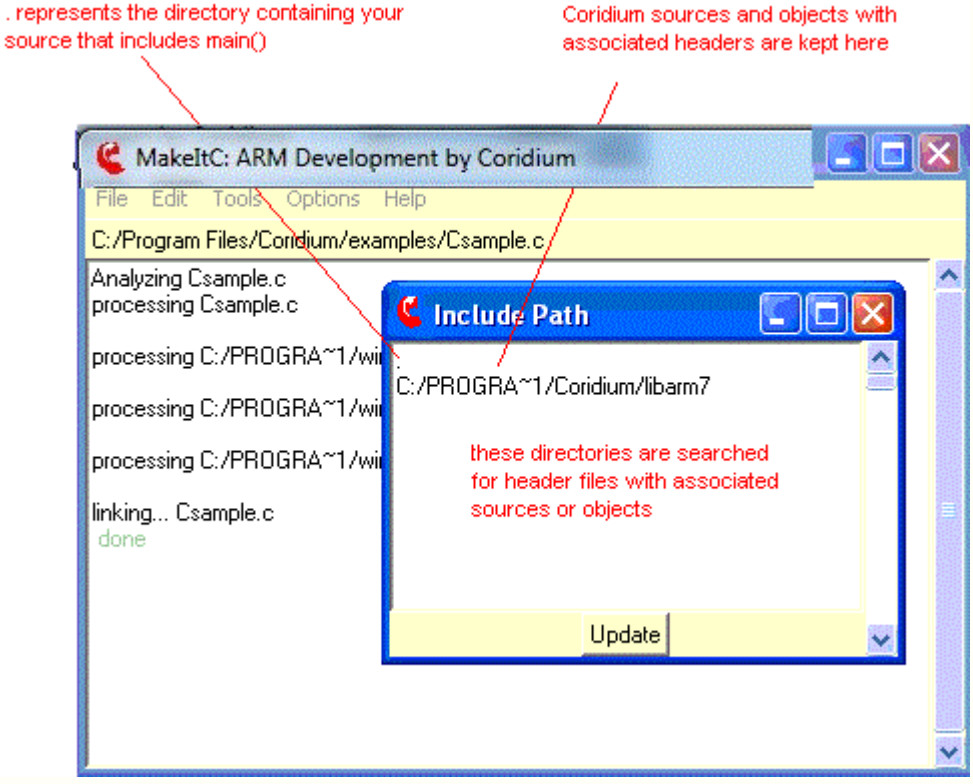


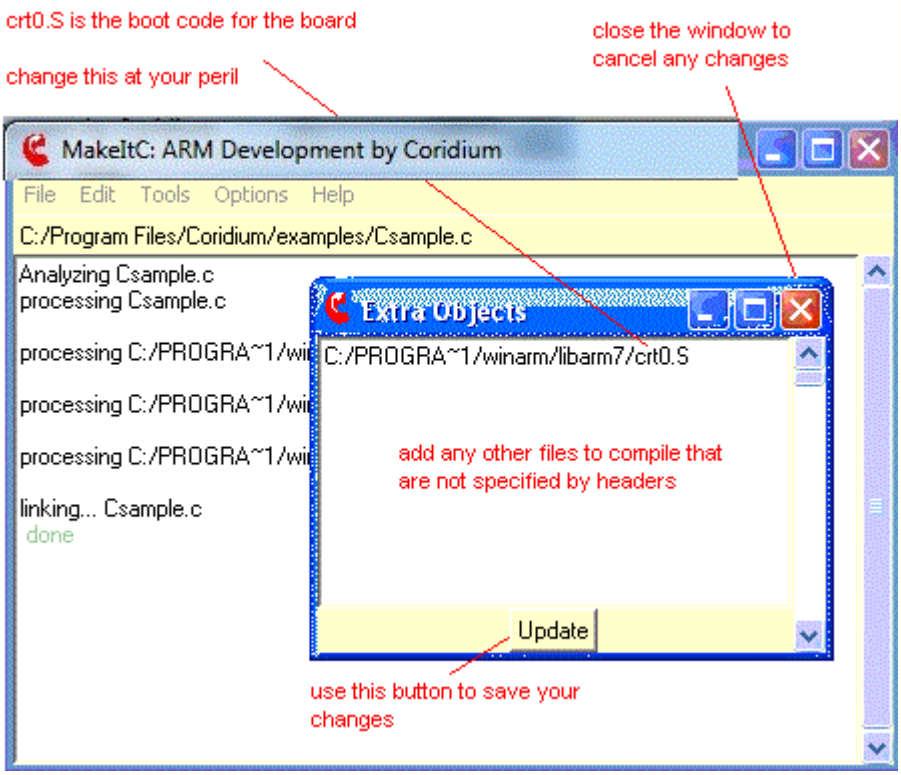
keyw ords: choose editor serial port target PCB compile include linker extra options restore defaults



-


normally empty, but you may specify compile time switches here
as we use this to set -DFOR_BASIC when we are compiling the
BASIC kernal
MakeItC: ARM Development by Coridium
File Edit Tools Options Help
C:/Program Files/Coridium/examples/Csample.c
Analyzing Csample.c
processing Csample.c
processing C:/PROGRA~1/wi
processing C:/PROGRA~1/wi
processing C:/PROGRA~1/wi
linking... Csample.c
done
User Compiler Options
Update


. represents the directory containing your
source that includes main()
Coridium sources and objects with
associated headers are kept here
MakeItC: ARM Development by Coridium
File Edit Tools Options Help
C:/Program Files/Coridium/examples/Csample.c
Analyzing Csample.c
processing Csample.c
processing C:/PROGRA~1/wi
processing C:/PROGRA~1/wi
processing C:/PROGRA~1/wi
linking... Csample.c
done
Include Path
.
C:/PROGRA~1/Coridium/libarm7
these directories are searched
for header files with associated
sources or objects
Update


crt0.S is the boot code for the board
change this at your peril
close the window to
cancel any changes
MakeItC: ARM Development by Coridium
File Edit Tools Options Help
C:/Program Files/Coridium/examples/Csample.c
Analyzing Csample.c
processing Csample.c
processing C:/PROGRA~1/wi
processing C:/PROGRA~1/wi
processing C:/PROGRA~1/wi
linking... Csample.c
done
Extra Objects
C:/PROGRA~1/winarm/libarm7/crt0.S
add any other files to compile that
are not specified by headers
Update
use this button to save your
changes

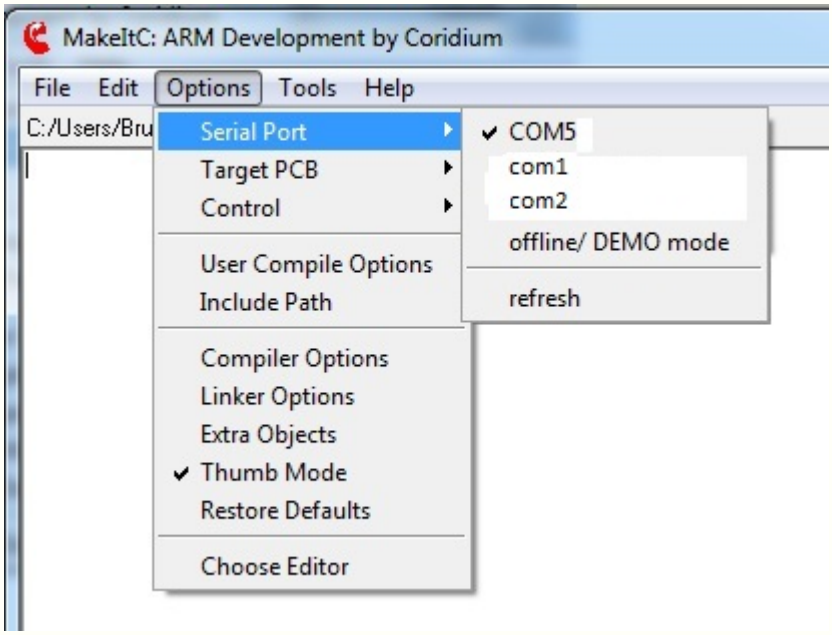
Step 6: Trouble Shooting

Check your cables, check the LED

See **Connect USB** All Coridium boards have an Green LED that is driven when power is applied. If that LED is not on, check you connection, or using the **Schematics** trace the power connections.

Determining which COM port should be used

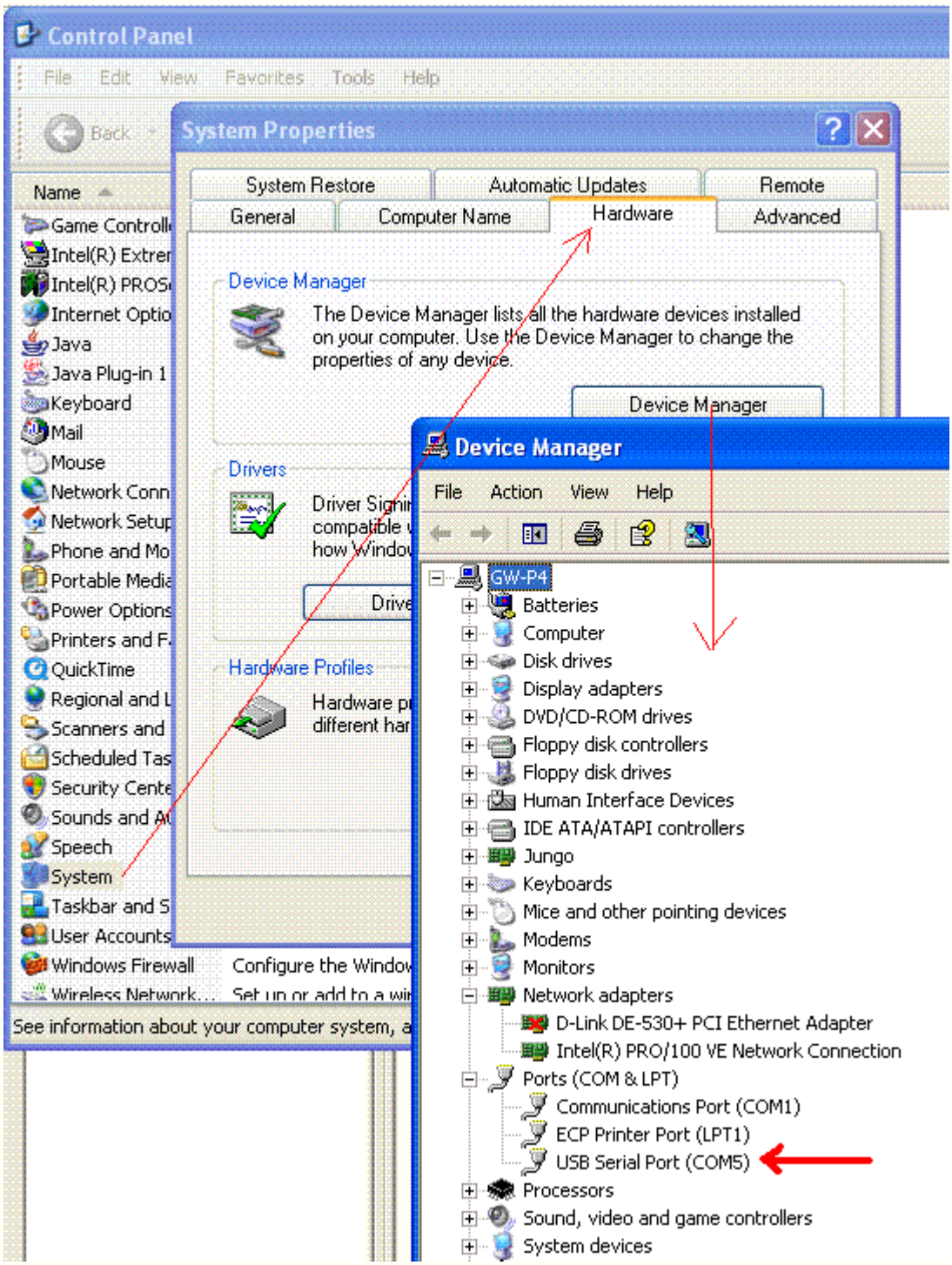
The tools will query the Window registry and will list the available COM ports. If you are using a Coridium USB dongle, a Coridium card with built in USB connection, a SparkFun USB dongle or a FTDI USB cable; then those COM ports will be listed in capital letters. Those in lower case are NOT using a Coridium dongle or built in USB port, do not select those. If you are using some other serial connection, refer to the section on **ISP checks** .



In the example above, COM5 is a Coridium board, com1 and com2 are serial ports built into the PC.

You can also identify an FTDI port using the Device Manager.

Open the Control Panel>System>Device Manager



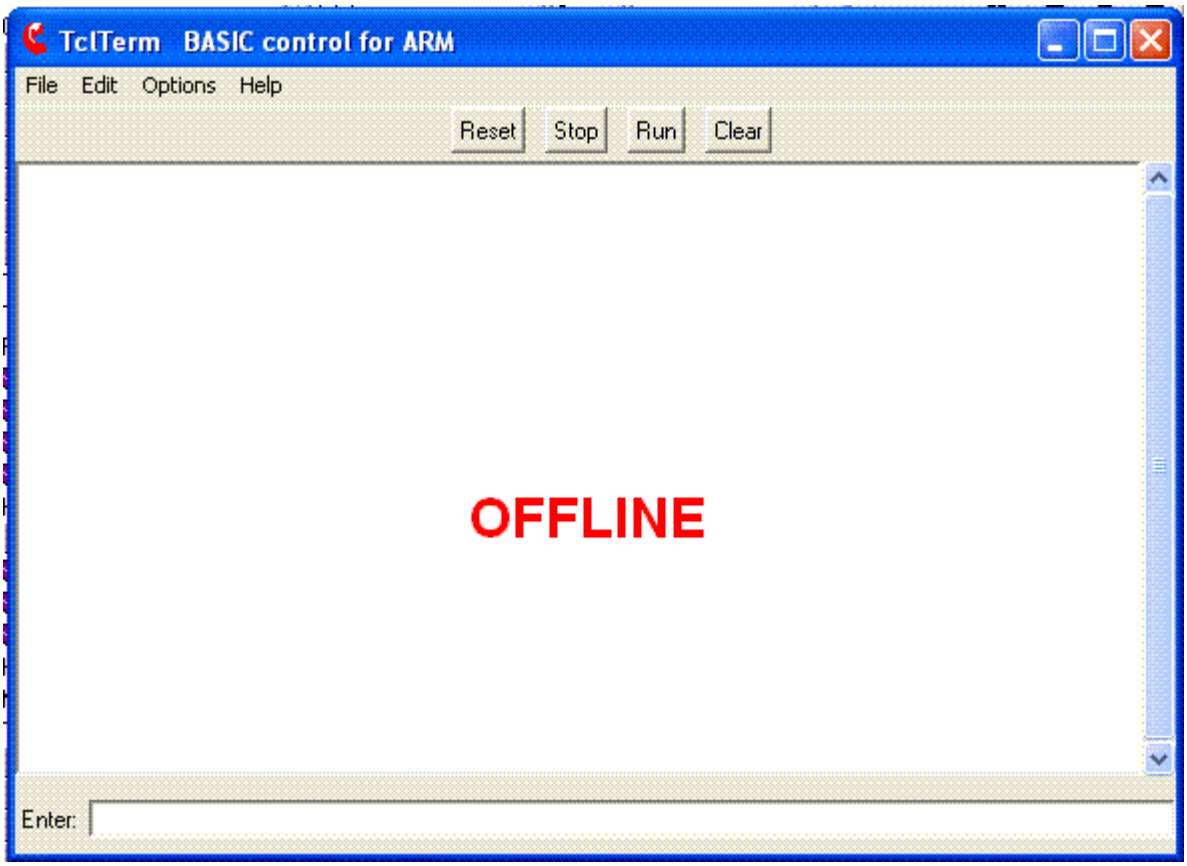
USB(COMx not appearing)

If the USB Serial Port does not appear, check the connection and if its still not there install the driver.

Open the /Program Files(x86)/Coridium/Windows Drivers folder. Run the .exe file in that folder. This will install the FTDI driver. The FTDI FT232RL is the USB device we use on the Coridium USB dongle.

Offline indicator

This will be shown if the port you were using last time the program was run is no longer available. You must reselect a Port using the Option Menu to reestablish communication with the ARM. Make sure any other copies of MakeItC or BASICtools are closed, as you can not open a port simultaneously with more than one program.



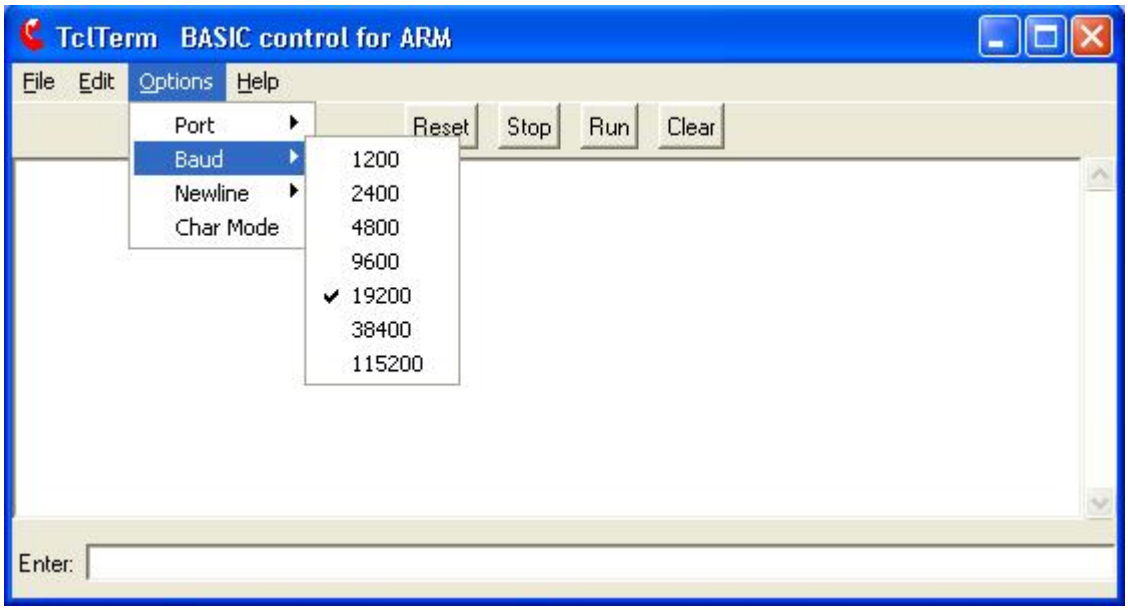
Reset ARM shows no message

If your C program does not appear to start up, go back to sample.c to make sure that is working.

Does your C program initialize the UART0 correctly (other UARTs connect to different pins and require other connections to monitor their ouputs).

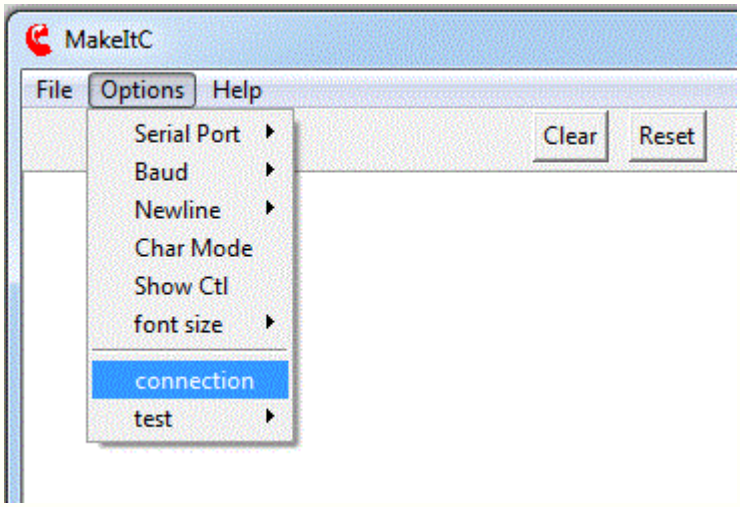
Check Baud Rate

Or you might not have the correct baud rate selected. Make sure it is set to 19200 baud. Baud settings in the Device Manager do NOT affect the MakeItc or BASICtools.

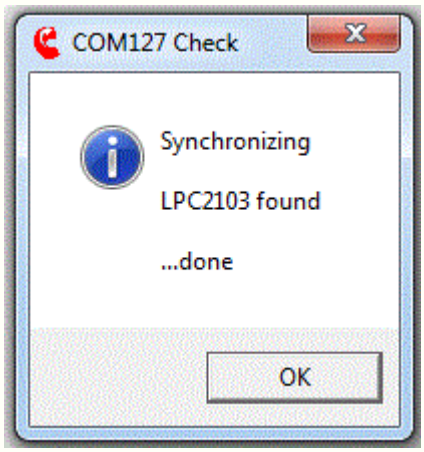


Check ARM connection

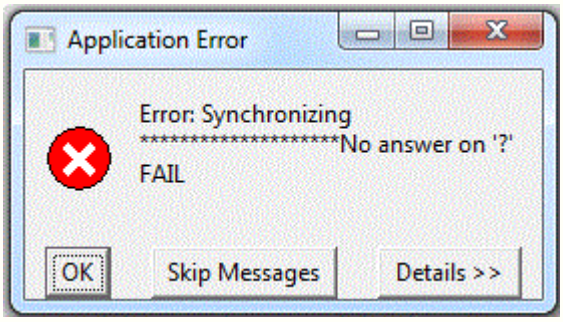
TclTerm has some tools to verify a connection to the ARM chip.



The connection test will use the Coridium USB dongle or built in USB connection to place the ARM into ISP mode (holds P0.14, P2.10 or P0.1 low during RESET, depending on the part -- details in the corresponding NXP User manual). Then a ? will be sent followed by "Synchronized". If an appropriate answer is received a command to ID the part will be sent and that will be reported --



Or if no part is found



If no part is found go back check the connection, power connections, or which port has been selected.

The Compiler

The Compiler

- About
- C Dialect options
- ARM options
- Precompiled Headers
- C Implementation
- C externsions
- License
- Notices

This file documents the use of the GNU compilers.

Also thanks to Georges Menie (www.menie.org) for providing an GPL version of printf that is relatively small and tailored for the embedded environment.

Short Contents

Table of Contents

Next: G++ and GCC, Up: (DIR)

Introduction

This manual documents how to use the GNU compilers, as well as their features and incompatibilities, and how to report bugs. It corresponds to GCC version 4.1.1. The internals of the GNU compilers, including how to port them to new targets and some information about how to write front ends for new languages, are documented in a separate manual. See Introduction.

- G++ and GCC: You can compile C or C++ programs.
- Standards: Language standards supported by GCC.
- Invoking GCC: Command options supported by `gcc'.
- C Implementation: How GCC implements the ISO C specification.
- C Extensions: GNU extensions to the C language family.
- C++ Extensions: GNU extensions to the C++ language.
- Objective-C: GNU Objective-C runtime features.
- Compatibility: Binary Compatibility
- Gcov: `gcov`---a test coverage program.
- Trouble: If you have trouble using GCC.
- Bugs: How, why and where to report bugs.
- Service: How to find suppliers of support for GCC.
- Contributing: How to contribute to testing and developing GCC.
- Funding: How to help assure funding for free software.
- GNU Project: The GNU Project and GNU/Linux.
- Copying: GNU General Public License says how you can copy and share GCC.
- GNU Free Documentation License: How you can copy and share this manual.
- Contributors: People who have contributed to GCC.
- Option Index: Index to command line options.
- Keyword Index: Index of concepts and symbol names.

3.4 Options Controlling C Dialect

The following options control the dialect of C (or languages derived from C, such as C++, Objective-C and Objective-C++) that the compiler accepts:

`-ansi`

In C mode, support all ISO C90 programs. In C++ mode, remove GNU extensions that conflict with ISO C++.

This turns off certain features of GCC that are incompatible with ISO C90 (when compiling C code), or of standard C++ (when compiling C++ code), such as the `asm` and `typeof` keywords, and predefined macros such as `unix` and `vax` that identify the type of system you are using. It also enables the undesirable and rarely used ISO trigraph feature. For the C compiler, it disables recognition of C++ style `//' comments as well as the `inline` keyword.

The alternate keywords `__asm__`, `__extension__`, `__inline__` and `__typeof__` continue to work despite `-ansi`. You would not want to use them in an ISO C program, of course, but it is useful to put them in header files that might be included in compilations done with `-ansi`. Alternate predefined macros such as `__unix__` and `__vax__` are also available, with or without `-ansi`.

The `-ansi` option does not cause non-ISO programs to be rejected gratuitously. For that, `-pedantic` is required in addition to `-ansi`. See Warning Options.

The macro `__STRICT_ANSI__` is predefined when the `-ansi` option is used. Some header files may notice this macro and refrain from declaring certain functions or defining certain macros that the ISO standard doesn't call for; this is to avoid interfering with any programs that might use these names for other things.

Functions which would normally be built in but do not have semantics defined by ISO C (such as `alloca` and `ffs`) are not built-in functions with `-ansi` is used. See Other built-in functions provided by GCC, for details of the functions affected.

`-std=`

Determine the language standard. This option is currently only supported when compiling C or C++. A value for this option must be provided; possible values are

`c89'
`iso9899:1990'
ISO C90 (same as `-ansi`).

`iso9899:199409'
ISO C90 as modified in amendment 1.

`c99'
`c9x'
`iso9899:1999'
`iso9899:199x'
ISO C99. Note that this standard is not yet fully supported; see http://gcc.gnu.org/gcc-4.1/c99status.html for more information. The names `c9x' and `iso9899:199x' are deprecated.

`gnu89'
Default, ISO C90 plus GNU extensions (including some C99 features).

`gnu99'
`gnu9x'

ISO C99 plus GNU extensions. When ISO C99 is fully implemented in GCC, this will become the default. The name `gnu9x' is deprecated.

`c++98'
The 1998 ISO C++ standard plus amendments.

`gnu++98'
The same as `-std=c++98` plus GNU extensions. This is the default for C++ code.

Even when this option is not specified, you can still use some of the features of newer standards in so far as they do not conflict with previous C standards. For example, you may use `__restrict__` even when `-std=c99` is not specified.

The `-std` options specifying some version of ISO C have the same effects as `-ansi`, except that features that were not in ISO C90 but are in the specified version (for example, `//' comments and the `inline` keyword in ISO C99) are not disabled.

See Language Standards Supported by GCC, for details of these standard versions.

`-aux-info` *filename*
Output to the given filename prototyped declarations for all functions declared and/or defined in a translation unit, including those in header files. This option is silently ignored in any language other than C.

Besides declarations, the file indicates, in comments, the origin of each declaration (source file and line), whether the declaration was implicit, prototyped or unprototyped (`I', `N' for new or `O' for old, respectively, in the first character after the line number and the colon), and whether it came from a declaration or a definition (`C' or `F', respectively, in the following character). In the case of function definitions, a K&R-style list of arguments followed by their declarations is also provided, inside comments, after the declaration.

`-fno-asm`
Do not recognize `asm`, `inline` or `typeof` as a keyword, so that code can use these words as identifiers. You can use the keywords `__asm__`, `__inline__` and `__typeof__` instead. `-ansi` implies `-fno-asm`.

In C++, this switch only affects the `typeof` keyword, since `asm` and `inline` are standard keywords. You may want to use the `-fno-gnu-keywords` flag instead, which has the same effect. In C99 mode (`-std=c99` or `-std=gnu99`), this switch only affects the `asm` and `typeof` keywords, since `inline` is a standard keyword in ISO C99.

`-fno-builtin`
`-fno-builtin-`*function*
Don't recognize built-in functions that do not begin with `__builtin_' as prefix. See Other built-in functions provided by GCC, for details of the functions affected, including those which are not built-in functions when `-ansi` or `-std` options for strict ISO C conformance are used because they do not have an ISO standard meaning.

GCC normally generates special code to handle certain built-in functions more efficiently; for instance, calls to `alloca` may become single instructions that adjust the stack directly, and calls to `memcpy` may become inline copy loops. The resulting code is often both smaller and faster, but since the function calls no longer appear as such, you cannot set a breakpoint on those calls, nor can you change the behavior of the functions by linking with a different library. In addition, when a function is recognized as a built-in function, GCC may use information about that function to warn about problems with calls to that function, or to generate more efficient code, even if the resulting code still contains calls to that function. For example, warnings are given with `-Wformat` for bad calls to `printf`, when `printf` is built in, and `strlen` is known not to modify global memory.

With the `-fno-builtin-`*`function`* option only the built-in function *function* is disabled. *function* must not begin with `__builtin_'. If a function is named this is not built-in in this version of GCC, this option is ignored. There is no corresponding `-fbuiltin-`*`function`* option; if you wish to enable built-in functions selectively when using `-fno-builtin` or `-ffreestanding`, you may define macros such as:

```
#define abs(n)          __builtin_abs ((n))
#define strcpy(d, s)    __builtin_strcpy ((d), (s))
```

`-fhosted`

Assert that compilation takes place in a hosted environment. This implies `-fbuiltin`. A hosted environment is one in which the entire standard library is available, and in which `main` has a return type of `int`. Examples are nearly everything except a kernel. This is equivalent to `-fno-freestanding`.

`-ffreestanding`

Assert that compilation takes place in a freestanding environment. This implies `-fno-builtin`. A freestanding environment is one in which the standard library may not exist, and program startup may not necessarily be at `main`. The most obvious example is an OS kernel. This is equivalent to `-fno-hosted`.

See Language Standards Supported by GCC, for details of freestanding and hosted environments.

`-fms-extensions`

Accept some non-standard constructs used in Microsoft header files.

Some cases of unnamed fields in structures and unions are only accepted with this option. See Unnamed struct/union fields within structs/unions, for details.

`-trigraphs`

Support ISO C trigraphs. The `-ansi` option (and `-std` options for strict ISO C conformance) implies `-trigraphs`.

`-no-integrated-cpp`

Performs a compilation in two passes: preprocessing and compiling. This option allows a user supplied "cc1", "cc1plus", or "cc1obj" via the `-B` option. The user supplied compilation step can then add in an additional preprocessing step after normal preprocessing but before compiling. The default is to use the integrated cpp (internal cpp)

The semantics of this option will change if "cc1", "cc1plus", and "cc1obj" are merged.

`-traditional`
`-traditional-cpp`

Formerly, these options caused GCC to attempt to emulate a pre-standard C compiler. They are now only supported with the `-E` switch. The preprocessor continues to support a pre-standard mode. See the GNU CPP manual for details.

`-fcond-mismatch`

Allow conditional expressions with mismatched types in the second and third arguments. The value of such an expression is void. This option is not supported for C++.

`-funsigned-char`

Let the type `char` be unsigned, like `unsigned char`.

Each kind of machine has a default for what `char` should be. It is either like `unsigned char` by default or like `signed char` by default.

Ideally, a portable program should always use `signed char` or `unsigned char` when it depends on the signedness of an object. But many programs have been written to use plain `char` and expect it to be signed, or expect it to be unsigned, depending on the machines they were written for. This option, and its inverse, let you make such a program work with the opposite default.

The type `char` is always a distinct type from each of `signed char` or `unsigned char`, even though its behavior is always just like one of those two.

`-fsigned-char`

Let the type `char` be signed, like `signed char`.

Note that this is equivalent to `-fno-unsigned-char`, which is the negative form of `-funsigned-char`. Likewise, the option `-fno-signed-char` is equivalent to `-funsigned-char`.

`-fsigned-bitfields`
`-funsigned-bitfields`
`-fno-signed-bitfields`
`-fno-unsigned-bitfields`

These options control whether a bit-field is signed or unsigned, when the declaration does not use either `signed` or `unsigned`. By default, such a bit-field is signed, because this is consistent: the basic integer types such as `int` are signed types.

3.17.2 ARM Options

These `-m' options are defined for Advanced RISC Machines (ARM) architectures:

`-mabi=`*name*
: Generate code for the specified ABI. Permissible values are: `apcs-gnu', `atpcs', `aapcs', `aapcs-linux' and `iwmmxt'.

`-mapcs-frame`
: Generate a stack frame that is compliant with the ARM Procedure Call Standard for all functions, even if this is not strictly necessary for correct execution of the code. Specifying `-fomit-frame-pointer` with this option will cause the stack frames not to be generated for leaf functions. The default is `-mno-apcs-frame`.

`-mapcs`
: This is a synonym for `-mapcs-frame`.

`-mthumb-interwork`
: Generate code which supports calling between the ARM and Thumb instruction sets. Without this option the two instruction sets cannot be reliably used inside one program. The default is `-mno-thumb-interwork`, since slightly larger code is generated when `-mthumb-interwork` is specified.

`-mno-sched-prolog`
: Prevent the reordering of instructions in the function prolog, or the merging of those instruction with the instructions in the function's body. This means that all functions will start with a recognizable set of instructions (or in fact one of a choice from a small set of different function prologues), and this information can be used to locate the start if functions inside an executable piece of code. The default is `-msched-prolog`.

`-mhard-float`
: Generate output containing floating point instructions. This is the default.

`-msoft-float`
: Generate output containing library calls for floating point. **Warning:** the requisite libraries are not available for all ARM targets. Normally the facilities of the machine's usual C compiler are used, but this cannot be done directly in cross-compilation. You must make your own arrangements to provide suitable library functions for cross-compilation.

  `-msoft-float` changes the calling convention in the output file; therefore, it is only useful if you compile *all* of a program with this option. In particular, you need to compile `libgcc.a`, the library that comes with GCC, with `-msoft-float` in order for this to work.

`-mfloat-abi=`*name*
: Specifies which ABI to use for floating point values. Permissible values are: `soft', `softfp' and `hard'.

  `soft' and `hard' are equivalent to `-msoft-float` and `-mhard-float` respectively. `softfp' allows the generation of floating point instructions, but still uses the soft-float calling conventions.

`-mlittle-endian`
: Generate code for a processor running in little-endian mode. This is the default for all standard configurations.

`-mbig-endian`
: Generate code for a processor running in big-endian mode; the default is to compile code for a little-endian processor.

`-mwords-little-endian`
: This option only applies when generating code for big-endian processors. Generate code for a

little-endian word order but a big-endian byte order. That is, a byte order of the form `32107654'. Note: this option should only be used if you require compatibility with code for big-endian ARM processors generated by versions of the compiler prior to 2.8.

`-mcpu=`*name*

This specifies the name of the target ARM processor. GCC uses this name to determine what kind of instructions it can emit when generating assembly code. Permissible names are: `arm2', `arm250', `arm3', `arm6', `arm60', `arm600', `arm610', `arm620', `arm7', `arm7m', `arm7d', `arm7dm', `arm7di', `arm7dmi', `arm70', `arm700', `arm700i', `arm710', `arm710c', `arm7100', `arm7500', `arm7500fe', `arm7tdmi', `arm7tdmi-s', `arm8', `strongarm', `strongarm110', `strongarm1100', `arm8', `arm810', `arm9', `arm9e', `arm920', `arm920t', `arm922t', `arm946e-s', `arm966e-s', `arm968e-s', `arm926ej-s', `arm940t', `arm9tdmi', `arm10tdmi', `arm1020t', `arm1026ej-s', `arm10e', `arm1020e', `arm1022e', `arm1136j-s', `arm1136jf-s', `mpcore', `mpcorenovfp', `arm1176jz-s', `arm1176jzf-s', `xscale', `iwmmxt', `ep9312'.

`-mtune=`*name*

This option is very similar to the `-mcpu=` option, except that instead of specifying the actual target processor type, and hence restricting which instructions can be used, it specifies that GCC should tune the performance of the code as if the target were of the type specified in this option, but still choosing the instructions that it will generate based on the cpu specified by a `-mcpu=` option. For some ARM implementations better performance can be obtained by using this option.

`-march=`*name*

This specifies the name of the target ARM architecture. GCC uses this name to determine what kind of instructions it can emit when generating assembly code. This option can be used in conjunction with or instead of the `-mcpu=` option. Permissible names are: `armv2', `armv2a', `armv3', `armv3m', `armv4', `armv4t', `armv5', `armv5t', `armv5te', `armv6', `armv6j', `iwmmxt', `ep9312'.

`-mfpu=`*name*

`-mfpe=`*number*

`-mfp=`*number*

This specifies what floating point hardware (or hardware emulation) is available on the target. Permissible names are: `fpa', `fpe2', `fpe3', `maverick', `vfp'. `-mfp` and `-mfpe` are synonyms for `-mfpu=`fpe'*number*, for compatibility with older versions of GCC.

If `-msoft-float` is specified this specifies the format of floating point values.

`-mstructure-size-boundary=`*n*

The size of all structures and unions will be rounded up to a multiple of the number of bits set by this option. Permissible values are 8, 32 and 64. The default value varies for different toolchains. For the COFF targeted toolchain the default value is 8. A value of 64 is only allowed if the underlying ABI supports it.

Specifying the larger number can produce faster, more efficient code, but can also increase the size of the program. Different values are potentially incompatible. Code compiled with one value cannot necessarily expect to work with code or libraries compiled with another value, if they exchange information using structures or unions.

`-mabort-on-noreturn`

Generate a call to the function `abort` at the end of a `noreturn` function. It will be executed if the function tries to return.

`-mlong-calls`

`-mno-long-calls`

Tells the compiler to perform function calls by first loading the address of the function into a

register and then performing a subroutine call on this register. This switch is needed if the target function will lie outside of the 64 megabyte addressing range of the offset based version of subroutine call instruction.

Even if this switch is enabled, not all function calls will be turned into long calls. The heuristic is that static functions, functions which have the `short-call' attribute, functions that are inside the scope of a `#pragma no_long_calls' directive and functions whose definitions have already been compiled within the current compilation unit, will not be turned into long calls. The exception to this rule is that weak function definitions, functions with the `long-call' attribute or the `section' attribute, and functions that are within the scope of a `#pragma long_calls' directive, will always be turned into long calls.

This feature is not enabled by default. Specifying `-mno-long-calls` will restore the default behavior, as will placing the function calls within the scope of a `#pragma long_calls_off' directive. Note these switches have no effect on how the compiler generates code to handle function calls via function pointers.

`-mnop-fun-dllimport`
Disable support for the `dllimport` attribute.

`-msingle-pic-base`
Treat the register used for PIC addressing as read-only, rather than loading it in the prologue for each function. The run-time system is responsible for initializing this register with an appropriate value before execution begins.

`-mpic-register=`*reg*
Specify the register to be used for PIC addressing. The default is R10 unless stack-checking is enabled, when R9 is used.

`-mcirrus-fix-invalid-insns`
Insert NOPs into the instruction stream to in order to work around problems with invalid Maverick instruction combinations. This option is only valid if the `-mcpu=ep9312` option has been used to enable generation of instructions for the Cirrus Maverick floating point co-processor. This option is not enabled by default, since the problem is only present in older Maverick implementations. The default can be re-enabled by use of the `-mno-cirrus-fix-invalid-insns` switch.

`-mpoke-function-name`
Write the name of each function into the text section, directly preceding the function prologue. The generated code is similar to this:

```
t0
    .ascii "arm_poke_function_name", 0
    .align
t1
    .word 0xff000000 + (t1 - t0)
arm_poke_function_name
    mov     ip, sp
    stmfd   sp!, {fp, ip, lr, pc}
    sub     fp, ip, #4
```

When performing a stack backtrace, code can inspect the value of `pc` stored at `fp + 0`. If the trace function then looks at location `pc - 12` and the top 8 bits are set, then we know that there is a function name embedded immediately preceding this location and has length `((pc[-3]) & 0xff000000)`.

`-mthumb`
Generate code for the 16-bit Thumb instruction set. The default is to use the 32-bit ARM instruction set.

`-mtpcs-frame`

Generate a stack frame that is compliant with the Thumb Procedure Call Standard for all non-leaf functions. (A leaf function is one that does not call any other functions.) The default is `-mno-tpcs-frame`.

`-mtpcs-leaf-frame`

Generate a stack frame that is compliant with the Thumb Procedure Call Standard for all leaf functions. (A leaf function is one that does not call any other functions.) The default is `-mno-apcs-leaf-frame`.

`-mcallee-super-interworking`

Gives all externally visible functions in the file being compiled an ARM instruction set header which switches to Thumb mode before executing the rest of the function. This allows these functions to be called from non-interworking code.

`-mcaller-super-interworking`

Allows calls via function pointers (including virtual functions) to execute correctly regardless of whether the target code has been compiled for interworking or not. There is a small overhead in the cost of executing a function pointer if this option is enabled.

`-mtp=`*name*

Specify the access model for the thread local storage pointer. The valid models are `soft`, which generates calls to `__aeabi_read_tp`, `cp15`, which fetches the thread pointer from `cp15` directly (supported in the arm6k architecture), and `auto`, which uses the best available method for the selected processor. The default setting is `auto`.

3.20 Using Precompiled Headers

Often large projects have many header files that are included in every source file. The time the compiler takes to process these header files over and over again can account for nearly all of the time required to build the project. To make builds faster, GCC allows users to `precompile' a header file; then, if builds can use the precompiled header file they will be much faster.

To create a precompiled header file, simply compile it as you would any other file, if necessary using the `-x` option to make the driver treat it as a C or C++ header file. You will probably want to use a tool like `make` to keep the precompiled header up-to-date when the headers it contains change.

A precompiled header file will be searched for when `#include` is seen in the compilation. As it searches for the included file (see Search Path) the compiler looks for a precompiled header in each directory just before it looks for the include file in that directory. The name searched for is the name specified in the `#include` with `` `.gch' `` appended. If the precompiled header file can't be used, it is ignored.

For instance, if you have `#include "all.h"`, and you have `all.h.gch` in the same directory as `all.h`, then the precompiled header file will be used if possible, and the original header will be used otherwise.

Alternatively, you might decide to put the precompiled header file in a directory and use `-I` to ensure that directory is searched before (or instead of) the directory containing the original header. Then, if you want to check that the precompiled header file is always used, you can put a file of the same name as the original header in this directory containing an `#error` command.

This also works with `-include`. So yet another way to use precompiled headers, good for projects not designed with precompiled header files in mind, is to simply take most of the header files used by a project, include them from another header file, precompile that header file, and `-include` the precompiled header. If the header files have guards against multiple inclusion, they will be skipped because they've already been included (in the precompiled header).

If you need to precompile the same header file for different languages, targets, or compiler options, you can instead make a *directory* named like `all.h.gch`, and put each precompiled header in the directory, perhaps using `-o`. It doesn't matter what you call the files in the directory, every precompiled header in the directory will be considered. The first precompiled header encountered in the directory that is valid for this compilation will be used; they're searched in no particular order.

There are many other possibilities, limited only by your imagination, good sense, and the constraints of your build system.

A precompiled header file can be used only when these conditions apply:

- Only one precompiled header can be used in a particular compilation.
- A precompiled header can't be used once the first C token is seen. You can have preprocessor directives before a precompiled header; you can even include a precompiled header from inside another header, so long as there are no C tokens before the `#include`.
- The precompiled header file must be produced for the same language as the current compilation. You can't use a C precompiled header for a C++ compilation.
- The precompiled header file must have been produced by the same compiler binary as the current compilation is using.
- Any macros defined before the precompiled header is included must either be defined in the

same way as when the precompiled header was generated, or must not affect the precompiled header, which usually means that they don't appear in the precompiled header at all.

The `-D` option is one way to define a macro before a precompiled header is included; using a `#define` can also do it. There are also some options that define macros implicitly, like `-O` and `-Wdeprecated`; the same rule applies to macros defined this way.

- If debugging information is output when using the precompiled header, using `-g` or similar, the same kind of debugging information must have been output when building the precompiled header. However, a precompiled header built using `-g` can be used in a compilation when no debugging information is being output.
- The same `-m` options must generally be used when building and using the precompiled header. See Submodel Options, for any cases where this rule is relaxed.
- Each of the following options must be the same when building and using the precompiled header:
- `-fexceptions -funit-at-a-time`

- Some other command-line options starting with `-f`, `-p`, or `-O` must be defined in the same way as when the precompiled header was generated. At present, it's not clear which options are safe to change and which are not; the safest choice is to use exactly the same options when generating and using the precompiled header. The following are known to be safe:
- `-fmessage-length= -fpreprocessed`
- `-fsched-interblock -fsched-spec -fsched-spec-load -fsched-spec-load-dangerous`
- `-fsched-verbose=<number> -fschedule-insns -fvisibility=`
- `-pedantic-errors`

For all of these except the last, the compiler will automatically ignore the precompiled header if the conditions aren't met. If you find an option combination that doesn't work and doesn't cause the precompiled header to be ignored, please consider filing a bug report, see Bugs.

If you do use differing options when generating and using the precompiled header, the actual behavior will be a mixture of the behavior for the options. For instance, if you use `-g` to generate the precompiled header but not when using it, you may or may not get debugging information for routines in the precompiled header.

4 C Implementation-defined behavior

A conforming implementation of ISO C is required to document its choice of behavior in each of the areas that are designated "implementation defined". The following lists all such areas, along with the section numbers from the ISO/IEC 9899:1990 and ISO/IEC 9899:1999 standards. Some areas are only implementation-defined in one version of the standard.

Some choices depend on the externally determined ABI for the platform (including standard character encodings) which GCC follows; these are listed as "determined by ABI" below. See Binary Compatibility, and http://gcc.gnu.org/readings.html. Some choices are documented in the preprocessor manual. See Implementation-defined behavior. Some choices are made by the library and operating system (or other environment when compiling for a freestanding environment); refer to their documentation for details.

5 Extensions to the C Language Family

GNU C provides several language features not found in ISO standard C. (The `-pedantic` option directs GCC to print a warning message if any of these features is used.) To test for the availability of these features in conditional compilation, check for a predefined macro `__GNUC__`, which is always defined under GCC.

These extensions are available in C and Objective-C. Most of them are also available in C++. See Extensions to the C++ Language, for extensions that apply *only* to C++.

Some features that are in ISO C99 but not C89 or C++ are also, as extensions, accepted by GCC in C89 mode and in C++.

- Statement Exprs: Putting statements and declarations inside expressions.
- Local Labels: Labels local to a block.
- Labels as Values: Getting pointers to labels, and computed gotos.
- Nested Functions: As in Algol and Pascal, lexical scoping of functions.
- Constructing Calls: Dispatching a call to another function.
- Typeof: `typeof`: referring to the type of an expression.
- Conditionals: Omitting the middle operand of a `?:' expression.
- Long Long: Double-word integers---`long long int`.
- Complex: Data types for complex numbers.
- Hex Floats: Hexadecimal floating-point constants.
- Zero Length: Zero-length arrays.
- Variable Length: Arrays whose length is computed at run time.
- Empty Structures: Structures with no members.
- Variadic Macros: Macros with a variable number of arguments.
- Escaped Newlines: Slightly looser rules for escaped newlines.
- Subscripting: Any array can be subscripted, even if not an lvalue.
- Pointer Arith: Arithmetic on `void`-pointers and function pointers.
- Initializers: Non-constant initializers.
- Compound Literals: Compound literals give structures, unions or arrays as values.
- Designated Inits: Labeling elements of initializers.
- Cast to Union: Casting to union type from any member of the union.
- Case Ranges: `case 1 ... 9' and such.
- Mixed Declarations: Mixing declarations and code.
- Function Attributes: Declaring that functions have no side effects, or that they can never return.
- Attribute Syntax: Formal syntax for attributes.
- Function Prototypes: Prototype declarations and old-style definitions.
- C++ Comments: C++ comments are recognized.
- Dollar Signs: Dollar sign is allowed in identifiers.
- Character Escapes: `\e' stands for the character <ESC>.
- Variable Attributes: Specifying attributes of variables.
- Type Attributes: Specifying attributes of types.
- Alignment: Inquiring about the alignment of a type or variable.
- Inline: Defining inline functions (as fast as macros).
- Extended Asm: Assembler instructions with C expressions as operands. (With them you can define ``built-in'' functions.)
- Constraints: Constraints for asm operands

- Asm Labels: Specifying the assembler name to use for a C symbol.
- Explicit Reg Vars: Defining variables residing in specified registers.
- Alternate Keywords: `__const__`, `__asm__`, etc., for header files.
- Incomplete Enums: `enum foo;`, with details to follow.
- Function Names: Printable strings which are the name of the current function.
- Return Address: Getting the return or frame address of a function.
- Vector Extensions: Using vector instructions through built-in functions.
- Offsetof: Special syntax for implementing `offsetof`.
- Atomic Builtins: Built-in functions for atomic memory access.
- Object Size Checking: Built-in functions for limited buffer overflow checking.
- Other Builtins: Other built-in functions.
- Target Builtins: Built-in functions specific to particular targets.
- Target Format Checks: Format checks specific to particular targets.
- Pragmas: Pragmas accepted by GCC.
- Unnamed Fields: Unnamed struct/union fields within structs/unions.
- Thread-Local: Per-thread variables.

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The “Program”, below, refers to any such program or work, and a “work based on the Program” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term

“modification”.) Each licensee is addressed as “you”.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

2. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

4. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily

used for software interchange; or,

b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

1. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
1. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.
1. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
1. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in

other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

1. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
1. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

1. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

1. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
1. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY

YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
    one line to give the program's name and a brief idea of what it does.
    Copyright (C) year  name of author

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
02110-1301, USA
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
    type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
    Yoyodyne, Inc., hereby disclaims all copyright interest in the program
    `Gnomovision' (which makes passes at compilers) written by James Hacker.

    signature of Ty Coon, 1 April 1989
    Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Runtime Objects

The hardware support libraries and objects are part of the **ARMbasic©** compiler. It is distributed as part of hardware sold by Coridium Corp. such as the ARMexpress module and ARMmite single board computers. Those objects may be used without restriction on Coridium hardware or hardware of Coridium's licensees.

Notices

NO WARRANTY

1. THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. CORIDIUM PROVIDES THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

2. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL CORIDIUM BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

- Windows® is a registered trademark of Microsoft Corporation.
- VisualBASIC® is a registered trademark of Microsoft Corporation.
- BASIC Stamp® is a registered trademark of Parallax, Inc.
- PBASIC™ is a trademark of Parallax, Inc.
- I^2C® is a registered trademark of Philips Corporation.
- 1-Wire® is a registered trademark of Maxim/Dallas Semiconductor.
- SPI™ is a trademark of Motorola

Parts of this documentation are adapted from **www.cppreference.com**

This documentation is released under the **GFDL** license.

MakeItC Operation

Looking for files

MakeItC assumes that your project can be built completely from your file that includes the main() routine.

Any header files will opened and examined for other header files.

Any header file that has a corresponding C source file will be added to the list of files to compile. If there is no C source, but an object .o file, that object will be added to list of files to link.

In all cases the header and associated object or source file needs to be in the same directory, and that directory does not need to be in the directory of the file containing the main(), but it does need to be in the include path.

This process is similar to the make process, but is automated and simplified.

Libraries

Some simple input and output routines provide communication from the ARMmite/ARMexpress back to the PC. These routines are normally available to GCC on other platforms.

They have been adapted from the GCC sources or in some cases a simplified subset such as printf.

Com Functions

- getc
- gets
- printf
- putchar
- puts
- sprintf

String Convert Functions

- atoi
- atoh

Single Precision Math Functions

float atanf
float cosf
float sinf
float tanf
float tanhf
float frexpf
float modff
float ceilf
float fabsf
float floorf

Memory Functions

- memchr
- memcmp
- memcpy
- memmove
- memset

String Functions

- strcat
- strchr
- strcmp
- strcpy

strcspn
strlen
strncat
strncmp
strncpy
strstr
strrchr

cor_min_io Library

getCh
getStr
printDec
printHex
printStr
str2dec

Communication Functions

Some simple input and output routines provide communication from the ARMmite/ARMexpress back to the PC. These routines are normally available to GCC on other platforms.

They have been adapted from the GCC sources or in some cases a simplified subset as printf.

Com Functions

- **getc**
- **gets**
- **printf**
- **putchar**
- **puts**
- **sprintf**

getc

Syntax

int getc (int UARTnum);

Description

getc(int UARTnum) will return a single character (0-255) if one is available in the input buffer form the corresponding UART.

If no character is available then -1 is returned.

This routine unlike gets returns immediately whether a character is available or not.

Example

```
while (1) {
   ch = getc(0);
   if (ch != -1) break;
}
```

See also

- gets

gets

Syntax

void gets(char *stringPointer);

Description

gets will accept a string from the USB port on the ARMmite or SIN on the ARMexpress.

The string will be saved into the stringPointer char array. A CR (carriage return) will terminate the string and be replaced by a 0 (null) character will be returned in the array.

gets is implelmented using getline, with maxLen limiting the number of characters copied into stringPointer. If that limit is hit, then the routine will return. This limit is set to 256, and may be changed by the user in the coridium.h file

gets will wait for input indefinately.

Example

```
while (1) {
  puts("enter option:");
  gets(instring);
  inval = atoi(instring);

  switch (inval) {
    case 1:
```

See also

- getc
- gets

printf

Syntax

int printf (char * format, ...);

Description

printf will convert an arbitrary number of parameters and using the format string assign those into a string and send them to the serial port (SOUT)

format may be a simple string in which case that string is printed

or it can contain parameters to be filled with values

%c -- replace this with a single character

%s -- replace this with a string

%d -- replace with a decimal value

%x -- replace with a hexadecimal value (%X upshifts A-F)

%u -- replace with the unsigned decimal value

In addition there are modifiers to further control the output string

%#d -- where # indicates how many characters will be used to represent the value -- in this case the value is right justified and blank filled

%0#x -- in this case the number is right justified but 0 filled for # total spaces

%-#s -- this left justifies the string and fills the right side with spaces

NOTE:- the printf.c library generates 2 warnings on "dereferencing type-punned pointer will break strict-aliasing rules"

In 2012 floating point support has been added to printf, but it is not the default case as it adds 4K to the code. To enable floating point support, printf.c must be compiled with USE_FLOAT defined, which can be done by adding #define USE_FLOAT into the printf.h source or it can be done by adding the User Compile option in MakeItC -DUSE_FLOAT

Example

```
printf ("1234asdf");          // will display 1234asdf
printf ("%c", 0x31 );         // will display 1
printf ("%d", 1 );            // will display 1
printf ("%9d", 1 );           // will display "        1"
```

```
printf ("%-9d", 1 );          // will display "1         "
printf ("%09d", 1 );          // will display "0000000001"
printf ("%9s", "1" );         // will display "         1"
```

See also

- **putchar**
- **puts**

putchar

Syntax

int putchar (char c, int UARTnum);

Description

putchar will send a single character to the PC via SOUT.

putchar returns the character sent in form of an integer.

Example

```
putchar (0x31, 0);          // will display 1 to UART0 -- connected to the PC USB port
```

See also

- printf
- puts

puts

Syntax

int puts(char * string);

Description

puts send a single string to the PC via SOUT, followed by the newline character.

puts returns -1 on error, otherwise the last character printed

Example

```
puts("%c 0x31");          // will display %c 0x31
```

See also

- printf
- putchar

sprintf

Syntax

int sprintf (char *returnStr, char * format, ...);

Description

This routine will generate a formated string using parametric input.

The returnStr will be set to this formatted string, and the number of characters in that string are returned.

For format string details, see **printf** .

In 2012 floating point support has been added to printf, but it is not the default case as it adds 4K to the code. To enable floating point support, printf.c must be compiled with USE_FLOAT defined, which can be done by adding #define USE_FLOAT into the printf.h source or it can be done by adding the User Compile option in MakeItC -DUSE_FLOAT

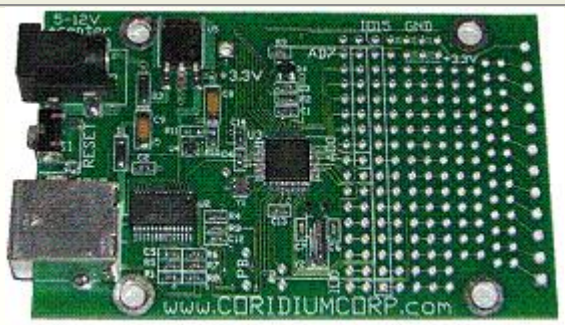
Example

```
sprintf (s, "x = %d", 33);   // will return x=33  in the string s
```

See also

- **printf**
- **putchar**
- **puts**

Math Libraries



These libraries have been adapted from the standard set.

included with #include <math.h>

Single Precision Math Functions

atanf
cosf
sinf
tanf
tanhf
frexpf
modff
ceilf
fabsf
floorf

atanf

Syntax

```
#include <math.h>

float atanf( float arg );
```

The function atan() returns the arc tangent of *arg*, which will be in the range [-pi/2, +pi/2].

Related topics:

- cosf
- sinf
- tanf

ceilf

Syntax

```
#include <math.h>


float ceilf( float num );
```

The ceilf() function returns the smallest integer no less than *num*. For example,

```
   y = 6.04;
  x = ceil( y );
```

would set x to 7.0.

Related topics:
floor
fmod

cosf

Syntax

```
#include <math.h>


float cosf( float arg );
```

The cos() function returns the cosine of *arg*, where *arg* is expressed in radians. The return value of cos() is in the range [-1,1]. If *arg* is infinite, cos() will return NAN and raise a floating-point exception.

Related topics:

- cosf
- sinf
- tanf

fabsf

Syntax

```
#include <math.h>


float fabsf( float arg );
```

The function fabsf() returns the absolute value of *arg*.

Related topics:
abs
fmod

floorf

Syntax

```
#include <math.h>


float floorf( float arg );
```

The function floorf() returns the largest integer not greater than *arg*. For example,

```
   y = 6.04;
  x = floorf( y );
```

would result in x being set to 6.0.

Related topics:
ceil
fmod

frexpf

Syntax

```
#include <math.h>


float frexpf( float num, int* exp );
The function frexpf() is used to decompose num into two parts:
a mantissa between 0.5 and 1 (returned by the function) and an
exponent returned as exp. Scientific notation works like this:

   num = mantissa * (2 ^ exp)
```

Related topics:
ldexp
modf

modf

Syntax

```
#include <math.h>


float modf( float num, float *i );
```

The function modf() splits *num* into its integer and fraction parts. It returns the fractional part and loads the integer part into *i*.

Related topics:
frexp
ldexp

sinf

Syntax

```
#include <math.h>


float sin( float arg );

The function sin() returns the sine of arg, where arg is given
in radians. The return value of sin() will be in the range
[-1,1]. If arg is infinite, sin() will return NAN and raise a
floating-point exception.
```

Related topics:

- cosf
- sinf
- tanf

tanf

Syntax

```
#include <math.h>


float tanf( float arg );
```

The tanf() function returns the tangent of *arg*, where *arg* is given in radians. If *arg* is infinite, tan() will return NAN and raise a floating-point exception.

Related topics:

- cosf
- sinf
- tanf

tanhf

Syntax

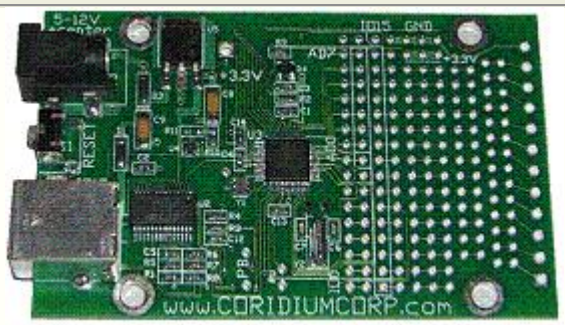
```
#include <math.h>

float tanhf( float arg );
```

The function tanh() returns the hyperbolic tangent of *arg* .

Related topics:

- cosf
- sinf
- tanf

mem Libraries



These libraries have been adapted from the standard set.

included with #include "mem.h"

Memory Functions

- memchr
- memcmp
- memcpy
- memmove
- memset

memchr

Syntax

```
#include "mem.h"


void *memchr( const void *buffer, int ch, unsigned int count );
```

The memchr() function looks for the first occurrence of *ch* within *count* characters in the array pointed to by *buffer*. The return value points to the location of the first occurrence of *ch*, or **NULL** if *ch* isn't found. For example:

```
 char names[] = "Alan Bob Chris X Dave";
if( memchr(names,'X',strlen(names)) == NULL )
  printf( "Didn't find an X\n" );
else
  printf( "Found an X\n" );
```

Related topics:

memcmp
memcpy
strstr

memcmp

Syntax

```
#include "mem.h"


int memcmp( const void *buffer1, const void *buffer2, unsigned int count );
The function memcmp() compares the first count characters of buffer1 and
buffer2      .
The return values are as follows:
```

Value	Explanation
less than 0	buffer1 is less than buffer2
equal to 0	buffer1 is equal to buffer2
greater than 0	buffer1 is greater than buffer2

Related topics:

memchr
memcpy
memset
strcmp

memcpy

Syntax

```
#include "mem.h"


void *memcpy( void *to, const void *from, unsigned int count );
```

The function memcpy() copies *count* characters from the array *from* to the array *to*.

The return value of memcpy() is *to*. The behavior of memcpy() is undefined if *to* and *from* overlap.

Related topics:

- **memchr**
- **memcmp**
- **memmove**
- **memset**
- **strcpy**
- **strlen**
- **strncpy**

memmove

Syntax

```
#include "mem.h"


void *memmove( void *to, const void *from, unsigned int count );
```

The memmove() function is identical to **memcpy**(), except that it works even if *to* and *from* overlap.

Related topics:

memcpy
memset

memset

Syntax

```
#include "mem.h"


 void* memset( void* buffer, int ch, unsigned int count );
```

The function memset() copies *ch* into the first *count* characters of *buffer*, and returns *buffer*.
memset() is useful for intializing a section of memory to some value. For example, this command:

```
 const int ARRAY_LENGTH;
char the_array[ARRAY_LENGTH];
...
// zero out the contents of the_array
memset( the_array, '\0', ARRAY_LENGTH );
```

...is a very efficient way to set all values of the_array to zero.

Related topics:
- **memcmp**
- **memcpy**
- **memmove**

string Libraries

These libraries have been adapted from the standard set where possible or from subsets as is the case with printf.

included with #include "string.h"

String Functions

- strcat
- strchr
- strcmp
- strcpy
- strcspn
- strlen
- strncat
- strncmp
- strncpy
- strstr
- strrchr
- islower
- isupper
- toupper
- tolower

islower

Syntax

```
#include "string.h"


int islower( int ch );
```

The islower() function returns non-zero if its argument is a lowercase letter. Otherwise, zero is returned.

Related topics:

isupper

isupper

Syntax

```
#include "string.h"


int isupper( int ch );
```

The isupper() function returns non-zero if its argument is an uppercase letter. Otherwise, zero is returned.

Related topics:

- **islower**
- **tolower**

strcat

Syntax

```
#include "string.h"


 char *strcat( char *str1, char *str2 );
```

The strcat() function concatenates *str2* onto the end of *str1*, and returns *str1*. For example:

```
   printf( "Enter your name: " );
  scanf( "%s", name );
  title = strcat( name, " the Great" );
  printf( "Hello, %s\n", title );
```

Note that strcat() does not perform bounds checking, and thus risks overrunning *str1* or *str2*.
For a similar (and safer) function that includes bounds checking, see **strncat()**.

Related topics:

- **strchr**
- **strcmp**
- **strcpy**
- **strncat**

strchr

Syntax

```
#include "string.h"


 char *strchr( char *str, int ch );
```

The function strchr() returns a pointer to the first occurence of *ch* in *str*, or **NULL** if *ch* is not found.

Related topics:

- **strcat**
- **strcmp**
- **strcpy**
- **strlen**
- **strncat**
- **strncmp**
- **strncpy**
- **strpbrk**
- **strspn**
- **strstr**

strcmp

Syntax

```
#include "string.h"

int strcmp( char *str1, char *str2 );
```

The function strcmp() compares *str1* and *str2*, then returns:

Return value	Explanation
less than 0	"str1" is less than "str2"
equal to 0	"str1" is equal to "str2"
greater than 0	"str1" is greater than "str2"

For example:

```
printf( "Enter your name: " );
scanf( "%s", name );
if( strcmp( name, "Mary" ) == 0 ) {
  printf( "Hello, Dr. Mary!\n" );
}
```

Note that if *str1* or *str2* are missing a null-termination character, then strcmp() may not produce valid results. For a similar (and safer) function that includes explicit bounds checking, see **strncmp()**.

Related topics:

memcmp
strcat
strchr

strcpy

Syntax

```
#include "string.h"


 char *strcpy( char *to, char *from );
```

The strcpy() function copies characters in the string *from* to the string *to*, including the
null termination. The return value is *to*.

Note that strcpy() does not perform bounds checking, and thus risks overrunning *from* or *to*.
For a similar (and safer) function that includes bounds checking, see **strncpy()**.

Related topics:

- **memcpy**
- **strcat**
- **strchr**
- **strcmp**
- **strncmp**
- **strncpy**

strcspn

Syntax

```
#include "string.h"

unsigned int strcspn( char *str1, char *str2 );
```

The function strcspn() returns the index of the first character in *str1* that matches any of
the characters in *str2*.

Related topics:

- **strrchr**
- **strstr**

-

strlen

Syntax

```
#include "string.h"


int strlen( char *str );
```

The strlen() function returns the length of *str* (determined by the number of characters before
null termination).

Related topics:

- **memcpy**
- **strchr**
- **strcmp**
- **strncmp**

strncat

Syntax

```
#include "string.h"


char *strncat( char *str1, char *str2, unsigned int count );
```

The function strncat() concatenates at most *count* characters of *str2* onto *str1*, adding a
null termination. The resulting string is returned.

Related topics:

- **strcat**
- **strchr**
- **strncmp**
- **strncpy**

strncmp

Syntax

```
#include "string.h"


int strncmp( char *str1, char *str2, unsigned int count );
```

The strncmp() function compares at most *count* characters of *str1* and *str2*. The return value is as follows:

Return value	Explanation
less than 0	"str1" is less than "str2"
equal to 0	"str1" is equal to "str2"
greater than 0	"str1" is greater than str2"

If there are less than *count* characters in either string, then the comparison will stop after the first null termination is encountered.

Related topics:

- **strchr**
- **strcmp**
- **strcpy**
- **strlen**
- **strncat**
- **strncpy**

strncpy

Syntax

```
#include "string.h"


char *strncpy( char *to, char *from, unsigned int count );
```

The strncpy() function copies at most *count* characters of *from* to the string *to*. If from has less than *count* characters, the remainder is padded with '\0' characters. The return value is the resulting string.

Related topics:

- **memcpy**
- **strchr**
- **strcpy**
- **strncat**
- **strncmp**

strrchr

Syntax

```
#include "string.h"


char *strrchr( char *str, int ch );
```

The function strrchr() returns a pointer to the last occurrence of *ch* in *str*, or **NULL** if no match is found.

Related topics:

- strcspn
- strstr

strstr

Syntax

```
#include "string.h"


char *strstr( char *str1, char *str2 );
```

The function strstr() returns a pointer to the first occurrence of *str2* in *str1*, or **NULL** if no match is found. If the length of *str2* is zero, then strstr() will simply return *str1*.

For example, the following code checks for the existence of one string within another string:

```
  char* str1 = "this is a string of characters";
 char* str2 = "a string";
 char* result = strstr( str1, str2 );
 if( result == NULL ) printf( "Could not find '%s' in '%s'\n", str2, str1
);
 else printf( "Found a substring: '%s'\n", result );
```

When run, the above code displays this output:

```
  Found a substring: 'a string of characters'
```

Related topics:

- **memchr**
- **strchr**
- **strcspn**
- **strrchr**

tolower

Syntax

```
#include "string.h"

int tolower( int ch );
```

The function tolower() returns the lowercase version of the character *ch*.

Related topics:
- **isupper**
- **toupper**

toupper

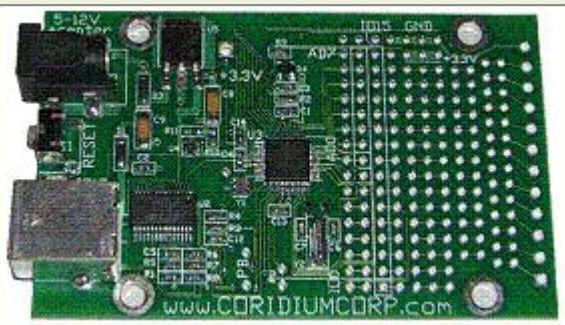
Syntax

```
#include "string.h"


int toupper( int ch );
```

The toupper() function returns the uppercase version of the character *ch*.

Related topics:
tolower

string convert Libraries



These libraries have been adapted from the standard set. And are included in coridium.h

String Conversion Functions

- atoh
- atoi

atoh

Syntax

```
#include "coridium.h"


int atoh( char *str );
```

The atoi() function converts *str* into an integer, and returns that integer. *str* should start with whitespace or some sort of number, and atoi() will stop reading from *str* as soon as a non-numerical character has been read. For example:

```
  int i;
 i = atoh( "200" );
 i = atoh( "0x200" );
 i = atoh( "   200." );
 i = atoh( "   200+22" );
 i = atoh( "   200 bottles of beer on the wall" );
```

All five of the above assignments to the variable *i* would result in it being set to 512.

If the conversion cannot be performed, then atoh() will return zero:

```
  int i = atoh( " does not work: " );  // results in i == 0
```

You can use **sprintf**() to convert a number into a string.

Related topics:

atoh
sprintf

atoi

Syntax

```
#include "coridium.h"


int atoi( char *str );
```

The atoi() function converts *str* into an integer, and returns that integer. *str* should start with whitespace or some sort of number, and atoi() will stop reading from *str* as soon as a non-numerical character has been read. For example:

```
  int i;
 i = atoi( "512" );
 i = atoi( "512.035" );
 i = atoi( "   512.035" );
 i = atoi( "   512+34" );
 i = atoi( "   512 bottles of beer on the wall" );
```

All five of the above assignments to the variable *i* would result in it being set to 512.

If the conversion cannot be performed, then atoi() will return zero:

```
  int i = atoi( " does not work: 512" );  // results in i == 0
```

You can use **sprintf**() to convert a number into a string.

Related topics:

- **atoh**
- **sprintf**

cor_min_io Library

Some simple input and output routines provide communication from the ARMmite/ARMexpress back to the PC. These routines adopted from ARMbasic are usually smaller and simpler than some similar routines available to GCC.

This library has been superceeded by printf and the communication library, it is been kept for backwards compatability.

cor_min_io Library

- **getCh**
- **getStr**
- **printDec**
- **printHex**
- **printStr**
- **str2dec**

getCh

Syntax

inr getCh ( );

Description

getCh() will return a single character (0-255) if one is available in the input buffer.

If no character is available then -1 is returned.

This routine unlike getStr returns immediately whether a character is available or not.

Example

```
while (1) {
   ch = getCh();
   if (ch != -1) break;
}
```

See also

- **getStr**

getStr

Syntax

void getStr(char *stringPointer, int maxLen);

Description

getStr will accept a string from the USB port on the ARMmite or SIN on the ARMexpress.

The string will be saved into the stringPointer char array. A CR (carriage return) will terminate the string and a 0 (null) character will be added to the array and the routine will return.

maxLen will limit the number of characters copied into stringPointer. If that limit is hit, then the routine will return.

getStr will wait for input indefinately.

Example

```
while (1) {
  printStr("enter option:");
  getStr(instring, 100);
  inval = str2dec(instring);

  switch (inval) {
    case 1:
```

See also

- getCh
- getStr

printDec

Syntax

printDec (*expression*);

Description

This sends the decimal string representation of the integer *expression* to the USB/serial port. The size of the result string depends on the integer type passed, it's not fixed.

Example

```
printDec ( 255 );    // will display 255
```

See also

- printCh
- printDec
- printHex
- printStr

printHex

Syntax

void printHex (*expression*);

Description

This sends the hexadecimal string representation of the integer *expression* to the USB/serial port. Hexadecimal values contain 0-9, and A-F. The size of the result string depends on the integer type passed, it's not fixed.

Example

```
printHex ( 255 );    // will display FF
```

See also

- printCh
- printDec
- printHex
- printStr

printStr

Syntax

void printStr (char * *pointer*);

Description

This sends the string pointed to by *pointer* to the USB/serial port. The size of the result string depends on the integer type passed, it's not fixed.

Example

```
printStr ("hi mom");    // will display    hi mom
```

See also

- printCh
- printDec
- printHex
- printStr

str2dec

Syntax

int str2dec (char * *stringPointer*);

Description

str2dec returns the numeric value of the string pointed to by stringPointer. The default conversion uses decimal, but it will convert hex values preceeded by 0x

Example

```
while (1) {
 printStr("enter option:");
 getStr(instring, 100);
 inval = str2dec(instring);

 switch (inval) {
  case 1:
...
```

See also

- **getStr**

str2hex

Syntax

int str2hex (char * *stringPointer*);

Description

str2hex returns the numeric value of the string pointed to by stringPointer. The default conversion uses hexadecimal.

Example

```
while (1) {
 printStr("enter option:");
 getStr(instring, 100);
 inval = str2hex(instring);

 switch (inval) {
  case 1:
...
```

See also

- **getStr**

Sample Programs

Csample.c

This sample program exercises all the cor_hwlib components.

It also includes the source for a simple break point facility, that allows the user to read and write memory and registers. When a break is hit memory can be listed with

@hex-num // will dump memory starting at that location

@ // will continue the dump from the end of the last location.

>hex-num1 hex-num2 // write hex-num2 into location hex-num1

The source for the breakpoint is now part of Csample.c, in the hopes it may be expanded by the user community.

Xsample.c

This is a bare bones example program that does the minimum to get a program started

Source code for cor_hwlib.c is available for purchase from Coridium.

Pin Controls

Pin Controls

- DIR
- HIGH
- IN
- INPUT
- LOW
- OUTPUT

DIR

Syntax

int DIR (int *expression*) ; // defined for port 0

int DIRx(int *pin*); // for pin 0-31 on port 0, pin 32-63 on port 1, pin 64-95 on port 2 ...

Description

DIR (expression) can be used to read the direction of the up to 32 configurable Port 0 pins. If DIR (expression) is 1 then the corresponding pin is an output. If the value is 0 then that pin is an input.

With the SuperPRO and PROplus, NXP has added more configurable port pins. We are adopting their convention for control of those pins. FIO0DIR, FIO1DIR, FIO2DIR, and FIO4DIR control the input/output status. Writing 1 to a bit will make it an output. More details in the **NXP user manuals.**

Example

```
// Set pin 4 as an input
INPUT(4);

// Check the direction of pin 4
dir4 = DIR(4);
```

See also

- **DIR**
- **HIGH**
- **IN**
- **INPUT**
- **LOW**
- **OUTPUT**

HIGH

Syntax

void HIGH (int *pin); // defined for port 0*

void HIGHx(int pin); // for pin 0-31 on port 0, pin 32-63 on port 1, pin 64-95 on port 2 ...

Description

HIGH will set the Port 0 pin corresponding to *expression* to a positive value (3.3V).

HIGH does not change the direction of the port 0 pin, ie. it must be set to an OUTPUT before HIGH has any affect.

With the SuperPRO and PROplus, NXP has added more configurable port pins. We are adopting their convention for control of those pins. FIO0SET, FIO1SET, FIO2SET, and FIO4SETwill set an output high when written with a 1. More details in the **NXP user manuals.**

Example

```
//  set pins 0 to 7 to output and low or to 0 V
for (i=0;i<8; i++) {
  OUTPUT( i );
  LOW( i );
}

for (i=0;i<8; i++) {
  WAIT(1000);
  OUTPUT( i );
  HIGH ( i );          // set each pin HIGH one after the other every second
}
```

See also

- **DIR**
- **HIGH**
- **IN**
- **INPUT**
- **LOW**
- **OUTPUT**

IN

Syntax

int IN (int *pin*); // defined for port 0

int INx(int *pin*); // for pin 0-31 on port 0, pin 32-63 on port 1, pin 64-95 on port 2 ...

Description

IN returns the value on the Port 0 pin corresponding to expression.

0 is returned for 0V and -1 for 2.5V or above. Why -1 and 0? NOT(~) 0 is equal to -1.

With the SuperPRO and PROplus, NXP has added more configurable port pins. We are adopting their convention for control of those pins. FIO0DIR, FIO1DIR, FIO2DIR, and FIO4DIR control the input/output status. Writing 1 to a bit will make it an output. More details in the**>NXP user manuals.**

Example

```
printf("Pin 15 is ");
INPUT(15);                    // make pin 15 an INPUT

printf("%d\n",IN(15));        // now read the value on that pin
```

See also

- **DIR**
- **HIGH**
- **IN**
- **INPUT**
- **LOW**
- **OUTPUT**

INPUT

Syntax

void INPUT (int *pin*); // defined for port 0

void INPUTx(int *pin*); // for pin 0-31 on port 0, pin 32-63 on port 1, pin 64-95 on port 2 ...

Description

INPUT will set the Port 0 pin corresponding to *expression* to an input.

With the SuperPRO and PROplus, NXP has added more configurable port pins. We are adopting their convention for control of those pins. FIO0DIR, FIO1DIR, FIO2DIR, and FIO4DIR control the input/output status. Writing 1 to a bit will make it an output. More details in the **NXP user manuals.**

Example

```
printf("Pin 15 is ");
 INPUT(15);                    // make pin 15 an INPUT

 printf("%d\n",IN(15));              // now read the value on that pin
```

See also

- **DIR**
- **HIGH**
- **IN**
- **INPUT**
- **LOW**
- **OUTPUT**

LOW

Syntax

void LOW (int *pin*); // defined for port 0

void LOWx(int *pin*); // for pin 0-31 on port 0, pin 32-63 on port 1, pin 64-95 on port 2 ...

Description

LOW will set the Port 0 pin corresponding to *expression* to a negative value (0V) and then set it to an output.

With the SuperPRO and PROplus, NXP has added more configurable port pins. We are adopting their convention for control of those pins. FIO0CLR, FIO1CLR, FIO2CLR, and FIO4CLR will set an output low when written with a 1. More details in the **NXP user manuals.**

Example

```
//  set pins 0 to 7 to output and low or to 0 V
for (i=0;i<8; i++) {
  OUTPUT( i );
  LOW( i );
}

for (i=0;i<8; i++) {
  WAIT(1000);
  OUTPUT( i );
  HIGH ( i );          // set each pin HIGH one after the other every second
}
```

See also

- **DIR**
- **HIGH**
- **IN**
- **INPUT**
- **LOW**
- **OUTPUT**

OUTPUT

Syntax

void OUTPUT (int *pin*); // defined for port 0

void OUTPUTx(int *pin*); // for pin 0-31 on port 0, pin 32-63 on port 1, pin 64-95 on port 2 ...

Description

OUTPUT will set the Port 0 pin corresponding to *expression* to an output.

With the SuperPRO and PROplus, NXP has added more configurable port pins. We are adopting their convention for control of those pins. FIO0DIR, FIO1DIR, FIO2DIR, and FIO4DIR control the input/output status. Writing 1 to a bit will make it an output. More details in the **NXP user manuals.**

Example

```
printf("LED on\n");
OUTPUT(15);
LOW(15);
```

See also

- **DIR**
- **HIGH**
- **IN**
- **INPUT**
- **LOW**
- **OUTPUT**

Hardware Function List

A

- AD

B

- BAUD_

C

- configAD
- COUNT

D

- DIGITAL

F

- FREQOUT

H

- HWPWM

I

- I2CIN
- I2COUT
- I2CSPEED

O

- OWIN
- OWOUT

P

- PULSIN
- PULSOUT
- PWM

R

- RCTIME
- RXD

S

- SERIN
- SERINtimeout
- SEROUT
- SHIFTIN
- SHIFTOUT
- SPIIN
- SPIMODE
- SPIOUT

T

- TIMER
- TXD

AD

Syntax

extern int AD(int channel);

Description

ARMmite, ARMexpress LITE and ARMweb

AD will return 0..65472 that corresponds to the voltage on the pin corresponding to *expression* . The value returned will have the top 10 bits being significant. (bits 5..0 will be 0). 0 would be read for 0V and 65472 for 3.3V.

An analog conversion on pin *expression* is performed. This process takes less than 6 usec.

Dual Use AD pins

ARMmite version

On reset or power up the AD pins are configured as AD inputs. To change those to digital IOs, the user must individually specify a switch to digital using DIGITAL(IOpin) which is a #define for IOpins 16-23.

ARMexpress LITE version

The ARMexpress LITE supports up to 6 channels of AD converters.

On the ARMexpress LITE and ARMweb these pins are configured as digital IOs at reset, but must be switched to AD using configAD(channel) which is a #define for channels 0-7.

AD(0)	IO(7)
AD(1)	IO(10)
AD(2)	IO(8)
AD(3)	not available
AD(4)	not available
AD(5)	IO(9)
AD(6)	IO(11)
AD(7)	IO(12)

Example

```
temperature = AD(0) >> 6;          // read a raw temperature voltage
```

See also

- TXD
- RXD

configAD

Syntax

#define configAD(int channel);

Description

Dual Use AD pins

ARMexpress LITE version

The ARMexpress LITE supports up to 6 channels of AD converters.

On the ARMexpress LITE and ARMweb these pins are configured as digital IOs at reset, but must be switched to AD using configAD(channel) which is a #define for channels 0-7.

AD(0)	IO(7)
AD(1)	IO(10)
AD(2)	IO(8)
AD(3)	not available
AD(4)	not available
AD(5)	IO(9)
AD(6)	IO(11)
AD(7)	IO(12)

Example

```
configAD(1);          // switch IO(10) to AD(1)
```

See also

- **TXD**
- **RXD**

COUNT

Syntax

int COUNT(int pin, int duration);

Description

Count the number of pulses low-high-low or high-low-high on *pin* over a duration of *milliseconds*, returning the value to *variable*.

Example

```
printf("\n count pulses on pin 0 for 5 seconds\n");

printf("%d pulses happened\n",COUNT(0, 5000));
```

See also

- **RCTIME**

DIGITAL

Syntax

#define DIGITAL(int IOpin)

Description

Dual Use AD pins

ARMmite version

On reset or power up the AD pins are configured as AD inputs. To change those to digital IOs, the user must individually specify a switch to digital using DIGITAL(IOpin) which is a #define for IOpins 16-23.

Example

```
DIGITAL(4);          // switch AD(4) to a digital pin accessed at IO(20)
```

See also

- TXD
- RXD

HWPWM

Syntax

void HWPWM (int *cycletime*,int *itemCnt*, int **hightimeList*);

Description

ARMmite and Wireless ARMmite version

The ARMmite supports up to 8 channels of hardware driven PWM. The IO direction of the pin will be set to output. Once programmed these will continue to generate the specified PWM until re-programmed or reset.

Cycletime is in microseconds, is the time for a single PWM cycle. *Hightimes* are also in microseconds and represent the amount of time during the cycle that the corresponding outputs are high. It is assumed, but not enforced that cycletimes for all channels will be the same.

channel1	IO(0)
channel2	IO(1)
channel3	IO(2)
channel4	IO(3)
channel5	IO(4)
channel6	IO(9)
channel7	IO(10)
channel8	IO(11)

ARMmite PRO version

The ARMmite PRO also supports up to 8 channels of hardware driven PWM. The IO direction of the pin will be set to output. Once programmed these will continue to generate the specified PWM until re-programmed or reset.

Cycletime is in microseconds, is the time for a single PWM cycle. *Hightimes* are also in microseconds and represent the amount of time during the cycle that the corresponding outputs are high. It is assumed, but not enforced that cycletimes for all channels will be the same.

channel1	IO(0)
channel2	IO(1)
channel3	IO(8)
channel4	IO(5)
channel5	IO(14)
channel6	IO(10)
channel7	IO(11)
channel8	IO(3)

ARMexpress LITE version

The ARMexpress LITE supports up to 6 channels of hardware driven PWM. The IO direction of the pin will be set to output. Once programmed these will continue to generate the specified PWM until re-programmed or reset. 2 of the channels are not available on the pins.

Cycletime is in microseconds, is the time for a single PWM cycle. *Hightimes* are also in microseconds and represent the amount of time during the cycle that the corresponding outputs are high. It is assumed, but not enforced that cycletimes for all channels will be the same.

channel1	IO(5)
channel2	IO(6)
channel3	IO(3)
channel4	not available
channel5	IO(14)
channel6	not available
channel7	IO(13)
channel8	IO(15)

SuperPRO version

The PROplus and SuperPRO support up to 6 channels of hardware driven PWM. The IO direction of the pin will be set to output. Once programmed these will continue to generate the specified PWM until re-programmed or reset.

Cycletime is in microseconds, is the time for a single PWM cycle. *Hightimes* are also in microseconds and represent the amount of time during the cycle that the corresponding outputs are high. It is assumed, but not enforced that cycletimes for all channels will be the same.

channel1	P2.0
channel2	P2.1
channel3	P2.2
channel4	P2.3
channel5	P2.4
channel6	P2.5

The LPC17xx series processors also have an additional 6 channels designed to drive motors. See details in the Motor PWM Control chapter of the NXP LPC17xx User Manual. Also these pins can be re-assigned as selected by the PINSEL registers.

Example

```
times[0] = -1
times[1] = 750
times[2] = 100
HWPWM (1000, 3, times)      'generate 1KHz 75% and 10% signals on pins 1,2

times[0] = 2000
times[1] = 1000
times[2] = 500
HWPWM (4000, 3, times)      'gen 250 Hz 50%, 25% and 12.5% signals on pins 0,1,2
```

See also

- **FREQOUT**
- **PWM**

I2CIN

Syntax

void I2CIN (int *sda_pin*, int *scl_pin*, int *slaveADDR*, int opt1, int opt2, int opt3, int opt4, int opt5, int *cnt* , char **inList*);

Description

I2CIN will read a series of bytes from an I2C slave device. *sda_pin* is any expression defining the SDA pin to use. *scl_pin* will be designated the SCL pin. *slaveADDR* will select a device on the I2C bus.

Up to 5 optional byte values may be writen out prior to reading the *inList* . If the *optX* value is -1 then it will not be sent out. Most i2c devices have a sub-address field that is written immediately before reading. This field would be held in opt1. For devices that do not have a sub-address, opt1 can be set to -1. In this case, no I2C write is performed before the I2CIN. For some slow i2c devices, they can not respond to a write of sub-address, immediately followed by a read of the device. For these slow devices (often implemented with a slower micro-processor), it is necessary to do seperate I2COUT and I2CIN with a delay in between.

After any write of the *optX* ,a series of *cnt* bytes will be read from the slave to fill the character array *inList* .

I2C is a byte oriented bus, so each transaction will either send a byte value (0 to 255) or receive a byte for each element of the *inList* .

Data is shifted in at 350 Kbits/sec.

Example

```
case 41:   // test EEPROM 24LC02 on pins 0 == SDA and 1 == SCL
 shortMessage[0] = 0;  // address into EEPROM
 for (i=1; i<8; i++) shortMessage[i] = 0x30+i; // set shortMessage to "1234567"
 present = I2COUT (0, 1, 0xA0,8, shortMessage);

 if (present == 0) printf("NO i2c device ***\n"); else printf("i2c device found\n");
 I2CIN(0, 1, 0xA0, 0, -1, -1, -1, -1, 7, shortResponse);
```

See also

- I2COUT

I2COUT

Syntax

int I2COUT (int *sda_pin*, int *scl_pin*, int *slaveADDR* , int *cnt*, char **outList*);

// I2COUT returns TRUE if the device was present and responding

Description

I2COUT will send a series of bytes from an I2C slave device. *sda_pin* is any expression defining the SDA pin to use. *scl_pin* will be designated the SCL pin. *slaveADDR* will select a device on the I2C bus.

After that *cnt* bytes will be written to the slave from the character array *outList*.

I2C is a byte oriented bus, so each transaction will send a byte values (0 to 255) to an I2C slave. If the value from an *expr* ession in the *OutputList* is larger than 8 bits, the MSBs will be truncated.

If a I2C device responds the function returns 1, else 0.

Data is shifted out at 350 Kbits/sec.

Example

```
case 41:   // test EEPROM 24LC02 on pins 0 == SDA and 1 == SCL
 shortMessage[0] = 0;
 for (i=1; i<8; i++) shortMessage[i] = 0x30+i; // set shortMessage to "1234567"
 present = I2COUT (0, 1, 0xA0, 8, shortMessage);
```

See also

- **I2CIN**

I2CSPEED

Syntax

void I2CSPEED (int *Kbits);*

Kbits= 400 | 100 | 50

Description

The default speed for I2C operations is approx 350 Kbits, which is acceptable to most modern I2C chips.

In order to support older chips or longer cable runs, the I2C operations can be slowed down to 100 Kbits.

A very slow 50 Kb rate is also supported.

Example

```
I2CSPEED (100);                 // slow down the operation
```

See also

- **I2CIN**
- **I2COUT**

OWIN

Syntax

void OWIN (int *pin*, int *out1*,int *out2*,int *out3*,int *out4*,int *out5*, int *out6*, int *cnt*, char **inList*) ;

Description

OWIN begins with a RESET/Presence sequence on the designated *Pin*.

Then upto 6 *Output* bytes will be transfered to the device to select the command. Byte values are sent out, if the value is -1, then that byte is not sent.

Following that cnt bytes will be read into the character array *inList* .

The bit order for the 1-Wire device is assumed to be LSB (bit 0) first.

Example

```
present = OWOUT(7, 7, shortMessage);
if (present) printf("one wire device found\n");
else {printf("*** NO one wire device ***\n"); break;}

OWIN(7, 0xCC, 0xAA, 0, -1, -1, -1, 4, shortResponse);

for (i=0; i<4; i++) if (shortMessage[i+3] != shortResponse[i]) break;
```

See also

- OWOUT

OWOUT

Syntax

int OWOUT (int *pin*, int *cnt*, char **outList*) ;

// OWOUT returns TRUE if the device was present and responding

Description

OWOUT begins with a RESET/Presence sequence on the designated *Pin*.

Following that *cnt*bytes of the character array *outList*will be sent to the device.

The bit order for the 1-Wire device is assumed to be LSB (bit 0) first.

If a one-wire device responds the function returns 1, else 0.

Example

```
case 40:   // test the EEPROM of a DS2430 on pin 7
 shortMessage[0] = 0xCC;
 shortMessage[1] = 0x0F;
 shortMessage[2] = 0x00;
 shortMessage[3] = 0x44;
 shortMessage[4] = 0x11;
 shortMessage[5] = 0x22;
 shortMessage[6] = 0xBE;

 present = OWOUT(7, 7, shortMessage);
 OWIN(7, 0xCC, 0xAA, 0, -1, -1, -1, 4, shortResponse);
```

See also

- **OWIN**

PULSIN

Syntax

int PULSIN (int pin, int state);

Description

Measure an input pulse on *pin* at *level*, returning the value.

The IO direction of *pin* will be set to input.

If *pin* is already at *level* when the function is called it will wait to a transition to the opposite *level*.

The function will wait 1 second for *pin* to go to *level*. The length of time is measured in microseconds(us). The minimum pulse that can be measured is 1 microseconds. If *pin* does not go to level or remains at *level* longer than 1 second 0 is returned.

Example

```
case 32:
 prinf("\n Measure PULSIN on pin 0, four times as low pulse then high pulse\n");
 for (i=0; i<4; i++) {
  printf ("%d\n", PULSIN (0, 0) );
 }
 for (i=0; i<4; i++) {
  printf ("%d\n", PULSIN (0, 1) );
 }
```

See also

- **RCTIME**
- **COUNT**

PULSOUT

Syntax

void PULSOUT (int pin, int duration);

Description

Generate an output pulse on *pin* for *microseconds*.

The IO direction of *pin* will be set to output. The level of the output will be switched, driven for *microseconds*, then switched back to its initial level. The minimum pulse period is 1 microseconds.

Example

```
printf("Flash LED for 250ms off for 500ms\n");
OUTPUT(15);
HIGH(15);
for (i=0; i<20; i++) { PULSOUT (15,250000); WAIT (500);}
```

See also

- **PULSIN**

PWM

Syntax

void PWM(int pin, int duty, int duration);

Description

Generate an analog signal on *pin* for *milliseconds* with a *duty* cycle of 0 to 255.
A *duty* cycle of 255 corresponds to an output value of 100%.

The IO direction of the pin will be set to output, the PWM pulse train is output, and then the pin is set to tristate (input). If the pin is connected to an RC filter, then the voltage will stay on the capacitor for a period of time determined by the load.

Example

```
printf("Ramp thru PWM on LED\n");
 for (i=0; i<256; i++) PWM (15,255-i,20);
```

See also

- **FREQOUT**
- **PULSOUT**

RCTIME

Syntax

int RCTIME(int *pin*, int *state*) ;

Description

Measure the time which *pin* remains at *state*, returning the value to *variable*.
The length of time is measured in microseconds(us). The minimum time measured is 1 microseconds.
If *pin* is not at *level* when the function is called *variable* is set to 1.
If *pin* remains at *level* longer than 1 second *variable* is set to 0.

Example

```
    printf("\n check RCTIME into a 700ohm 0.1uF drive on 14, sense on 13\n");
    OUTPUT (14);
//  while(1) {HIGH(14);WAIT(1);LOW(14);WAIT(1);} // scope loop

    HIGH (14);
    printf("high to low ");
    WAIT(10);  // make sure line is high
    LOW(14);
    printf("%d microseconds, high to low ",RCTIME (13, 1));
    WAIT(10);  // make sure line is low
    HIGH (14);
    printf("%d microseconds, low to high\n", (RCTIME (13, 0));
```

```
 check RCTIME into a 700ohm 0.1uF drive on 14, sense on 13
high to low 89 microseconds, high to low 56 microseconds, low to high
```

See also

- **PULSIN**

RXD

Syntax

int RXD (int *pin*);

int RXD0(); // for UART0 access

int RXD1(); // for UART1 access

Description

RXD (*pin*) will receive a single byte of data that is shifted as an asynchronous serial stream. This function is similar to SERIN, but is a more efficient implementation. The baudrate for the pin should be set before using RXD, that is done by setting the SERbauds[] array.

RXD will timeout after 0.5 seconds and return -1. These routines are "bit-banged" by the processor, so the processor is consumed during these operations. Interupts are also disabled during each byte for these operations.

The 0.5 second timeout can be changed by **SERINtimeout**.

Baudrate can be upto 115.2Kb.

UART0 UART1 support-

For RXD0 data is received on the SIN pin. SIN and SOUT are always negative true. UART0 of the LPC21xx, RXD1 for UART1.

For ARMexpress modules baudrates for RXD0 can be upto 19.2 Kbaud as it is limited by the level translators. No limit for ARMmite/ARMweb Use setbaud() routine to set this baud.

Example

```
while (1) {
   ch = RXD(4);
   if (ch != -1) printf("%c", ch );
   if (ch == 'Q') break;
} // receive and echo characters from testjig
```

See also

- **setbaud**
- **TXD**
- **SERIN**

SERbauds, setbaud

Syntax

extern int SERbauds[];

void setbaud (int channel, int baudDiv); // for UART0 and UART1

Description

This array contains the baud rates for up to 32 IOs. It is used when RXD or TXD routines are called.

Baudrates can be upto 115.2 Kbaud.

The UART0, UART1 hardware channel baudrate can be set by calling-

void setbaud(0, baudDiv); // baudDiv = 15000000 / baudrate / 16 -- for example 19.2Kb is 49 for ARMmite, ARMexpress, and ARMweb

void setbaud(1, baudDiv); // baudDiv = 25000000 / baudrate / 16 -- for example 19.2Kb is 81 forSuperPRO and PROplus

For the ARMexpress SIN and SOUT are limited by the level translators to 19.2Kbaud. This last limit does not apply to the SuperPRO, PROplus, ARMmite or ARMweb.

Example

```
SERbauds[4] = 19200;
 SERbauds[3] = SERbauds[4];

 setbaud (1, 17);  // set UART1 baudrate to 56Kb
```

See also

- TXD
- RXD

SERIN

Syntax

void SERIN (int *pin,* int *baudrate,* int *posTrue,* char *cnt,* char **inList*);

Description

SERIN receives *cnt* bytes as asynchronous serial data on *pin* at a *baudrate* and saves the data into *inList* .

PosTrue if set to 0 then the data is inverted.

inList is a pointer to a character array.

SERIN will timeout after 0.5 seconds and return -1 and 255 in the next item in the *inList* before the timeout. These routines are "bit-banged" by the processor, so the processor is consumed during these operations. Interupts are also disabled during each byte for these operations.

The 0.5 second timeout can be changed by **SERINtimeout**.

Baudrates can be upto 115.2 Kbaud for all pins.

For UART0 support use-

int getline(char *line, int max_len) ;

For ARMexpress modules baudrates for RXD0 can be upto 19.2 Kbaud as it is limited by the level translators. No limit for ARMmite/ARMweb Use setUART0baud() routine to set this baud.

Example

```
while (1) {
   SERIN (4,19200,1, 1, shortMessage);
   if (shortMessage[0] != 255) printCh( shortMessage[0] );
   if (shortMessage[0] == 'Q') break;
} // receive and echo characters from testjig
```

See also

- **SEROUT**
- **SERINtimeout**

SERINtimeout

Syntax

extern int SERINtimeout;

Description

The default time that SERIN waits for input on pins 0-31 is 0.5 seconds.

This may be changed using this statement. The timeout will be set in micro-seconds.

Example

```
// set the timeout shorter on for polling an LCD/keypad combination to 100 ms
SERINtimeout = 100000;
   ...

proc GetKey() {
   LCDcmd[0] = 24;
   LCDlen = 0;
   GenCRC();
   SEROUT (1,LCDbaud,1,LCDlen+3,LCDcmd);
   SERIN  (1,LCDbaud,1,3,LCDcmd);
}
```

See also

- SERIN
- SEROUT

SEROUT

Syntax

void SEROUT (int *pin,*int *baudrate,* int *posTrue,*int *cnt,* char **outList*);

Description

SEROUT sends *cnt* bytes pointed to by *outList* out as asynchronous serial data on *pin* at a *baudrate*. *PosTrue* if set to 0 then the data is inverted.

*outList*is a character array.

These routines are "bit-banged" by the processor, so the processor is consumed during these operations. Interupts are also disabled during each byte for these operations.

Baudrates can be upto 115.2 Kbaud for all pins

UART0 support-

Use printf or TXD0. The hardware serial port routines are used, so the CPU is not tied up. So when a byte is sent it is placed into the UART FIFO, but if the 16 byte FIFO is full then the CPU will wait until space is available.

For ARMexpress modules baudrates for RXD0 can be upto 19.2 Kbaud as it is limited by the level translators. No limit for ARMmite/ARMweb Use setUART0baud() routine to set this baud.

Example

```
shortMessage[0] = 0x33;
 shortMessage[1] = 0x34;
 shortMessage[2] = 0x35;

 SEROUT(3,19200, 1, 3, shortMessage);    // send 123 to the testjig serial connection
```

See also

- SERIN

SERbauds, setbaud

Syntax

extern int SERbauds[];

void setbaud (int channel, int baudDiv); // for UART0 and UART1

Description

This array contains the baud rates for up to 32 IOs. It is used when RXD or TXD routines are called.

Baudrates can be upto 115.2 Kbaud.

The UART0, UART1 hardware channel baudrate can be set by calling-

void setbaud(0, baudDiv); // baudDiv = 15000000 / baudrate / 16 -- for example 19.2Kb is 49 for ARMmite, ARMexpress, and ARMweb

void setbaud(1, baudDiv); // baudDiv = 25000000 / baudrate / 16 -- for example 19.2Kb is 81 forSuperPRO and PROplus

For the ARMexpress SIN and SOUT are limited by the level translators to 19.2Kbaud. This last limit does not apply to the SuperPRO, PROplus, ARMmite or ARMweb.

Example

```
SERbauds[4] = 19200;
 SERbauds[3] = SERbauds[4];

 setbaud (1, 17);  // set UART1 baudrate to 56Kb
```

See also

- TXD
- RXD

SHIFTIN

Syntax

void SHIFTIN (int *in_pin*, int *clk_pin*, int *mode*, int *cnt*, int **wordList* , int *bitLengths*) ;

Description

SHIFTIN has been kept as a compatable function with PBASIC. It can be used for devices that are not covered by SPI, I2C or 1-Wire. Data is shifted in on *in_pin*, and a positive clock is sent on *clk_pin* for each bit.

While most other hardware functions use bytes, SHIFTIN is oriented for bit control. The length of each variable defines the number of bits that will be shifted out (2 - 32). For each word *bitLengths* bits will be shifted in.

- Mode = 0 data is shifted in MSB first, and sampling starts before the first clock pulse
- Mode = 1 data is shifted in LSB first, and sampling starts before the first clock pulse
- Mode = 2 data is shifted in MSB first, and sampling starts before the second clock pulse
- Mode = 3 data is shifted in LSB first, and sampling starts before the second clock pulse

SHIFTIN fills the integer array *wordList* . *cnt* integers will be shifted in.

Data is shifted in at 600 Kbits/sec.

Example

```
printf("FPU SHIFT test\n");     for (i= 0; i<10; i++) wordMessage[i]= 0xFF;
 wordMessage [i] = 0;
 SHIFTOUT (14,15,MSBFIRST,11,wordMessage, 8); // reset FPU
 WAIT (10);    wordMessage [0] = 0xF0;       // sync character
 SHIFTOUT (14,15,MSBFIRST,1,wordMessage, 8);  // sync FPU
 save_time = TIMER;
 while ((TIMER - save_time) < 15) ;    // wait 15 uSec

 SHIFTIN (14,15, MSBPRE,  1,wordMessage, 8);  // get 1 byte status back
 if (wordMessage[0] != 0x5C) {
   printf(%x "FPU found\nwordMessage[0]);
   break;
 }
```

See also

- **SHIFTOUT**
- **SPIIN**

SHIFTOUT

Syntax

void SHIFTOUT (int *out_pin*, int *clk_pin*, int *mode*, int *cnt* , int **wordList*, int *bitLengths*) ;

Description

SHIFTOUT has been kept as a compatable function with PBASIC. It can be used for devices that are not covered by SPI, I2C or 1-Wire. Data is placed on *out_pin* and the *clk_pin* is pulsed each bit.

While most other hardware functions use bytes, SHIFTOUT is oriented for bit control. The length of each variable defines the number of bits that will be shifted out (2 - 32). For each element *bitLengths* bits are shifted out.

- Mode = 0 data is shifted out LSB first
- Mode = 1 data is shifted out MSB first

SHIFTOUT uses values from the interger array *wordList*. *cnt* integers will be shifted out.

Data is shifted out of the device at 800 Kbits/sec.

Example

```
printf("FPU SHIFT test\n");     for (i= 0; i<10; i++) wordMessage[i]= 0xFF;
 wordMessage [i] = 0;
 SHIFTOUT (14,15,MSBFIRST,11,wordMessage, 8); // reset FPU
 WAIT (10);    wordMessage [0] = 0xF0;       // sync character
 SHIFTOUT (14,15,MSBFIRST,1,wordMessage, 8);  // sync FPU
 save_time = TIMER;
 while ((TIMER - save_time) < 15) ;    // wait 15 uSec

 SHIFTIN (14,15, MSBPRE,  1,wordMessage, 8);  // get 1 byte status back
 if (wordMessage[0] != 0x5C) {
  printf("%x No FPU found\n",shortResponse[0]);      break;
 }
 printf("FPU found\n");
```

See also

- **SHIFTIN**
- **SPIIN**

SPIIN

Syntax

void SPIIN (int *CS_pin*,int *in_pin,*int *clk_pin,*int *out_pin,* int *out1,* int *out2,* int *out3,* int *cnt, char *InputList*);

Description

SPIIN supports the loosely defined serial protocol used by a variety of manufacturers. The desired device is selected by asserting *CS_pin* LOW. If there is no *CS_pin* , the value should be set to -1.

In the simplest case, *in_pin* is used to input data clocked by *clk_pin*, to fill the character array *InputList* with *cnt* bytes.

In bi-directional cases, *out1..out3* byte values will be output on *out_pin* before reading the *InputList* . If not used, those should be set to -1. It is also allowable to have *in_pin* equal to *out_pin* , in which case that pin will be driven for the *out1..out3* and then converted to an input for *in_pin* .

Data is shifted in LSB first and each element of the *InputList* is filled with a byte of data. To use negative edge clocks or MSB first, the default SPImode may be changed.

Data is shifted in at 600 Kbits/sec.

Example

```
case 42:   // check MicroMega FPU status
           // clock on pin 15 and bi-directional data on 14, no CS used
 SPImode = 0; // FPU uses MSB first -- positive clock
 for (i=0; i<10; i++) shortMessage[i]= 0xFF;
 shortMessage [i] = 0;
 SPIOUT (-1,14,15,11,shortMessage);  // reset FPU
 WAIT (10);

 shortMessage [0] = 0xF0;    // sync character
 SPIOUT (-1,14,15,1,shortMessage);  // sync FPU
 save_time = TIMER;
 while ((TIMER - save_time) < 15) ;     // wait 15 uSec
 SPIIN (-1,14,15, -1,-1,-1,-1, 1,shortResponse);  // get 1 byte status back

 if (shortResponse[0] != 0x5C) {
  printf("%x No FPU found\n",shortResponse[0]);
  break;
 }
 printf("FPU found\n");
```

See also

- **SPIOUT**
- **SPImode**

SPImode

Syntax

extern char SPImode;

Description

SPImode will change the bit order and clock sense for all suceeding SPI commands or until SPImode is changed again.

- SPImode= 0 data is shifted in MSB first, and sampling starts before the first rising clock edge (positive true clock)
- SPImode= 1 data is shifted in LSB first, and sampling starts before the first rising clock edge. (positive true clock) **(default value)**
- SPImode= 2 data is shifted in MSB first, and sampling starts before the second clock pulse
- SPImode= 3 data is shifted in LSB first, and sampling starts before the second clock pulse

Example

```
case 42:   // check MicroMega FPU status
           // clock on pin 15 and bi-directional data on 14, no CS used
 SPImode = 0; // FPU uses MSB first -- positive clock
 for (i=0; i<10; i++) shortMessage[i]= 0xFF;
 shortMessage [i] = 0;
 SPIOUT (-1,14,15,11,shortMessage);  // reset FPU
 WAIT (10);

 shortMessage [0] = 0xF0;    // sync character
 SPIOUT (-1,14,15,1,shortMessage);  // sync FPU
 save_time = TIMER;
 while ((TIMER - save_time) < 15) ;     // wait 15 uSec
 SPIIN (-1,14,15, -1,-1,-1,-1, 1,shortResponse);  // get 1 byte status back

 if (shortResponse[0] != 0x5C) {
   printf("%x No FPU found\n",shortResponse[0]);      break;
 }
 printf("FPU found\n");
```

See also

- **SPIOUT**
- **SPImode**

SPIOUT

Syntax

void SPIOUT (int *CSpin*, int *out_pin,* int *clk_pin,* int *cnt,* char **OutputList*);

Description

SPIOUT supports the loosely defined serial protocol used by a variety of manufacturers. The desired device is selected by asserting *CSpin* LOW. If there is no *cspin*, the value should be set to -1.

In the simplest case, *out_pin* is used to output data clocked by *clk_pin*, from the *OutputList*. *CSpin*, *out_pin* and *clk_pin* are left as and outputs.

Data is shifted out LSB first and each element of the *OutputList* is treated as a byte. This order can be changed with SPImode.

Data is shifted out at 800 Kbits/sec

Example

```
shortMessage [0] = 0xF3;    // get version
 SPIOUT (-1,14,15,1,shortMessage);  // sync FPU
 INPUT (14);        // allow FPU to drive this bidirectional line
 while (IN(14));       // wait for FPU to drive that line low
 shortMessage [0] = 0xF2;    // get string back
 SPIOUT (-1,14,15,1,shortMessage);  // sync FPU
 save_time = TIMER;
 while ((TIMER - save_time) < 15) ;     // wait 15 uSec
 while (1) {
  SPIIN (-1,14,15, -1,-1,-1,-1, 1,shortResponse); // get 1 byte at a time back and print it
  if (shortResponse[0] == 0) break;
  printf("%x\n",shortResponse[0]);
 }
```

See also

- **SPIIN**

TIMER

Syntax

TIMER

Description

TIMER is a free running timer that increments every microsecond. It is readable using this keyword. You can also write to the TIMER register.

Operations that require more precise timing should use the dedicated hardware routines, as interupts that are occuring for other time functions and serial input may make times using TIMER look longer than actual.

Example

```
int save_time;  // must be signed to handle roll-over

save_time = TIMER;
while ((TIMER - save_time) < 15) ;     // wait 15 uSec
```

```
TIMER = 0;   // reset the free running timer
```

See also

- DAY
- HOUR
- MINUTE
- MONTH
- SECOND
- TIMER
- WEEKDAY
- YEAR

TXD

Syntax

void TXD (int *pin,* int *byte*); // for bit-banged version on any pin

void TXD0(int byte); // for hardware support

void TXD1(int byte); // for UART1

Description

TXD (*pin, byte*) will send a single byte of data that is shifted out as an asynchronous serial stream on *pin* . This function is similar to SEROUT, but is a more efficient implementation. The baudrate for the pin should be set before using TXD, that is done setting the SERbauds[] array.

These routines are "bit-banged" by the processor, so the processor is consumed during these operations. Interupts are also disabled during each byte for these operations.

UART0 UART1 support-

The hardware serial port routines are used, so the CPU is not tied up. So when a byte is sent it is placed into the UART FIFO, but if the 16 byte FIFO is full then the CPU will wait until space is available.

For ARMexpress modules baudrates for RXD0 can be upto 19.2 Kbaud as it is limited by the level translators. No limit for ARMmite/ARMweb Use setUART0baud() routine to set this baud.

Example

```
TXD(3,'a'); TXD(3,'b'); TXD(3,'c');  // send out   abc
```

See also

- **setbaud**
- **RXD**
- **SEROUT**

Pin Controls

Time Function Library

- DAY
- HOUR
- MINUTE
- MONTH
- SECOND
- TIMER
- WEEKDAY
- YEAR

DAY

Syntax

RTC_DAY
void setDAY (x);

Description

Function setting or returning the day of the month.
Range 1 to 28, 29, 30, or 31
(depending on the month and whether it is a leap year).

Example

```
printf("This is: ");
switch (RTC_DOW) {
 case 0: printf("Sunday "); break;
 case 1: printf("Monday "); break;
 case 2: printf("Tuesday "); break;
 case 3: printf("Wednesday "); break;
 case 4: printf("Thursday "); break;
 case 5: printf("Friday "); break;
 case 6: printf("Saturday "); break;
}
printf(" %d/%d/%d \n",RTC_MONTH,RTC_DAY,RTC_YEAR);
```

The output would look like:

```
This is Friday 4/14/2006
```

See also

- DAY
- HOUR
- MINUTE
- MONTH
- SECOND
- TIMER
- WEEKDAY
- YEAR

HOUR

Syntax

RTC_HOUR

void setHOUR(int value);

Description

RTC_HOUR returns the hour of the real time clock.
Range 0 to 23.

setHOUR(x) will change the gour of the real time clock.

Example

```
printf("%d:%02d:%02d",RTC_HOUR,RTC_MIN,RTC_SEC);
```

The output would look like:

```
13:15:30
```

See also

- DAY
- HOUR
- MINUTE
- MONTH
- SECOND
- TIMER
- WEEKDAY
- YEAR

MINUTE

Syntax

RTC_MIN

void setMIN(int value);

Description

RTC_MIN returns the minute of the real time clock.
Range 0 to 59.

setMIN(x) will change the minute value of the real time clock.

Example

```
printf("%d:%02d:%02d",RTC_HOUR,RTC_MIN,RTC_SEC);
```

The output would look like:

```
13:15:30
```

See also

- DAY
- HOUR
- MINUTE
- MONTH
- SECOND
- TIMER
- WEEKDAY
- YEAR

MONTH

Syntax

RTC_MON
void setMON(x);

Description

Function setting or returning the month.
Range 1 to 12.

Example

```
printf("This is: ");
switch (RTC_DOW) {
 case 0: printf("Sunday "); break;
 case 1: printf("Monday "); break;
 case 2: printf("Tuesday "); break;
 case 3: printf("Wednesday "); break;
 case 4: printf("Thursday "); break;
 case 5: printf("Friday "); break;
 case 6: printf("Saturday "); break;
}
printf(" %d/%d/%d \n",RTC_MONTH,RTC_DAY,RTC_YEAR);
```

The output would look like:

```
This is Friday 4/14/2006
```

See also

- DAY
- HOUR
- MINUTE
- MONTH
- SECOND
- TIMER
- WEEKDAY
- YEAR

SECOND

Syntax

RTC_SEC

void setSEC(int value);

Description

RTC_SEC returns the second value of the real time clock.
Range 0 to 59.

setSEC(x) will set the SECOND value of the real time clock

Example

```
printf("%d:%02d:%02d",RTC_HOUR,RTC_MIN,RTC_SEC);
```

The output would look like:

```
13:15:30
```

See also

- DAY
- HOUR
- MINUTE
- MONTH
- TIMER
- WEEKDAY
- YEAR

TIMER

Syntax

TIMER

Description

TIMER is a free running timer that increments every microsecond. It is readable using this keyword. You can also write to the TIMER register.

Operations that require more precise timing should use the dedicated hardware routines, as interupts that are occuring for other time functions and serial input may make times using TIMER look longer than actual.

Example

```
int save_time;  // must be signed to handle roll-over

save_time = TIMER;
while ((TIMER - save_time) < 15) ;     // wait 15 uSec
```

```
TIMER = 0;   // reset the free running timer
```

See also

- DAY
- HOUR
- MINUTE
- MONTH
- SECOND
- TIMER
- WEEKDAY
- YEAR

WAIT

Syntax

WAIT(*milliseconds*) ;

Description

Delay program execution a number of milliseconds.
1000 milliseconds is one second

Example

```
case 42:   // check MicroMega FPU status -- clock pin 15 and data on 14, no CS used
 printf("FPU SPI test\n");

 SPImode = 0; // FPU uses MSB first -- positive clock
 for (i=0; i<10; i++) shortMessage[i]= 0xFF;
 shortMessage [i] = 0;
 SPIOUT (-1,14,15,11,shortMessage);  // reset FPU
 WAIT (10);                                  // allow recovery time - 10 milliseconds
```

See also

- **TIMER**
- **SLEEP**

WEEKDAY

Syntax

RTC_DOW

Description

This hardware register maintains the day of the week. 0 corresponding to Sunday through 6 corresponding to Saturday

This value will be set when the setYEAR routine is called, it is counted from a known date.

Example

```
printf("This is: ");
switch (RTC_DOW) {
 case 0: printf("Sunday "); break;
 case 1: printf("Monday "); break;
 case 2: printf("Tuesday "); break;
 case 3: printf("Wednesday "); break;
 case 4: printf("Thursday "); break;
 case 5: printf("Friday "); break;
 case 6: printf("Saturday "); break;
}
```

The output would look like:

```
This is Friday
```

See also

- DAY
- HOUR
- MINUTE
- MONTH
- SECOND
- TIMER
- WEEKDAY
- YEAR

YEAR

Syntax

RTC_YEAR

void setYEAR (x};

Description

Function setting or returning the year.

When setting the date, the year should be set last, as it will calculate the day of the week and day of the year.

Example

```
printf("This is: ");
switch (RTC_DOW) {
 case 0: printf("Sunday "); break;
 case 1: printf("Monday "); break;
 case 2: printf("Tuesday "); break;
 case 3: printf("Wednesday "); break;
 case 4: printf("Thursday "); break;
 case 5: printf("Friday "); break;
 case 6: printf("Saturday "); break;
}
printf(" %d/%d/%d \n",RTC_MONTH,RTC_DAY,RTC_YEAR);
```

The output would look like:

```
This is Friday 4/14/2006
```

See also

- DAY
- HOUR
- MINUTE
- MONTH
- SECOND
- TIMER
- WEEKDAY
- YEAR

Alphabetical Keyword List

AD

Syntax

extern int AD(int channel);

Description

ARMmite, ARMexpress LITE and ARMweb

AD will return 0..65472 that corresponds to the voltage on the pin corresponding to *expression* . The value returned will have the top 10 bits being significant. (bits 5..0 will be 0). 0 would be read for 0V and 65472 for 3.3V.

An analog conversion on pin *expression* is performed. This process takes less than 6 usec.

Dual Use AD pins

ARMmite version

On reset or power up the AD pins are configured as AD inputs. To change those to digital IOs, the user must individually specify a switch to digital using DIGITAL(IOpin) which is a #define for IOpins 16-23.

ARMexpress LITE version

The ARMexpress LITE supports up to 6 channels of AD converters.

On the ARMexpress LITE and ARMweb these pins are configured as digital IOs at reset, but must be switched to AD using configAD(channel) which is a #define for channels 0-7.

AD(0)	IO(7)
AD(1)	IO(10)
AD(2)	IO(8)
AD(3)	not available
AD(4)	not available
AD(5)	IO(9)
AD(6)	IO(11)
AD(7)	IO(12)

Example

```
temperature = AD(0) >> 6;          // read a raw temperature voltage
```

See also

- TXD
- RXD

configAD

Syntax

#define configAD(int channel);

Description

Dual Use AD pins

ARMexpress LITE version

The ARMexpress LITE supports up to 6 channels of AD converters.

On the ARMexpress LITE and ARMweb these pins are configured as digital IOs at reset, but must be switched to AD using configAD(channel) which is a #define for channels 0-7.

AD(0)	IO(7)
AD(1)	IO(10)
AD(2)	IO(8)
AD(3)	not available
AD(4)	not available
AD(5)	IO(9)
AD(6)	IO(11)
AD(7)	IO(12)

Example

```
configAD(1);          // switch IO(10) to AD(1)
```

See also

- **TXD**
- **RXD**

COUNT

Syntax

int COUNT(int pin, int duration);

Description

Count the number of pulses low-high-low or high-low-high on *pin* over a duration of *milliseconds*, returning the value to *variable*.

Example

```
printf("\n count pulses on pin 0 for 5 seconds\n");

printf("%d pulses happened\n",COUNT(0, 5000));
```

See also

- RCTIME

DAY

Syntax

RTC_DAY
void setDAY (x);

Description

Function setting or returning the day of the month.
Range 1 to 28, 29, 30, or 31
(depending on the month and whether it is a leap year).

Example

```
printf("This is: ");
switch (RTC_DOW) {
 case 0: printf("Sunday "); break;
 case 1: printf("Monday "); break;
 case 2: printf("Tuesday "); break;
 case 3: printf("Wednesday "); break;
 case 4: printf("Thursday "); break;
 case 5: printf("Friday "); break;
 case 6: printf("Saturday "); break;
}
printf(" %d/%d/%d \n",RTC_MONTH,RTC_DAY,RTC_YEAR);
```

The output would look like:

```
This is Friday 4/14/2006
```

See also

- DAY
- HOUR
- MINUTE
- MONTH
- SECOND
- TIMER
- WEEKDAY
- YEAR

DIGITAL

Syntax

#define DIGITAL(int IOpin)

Description

Dual Use AD pins

ARMmite version

On reset or power up the AD pins are configured as AD inputs. To change those to digital IOs, the user must individually specify a switch to digital using DIGITAL(IOpin) which is a #define for IOpins 16-23.

Example

```
DIGITAL(4);          // switch AD(4) to a digital pin accessed at IO(20)
```

See also

- TXD
- RXD

DIR

Syntax

int DIR (int *expression*) ; // defined for port 0

int DIRx(int *pin*); // for pin 0-31 on port 0, pin 32-63 on port 1, pin 64-95 on port 2 ...

Description

DIR (expression) can be used to read the direction of the up to 32 configurable Port 0 pins. If DIR (expression) is 1 then the corresponding pin is an output. If the value is 0 then that pin is an input.

With the SuperPRO and PROplus, NXP has added more configurable port pins. We are adopting their convention for control of those pins. FIO0DIR, FIO1DIR, FIO2DIR, and FIO4DIR control the input/output status. Writing 1 to a bit will make it an output. More details in the **NXP user manuals.**

Example

```
// Set pin 4 as an input
INPUT(4);

// Check the direction of pin 4
dir4 = DIR(4);
```

See also

- **DIR**
- **HIGH**
- **IN**
- **INPUT**
- **LOW**
- **OUTPUT**

HIGH

Syntax

void HIGH (int *pin); // defined for port 0*

void HIGHx(int pin); // for pin 0-31 on port 0, pin 32-63 on port 1, pin 64-95 on port 2 ...

Description

HIGH will set the Port 0 pin corresponding to *expression* to a positive value (3.3V).

HIGH does not change the direction of the port 0 pin, ie. it must be set to an OUTPUT before HIGH has any affect.

With the SuperPRO and PROplus, NXP has added more configurable port pins. We are adopting their convention for control of those pins. FIO0SET, FIO1SET, FIO2SET, and FIO4SETwill set an output high when written with a 1. More details in the **NXP user manuals.**

Example

```
//  set pins 0 to 7 to output and low or to 0 V
for (i=0;i<8; i++) {
  OUTPUT( i );
  LOW( i );
}

for (i=0;i<8; i++) {
  WAIT(1000);
  OUTPUT( i );
  HIGH ( i );          // set each pin HIGH one after the other every second
}
```

See also

- **DIR**
- **HIGH**
- **IN**
- **INPUT**
- **LOW**
- **OUTPUT**

HOUR

Syntax

RTC_HOUR

void setHOUR(int value);

Description

RTC_HOUR returns the hour of the real time clock.
Range 0 to 23.

setHOUR(x) will change the gour of the real time clock.

Example

```
printf("%d:%02d:%02d",RTC_HOUR,RTC_MIN,RTC_SEC);
```

The output would look like:

```
13:15:30
```

See also

- DAY
- HOUR
- MINUTE
- MONTH
- SECOND
- TIMER
- WEEKDAY
- YEAR

HWPWM

Syntax

void HWPWM (int *cycletime*,int *itemCnt*, int **hightimeList*);

Description

ARMmite and Wireless ARMmite version

The ARMmite supports up to 8 channels of hardware driven PWM. The IO direction of the pin will be set to output. Once programmed these will continue to generate the specified PWM until re-programmed or reset.

Cycletime is in microseconds, is the time for a single PWM cycle. *Hightimes* are also in microseconds and represent the amount of time during the cycle that the corresponding outputs are high. It is assumed, but not enforced that cycletimes for all channels will be the same.

channel1	IO(0)
channel2	IO(1)
channel3	IO(2)
channel4	IO(3)
channel5	IO(4)
channel6	IO(9)
channel7	IO(10)
channel8	IO(11)

ARMmite PRO version

The ARMmite PRO also supports up to 8 channels of hardware driven PWM. The IO direction of the pin will be set to output. Once programmed these will continue to generate the specified PWM until re-programmed or reset.

Cycletime is in microseconds, is the time for a single PWM cycle. *Hightimes* are also in microseconds and represent the amount of time during the cycle that the corresponding outputs are high. It is assumed, but not enforced that cycletimes for all channels will be the same.

channel1	IO(0)
channel2	IO(1)
channel3	IO(8)
channel4	IO(5)
channel5	IO(14)
channel6	IO(10)
channel7	IO(11)
channel8	IO(3)

ARMexpress LITE version

The ARMexpress LITE supports up to 6 channels of hardware driven PWM. The IO direction of the pin will be set to output. Once programmed these will continue to generate the specified PWM until re-programmed or reset. 2 of the channels are not available on the pins.

Cycletime is in microseconds, is the time for a single PWM cycle. *Hightimes* are also in microseconds and represent the amount of time during the cycle that the corresponding outputs are high. It is assumed, but not enforced that cycletimes for all channels will be the same.

channel1	IO(5)
channel2	IO(6)
channel3	IO(3)
channel4	not available
channel5	IO(14)
channel6	not available
channel7	IO(13)
channel8	IO(15)

SuperPRO version

The PROplus and SuperPRO support up to 6 channels of hardware driven PWM. The IO direction of the pin will be set to output. Once programmed these will continue to generate the specified PWM until re-programmed or reset.

Cycletime is in microseconds, is the time for a single PWM cycle. *Hightimes* are also in microseconds and represent the amount of time during the cycle that the corresponding outputs are high. It is assumed, but not enforced that cycletimes for all channels will be the same.

channel1	P2.0
channel2	P2.1
channel3	P2.2
channel4	P2.3
channel5	P2.4
channel6	P2.5

The LPC17xx series processors also have an additional 6 channels designed to drive motors. See details in the Motor PWM Control chapter of the NXP LPC17xx User Manual. Also these pins can be re-assigned as selected by the PINSEL registers.

Example

```
times[0] = -1
times[1] = 750
times[2] = 100
HWPWM (1000, 3, times)      'generate 1KHz 75% and 10% signals on pins 1,2

times[0] = 2000
times[1] = 1000
times[2] = 500
HWPWM (4000, 3, times)      'gen 250 Hz 50%, 25% and 12.5% signals on pins 0,1,2
```

See also

- **FREQOUT**
- **PWM**

I2CIN

Syntax

void I2CIN (int *sda_pin*, int *scl_pin*, int *slaveADDR*, int opt1, int opt2, int opt3, int opt4, int opt5, int *cnt* , char **inList*);

Description

I2CIN will read a series of bytes from an I2C slave device. *sda_pin* is any expression defining the SDA pin to use. *scl_pin* will be designated the SCL pin. *slaveADDR* will select a device on the I2C bus.

Up to 5 optional byte values may be writen out prior to reading the *inList* . If the *optX* value is -1 then it will not be sent out. Most i2c devices have a sub-address field that is written immediately before reading. This field would be held in opt1. For devices that do not have a sub-address, opt1 can be set to -1. In this case, no I2C write is performed before the I2CIN. For some slow i2c devices, they can not respond to a write of sub-address, immediately followed by a read of the device. For these slow devices (often implemented with a slower micro-processor), it is necessary to do seperate I2COUT and I2CIN with a delay in between.

After any write of the *optX* ,a series of *cnt* bytes will be read from the slave to fill the character array *inList* .

I2C is a byte oriented bus, so each transaction will either send a byte value (0 to 255) or receive a byte for each element of the *inList* .

Data is shifted in at 350 Kbits/sec.

Example

```
case 41:   // test EEPROM 24LC02 on pins 0 == SDA and 1 == SCL
 shortMessage[0] = 0;  // address into EEPROM
 for (i=1; i<8; i++) shortMessage[i] = 0x30+i; // set shortMessage to "1234567"
 present = I2COUT (0, 1, 0xA0,8, shortMessage);

 if (present == 0) printf("NO i2c device ***\n"); else printf("i2c device found\n");
 I2CIN(0, 1, 0xA0, 0, -1, -1, -1, -1, 7, shortResponse);
```

See also

- I2COUT

I2COUT

Syntax

int I2COUT (int *sda_pin*, int *scl_pin*, int *slaveADDR* , int *cnt*, char **outList*);

// I2COUT returns TRUE if the device was present and responding

Description

I2COUT will send a series of bytes from an I2C slave device. *sda_pin* is any expression defining the SDA pin to use. *scl_pin* will be designated the SCL pin. *slaveADDR* will select a device on the I2C bus.

After that *cnt* bytes will be written to the slave from the character array *outList*.

I2C is a byte oriented bus, so each transaction will send a byte values (0 to 255) to an I2C slave. If the value from an *expr* ession in the *OutputList* is larger than 8 bits, the MSBs will be truncated.

If a I2C device responds the function returns 1, else 0.

Data is shifted out at 350 Kbits/sec.

Example

```
case 41:   // test EEPROM 24LC02 on pins 0 == SDA and 1 == SCL
 shortMessage[0] = 0;
 for (i=1; i<8; i++) shortMessage[i] = 0x30+i; // set shortMessage to "1234567"
 present = I2COUT (0, 1, 0xA0, 8, shortMessage);
```

See also

- **I2CIN**

I2CSPEED

Syntax

void I2CSPEED (int *Kbits);*

Kbits= 400 | 100 | 50

Description

The default speed for I2C operations is approx 350 Kbits, which is acceptable to most modern I2C chips.

In order to support older chips or longer cable runs, the I2C operations can be slowed down to 100 Kbits.

A very slow 50 Kb rate is also supported.

Example

```
I2CSPEED (100);                 // slow down the operation
```

See also

- **I2CIN**
- **I2COUT**

IN

Syntax

int IN (int *pin*); // defined for port 0

int INx(int *pin*); // for pin 0-31 on port 0, pin 32-63 on port 1, pin 64-95 on port 2 ...

Description

IN returns the value on the Port 0 pin corresponding to expression.

0 is returned for 0V and -1 for 2.5V or above. Why -1 and 0? NOT(~) 0 is equal to -1.

With the SuperPRO and PROplus, NXP has added more configurable port pins. We are adopting their convention for control of those pins. FIO0DIR, FIO1DIR, FIO2DIR, and FIO4DIR control the input/output status. Writing 1 to a bit will make it an output. More details in the**>NXP user manuals.**

Example

```
printf("Pin 15 is ");
INPUT(15);                    // make pin 15 an INPUT

printf("%d\n",IN(15));        // now read the value on that pin
```

See also

- **DIR**
- **HIGH**
- **IN**
- **INPUT**
- **LOW**
- **OUTPUT**

INPUT

Syntax

void INPUT (int *pin*); // defined for port 0

void INPUTx(int *pin*); // for pin 0-31 on port 0, pin 32-63 on port 1, pin 64-95 on port 2 ...

Description

INPUT will set the Port 0 pin corresponding to *expression* to an input.

With the SuperPRO and PROplus, NXP has added more configurable port pins. We are adopting their convention for control of those pins. FIO0DIR, FIO1DIR, FIO2DIR, and FIO4DIR control the input/output status. Writing 1 to a bit will make it an output. More details in the **NXP user manuals.**

Example

```
printf("Pin 15 is ");
 INPUT(15);                    // make pin 15 an INPUT

 printf("%d\n",IN(15));              // now read the value on that pin
```

See also

- **DIR**
- **HIGH**
- **IN**
- **INPUT**
- **LOW**
- **OUTPUT**

LOW

Syntax

void LOW (int *pin*); // defined for port 0

void LOWx(int *pin*); // for pin 0-31 on port 0, pin 32-63 on port 1, pin 64-95 on port 2 ...

Description

LOW will set the Port 0 pin corresponding to *expression* to a negative value (0V) and then set it to an output.

With the SuperPRO and PROplus, NXP has added more configurable port pins. We are adopting their convention for control of those pins. FIO0CLR, FIO1CLR, FIO2CLR, and FIO4CLR will set an output low when written with a 1. More details in the **NXP user manuals.**

Example

```
//  set pins 0 to 7 to output and low or to 0 V
for (i=0;i<8; i++) {
  OUTPUT( i );
  LOW( i );
}

for (i=0;i<8; i++) {
  WAIT(1000);
  OUTPUT( i );
  HIGH ( i );          // set each pin HIGH one after the other every second
}
```

See also

- **DIR**
- **HIGH**
- **IN**
- **INPUT**
- **LOW**
- **OUTPUT**

MINUTE

Syntax

RTC_MIN

void setMIN(int value);

Description

RTC_MIN returns the minute of the real time clock.
Range 0 to 59.

setMIN(x) will change the minute value of the real time clock.

Example

```
printf("%d:%02d:%02d",RTC_HOUR,RTC_MIN,RTC_SEC);
```

The output would look like:

```
13:15:30
```

See also

- DAY
- HOUR
- MINUTE
- MONTH
- SECOND
- TIMER
- WEEKDAY
- YEAR

MONTH

Syntax

RTC_MON
void setMON(x);

Description

Function setting or returning the month.
Range 1 to 12.

Example

```
printf("This is: ");
switch (RTC_DOW) {
 case 0: printf("Sunday "); break;
 case 1: printf("Monday "); break;
 case 2: printf("Tuesday "); break;
 case 3: printf("Wednesday "); break;
 case 4: printf("Thursday "); break;
 case 5: printf("Friday "); break;
 case 6: printf("Saturday "); break;
}
printf(" %d/%d/%d \n",RTC_MONTH,RTC_DAY,RTC_YEAR);
```

The output would look like:

```
This is Friday 4/14/2006
```

See also

- DAY
- HOUR
- MINUTE
- MONTH
- SECOND
- TIMER
- WEEKDAY
- YEAR

OUTPUT

Syntax

void OUTPUT (int *pin*); // defined for port 0

void OUTPUTx(int *pin*); // for pin 0-31 on port 0, pin 32-63 on port 1, pin 64-95 on port 2 ...

Description

OUTPUT will set the Port 0 pin corresponding to *expression* to an output.

With the SuperPRO and PROplus, NXP has added more configurable port pins. We are adopting their convention for control of those pins. FIO0DIR, FIO1DIR, FIO2DIR, and FIO4DIR control the input/output status. Writing 1 to a bit will make it an output. More details in the **NXP user manuals.**

Example

```
printf("LED on\n");
OUTPUT(15);
LOW(15);
```

See also

- **DIR**
- **HIGH**
- **IN**
- **INPUT**
- **LOW**
- **OUTPUT**

OWIN

Syntax

void OWIN (int *pin*, int *out1*,int *out2*,int *out3*,int *out4*,int *out5*, int *out6*, int *cnt*, char **inList*) ;

Description

OWIN begins with a RESET/Presence sequence on the designated *Pin*.

Then upto 6 *Output* bytes will be transfered to the device to select the command. Byte values are sent out, if the value is -1, then that byte is not sent.

Following that cnt bytes will be read into the character array *inList* .

The bit order for the 1-Wire device is assumed to be LSB (bit 0) first.

Example

```
present = OWOUT(7, 7, shortMessage);
if (present) printf("one wire device found\n");
else {printf("*** NO one wire device ***\n"); break;}

OWIN(7, 0xCC, 0xAA, 0, -1, -1, -1, 4, shortResponse);

for (i=0; i<4; i++) if (shortMessage[i+3] != shortResponse[i]) break;
```

See also

- OWOUT

OWOUT

Syntax

int OWOUT (int *pin*, int *cnt*, char **outList*) ;

// OWOUT returns TRUE if the device was present and responding

Description

OWOUT begins with a RESET/Presence sequence on the designated *Pin*.

Following that *cnt*bytes of the character array *outList*will be sent to the device.

The bit order for the 1-Wire device is assumed to be LSB (bit 0) first.

If a one-wire device responds the function returns 1, else 0.

Example

```
case 40:   // test the EEPROM of a DS2430 on pin 7
 shortMessage[0] = 0xCC;
 shortMessage[1] = 0x0F;
 shortMessage[2] = 0x00;
 shortMessage[3] = 0x44;
 shortMessage[4] = 0x11;
 shortMessage[5] = 0x22;
 shortMessage[6] = 0xBE;

 present = OWOUT(7, 7, shortMessage);
 OWIN(7, 0xCC, 0xAA, 0, -1, -1, -1, 4, shortResponse);
```

See also

- **OWIN**

PULSIN

Syntax

int PULSIN (int pin, int state);

Description

Measure an input pulse on *pin* at *level*, returning the value.

The IO direction of *pin* will be set to input.

If *pin* is already at *level* when the function is called it will wait to a transition to the opposite *level*.

The function will wait 1 second for *pin* to go to *level*. The length of time is measured in microseconds(us). The minimum pulse that can be measured is 1 microseconds. If *pin* does not go to level or remains at *level* longer than 1 second 0 is returned.

Example

```
case 32:
 prinf("\n Measure PULSIN on pin 0, four times as low pulse then high pulse\n");
 for (i=0; i<4; i++) {
  printf ("%d\n", PULSIN (0, 0) );
 }
 for (i=0; i<4; i++) {
  printf ("%d\n", PULSIN (0, 1) );
 }
```

See also

- **RCTIME**
- **COUNT**

PULSOUT

Syntax

void PULSOUT (int pin, int duration);

Description

Generate an output pulse on *pin* for *microseconds*.

The IO direction of *pin* will be set to output. The level of the output will be switched, driven for *microseconds*, then switched back to its initial level. The minimum pulse period is 1 microseconds.

Example

```
printf("Flash LED for 250ms off for 500ms\n");
OUTPUT(15);
HIGH(15);
for (i=0; i<20; i++) { PULSOUT (15,250000); WAIT (500);}
```

See also

- **PULSIN**

PWM

Syntax

void PWM(int pin, int duty, int duration);

Description

Generate an analog signal on *pin* for *milliseconds* with a *duty* cycle of 0 to 255.
A *duty* cycle of 255 corresponds to an output value of 100%.

The IO direction of the pin will be set to output, the PWM pulse train is output, and then the pin is set to tristate (input). If the pin is connected to an RC filter, then the voltage will stay on the capacitor for a period of time determined by the load.

Example

```
printf("Ramp thru PWM on LED\n");
 for (i=0; i<256; i++) PWM (15,255-i,20);
```

See also

- **FREQOUT**
- **PULSOUT**

RCTIME

Syntax

int RCTIME(int *pin*, int *state*) ;

Description

Measure the time which *pin* remains at *state*, returning the value to *variable*.
The length of time is measured in microseconds(us). The minimum time measured is 1 microseconds.
If *pin* is not at *level* when the function is called *variable* is set to 1.
If *pin* remains at *level* longer than 1 second *variable* is set to 0.

Example

```
    printf("\n check RCTIME into a 700ohm 0.1uF drive on 14, sense on 13\n");
    OUTPUT (14);
//  while(1) {HIGH(14);WAIT(1);LOW(14);WAIT(1);} // scope loop

    HIGH (14);
    printf("high to low ");
    WAIT(10);  // make sure line is high
    LOW(14);
    printf("%d microseconds, high to low ",RCTIME (13, 1));
    WAIT(10);  // make sure line is low
    HIGH (14);
    printf("%d microseconds, low to high\n", (RCTIME (13, 0));
```

```
 check RCTIME into a 700ohm 0.1uF drive on 14, sense on 13
high to low 89 microseconds, high to low 56 microseconds, low to high
```

See also

- **PULSIN**

RXD

Syntax

int RXD (int *pin*);

int RXD0(); // for UART0 access

int RXD1(); // for UART1 access

Description

RXD (*pin*) will receive a single byte of data that is shifted as an asynchronous serial stream. This function is similar to SERIN, but is a more efficient implementation. The baudrate for the pin should be set before using RXD, that is done by setting the SERbauds[] array.

RXD will timeout after 0.5 seconds and return -1. These routines are "bit-banged" by the processor, so the processor is consumed during these operations. Interupts are also disabled during each byte for these operations.

The 0.5 second timeout can be changed by **SERINtimeout**.

Baudrate can be upto 115.2Kb.

UART0 UART1 support-

For RXD0 data is received on the SIN pin. SIN and SOUT are always negative true. UART0 of the LPC21xx, RXD1 for UART1.

For ARMexpress modules baudrates for RXD0 can be upto 19.2 Kbaud as it is limited by the level translators. No limit for ARMmite/ARMweb Use setbaud() routine to set this baud.

Example

```
while (1) {
   ch = RXD(4);
   if (ch != -1) printf("%c", ch );
   if (ch == 'Q') break;
} // receive and echo characters from testjig
```

See also

- **setbaud**
- **TXD**
- **SERIN**

SECOND

Syntax

RTC_SEC

void setSEC(int value);

Description

RTC_SEC returns the second value of the real time clock.
Range 0 to 59.

setSEC(x) will set the SECOND value of the real time clock

Example

```
printf("%d:%02d:%02d",RTC_HOUR,RTC_MIN,RTC_SEC);
```

The output would look like:

```
13:15:30
```

See also

- DAY
- HOUR
- MINUTE
- MONTH
- TIMER
- WEEKDAY
- YEAR

SERbauds, setbaud

Syntax

extern int SERbauds[];

void setbaud (int channel, int baudDiv); // for UART0 and UART1

Description

This array contains the baud rates for up to 32 IOs. It is used when RXD or TXD routines are called.

Baudrates can be upto 115.2 Kbaud.

The UART0, UART1 hardware channel baudrate can be set by calling-

void setbaud(0, baudDiv); // baudDiv = 15000000 / baudrate / 16 -- for example 19.2Kb is 49 for ARMmite, ARMexpress, and ARMweb

void setbaud(1, baudDiv); // baudDiv = 25000000 / baudrate / 16 -- for example 19.2Kb is 81 forSuperPRO and PROplus

For the ARMexpress SIN and SOUT are limited by the level translators to 19.2Kbaud. This last limit does not apply to the SuperPRO, PROplus, ARMmite or ARMweb.

Example

```
SERbauds[4] = 19200;
 SERbauds[3] = SERbauds[4];

 setbaud (1, 17);  // set UART1 baudrate to 56Kb
```

See also

- TXD
- RXD

SERIN

Syntax

void SERIN (int *pin,* int *baudrate,* int *posTrue,* char *cnt,* char **inList*);

Description

SERIN receives *cnt* bytes as asynchronous serial data on *pin* at a *baudrate* and saves the data into *inList* .

PosTrue if set to 0 then the data is inverted.

inList is a pointer to a character array.

SERIN will timeout after 0.5 seconds and return -1 and 255 in the next item in the *inList* before the timeout. These routines are "bit-banged" by the processor, so the processor is consumed during these operations. Interupts are also disabled during each byte for these operations.

The 0.5 second timeout can be changed by **SERINtimeout**.

Baudrates can be upto 115.2 Kbaud for all pins.

For UART0 support use-

int getline(char *line, int max_len) ;

For ARMexpress modules baudrates for RXD0 can be upto 19.2 Kbaud as it is limited by the level translators. No limit for ARMmite/ARMweb Use setUART0baud() routine to set this baud.

Example

```
while (1) {
   SERIN (4,19200,1, 1, shortMessage);
   if (shortMessage[0] != 255) printCh( shortMessage[0] );
   if (shortMessage[0] == 'Q') break;
} // receive and echo characters from testjig
```

See also

- **SEROUT**
- **SERINtimeout**

SERINtimeout

Syntax

extern int SERINtimeout;

Description

The default time that SERIN waits for input on pins 0-31 is 0.5 seconds.

This may be changed using this statement. The timeout will be set in micro-seconds.

Example

```
// set the timeout shorter on for polling an LCD/keypad combination to 100 ms
SERINtimeout = 100000;
   ...

proc GetKey() {
   LCDcmd[0] = 24;
   LCDlen = 0;
   GenCRC();
   SEROUT (1,LCDbaud,1,LCDlen+3,LCDcmd);
   SERIN  (1,LCDbaud,1,3,LCDcmd);
}
```

See also

- SERIN
- SEROUT

SEROUT

Syntax

void SEROUT (int *pin,*int *baudrate,* int *posTrue,*int *cnt,* char **outList*);

Description

SEROUT sends *cnt* bytes pointed to by *outList* out as asynchronous serial data on *pin* at a *baudrate*. *PosTrue* if set to 0 then the data is inverted.

*outList*is a character array.

These routines are "bit-banged" by the processor, so the processor is consumed during these operations. Interupts are also disabled during each byte for these operations.

Baudrates can be upto 115.2 Kbaud for all pins

UART0 support-

Use printf or TXD0. The hardware serial port routines are used, so the CPU is not tied up. So when a byte is sent it is placed into the UART FIFO, but if the 16 byte FIFO is full then the CPU will wait until space is available.

For ARMexpress modules baudrates for RXD0 can be upto 19.2 Kbaud as it is limited by the level translators. No limit for ARMmite/ARMweb Use setUART0baud() routine to set this baud.

Example

```
shortMessage[0] = 0x33;
 shortMessage[1] = 0x34;
 shortMessage[2] = 0x35;

 SEROUT(3,19200, 1, 3, shortMessage);    // send 123 to the testjig serial connection
```

See also

- SERIN

SERbauds, setbaud

Syntax

extern int SERbauds[];

void setbaud (int channel, int baudDiv); // for UART0 and UART1

Description

This array contains the baud rates for up to 32 IOs. It is used when RXD or TXD routines are called.

Baudrates can be upto 115.2 Kbaud.

The UART0, UART1 hardware channel baudrate can be set by calling-

void setbaud(0, baudDiv); // baudDiv = 15000000 / baudrate / 16 -- for example 19.2Kb is 49 for ARMmite, ARMexpress, and ARMweb

void setbaud(1, baudDiv); // baudDiv = 25000000 / baudrate / 16 -- for example 19.2Kb is 81 forSuperPRO and PROplus

For the ARMexpress SIN and SOUT are limited by the level translators to 19.2Kbaud. This last limit does not apply to the SuperPRO, PROplus, ARMmite or ARMweb.

Example

```
SERbauds[4] = 19200;
 SERbauds[3] = SERbauds[4];

 setbaud (1, 17);  // set UART1 baudrate to 56Kb
```

See also

- TXD
- RXD

SHIFTIN

Syntax

void SHIFTIN (int *in_pin*, int *clk_pin*, int *mode*, int *cnt*, int **wordList* , int *bitLengths*) ;

Description

SHIFTIN has been kept as a compatable function with PBASIC. It can be used for devices that are not covered by SPI, I2C or 1-Wire. Data is shifted in on *in_pin*, and a positive clock is sent on *clk_pin* for each bit.

While most other hardware functions use bytes, SHIFTIN is oriented for bit control. The length of each variable defines the number of bits that will be shifted out (2 - 32). For each word *bitLengths* bits will be shifted in.

- Mode = 0 data is shifted in MSB first, and sampling starts before the first clock pulse
- Mode = 1 data is shifted in LSB first, and sampling starts before the first clock pulse
- Mode = 2 data is shifted in MSB first, and sampling starts before the second clock pulse
- Mode = 3 data is shifted in LSB first, and sampling starts before the second clock pulse

SHIFTIN fills the integer array *wordList* . *cnt* integers will be shifted in.

Data is shifted in at 600 Kbits/sec.

Example

```
printf("FPU SHIFT test\n");     for (i= 0; i<10; i++) wordMessage[i]= 0xFF;
 wordMessage [i] = 0;
 SHIFTOUT (14,15,MSBFIRST,11,wordMessage, 8); // reset FPU
 WAIT (10);    wordMessage [0] = 0xF0;       // sync character
 SHIFTOUT (14,15,MSBFIRST,1,wordMessage, 8);  // sync FPU
 save_time = TIMER;
 while ((TIMER - save_time) < 15) ;    // wait 15 uSec

 SHIFTIN (14,15, MSBPRE,  1,wordMessage, 8);  // get 1 byte status back
 if (wordMessage[0] != 0x5C) {
   printf(%x "FPU found\nwordMessage[0]);
   break;
 }
```

See also

- **SHIFTOUT**
- **SPIIN**

SHIFTOUT

Syntax

void SHIFTOUT (int *out_pin*, int *clk_pin*, int *mode*, int *cnt* , int **wordList*, int *bitLengths*) ;

Description

SHIFTOUT has been kept as a compatable function with PBASIC. It can be used for devices that are not covered by SPI, I2C or 1-Wire. Data is placed on *out_pin* and the *clk_pin* is pulsed each bit.

While most other hardware functions use bytes, SHIFTOUT is oriented for bit control. The length of each variable defines the number of bits that will be shifted out (2 - 32). For each element *bitLengths* bits are shifted out.

- Mode = 0 data is shifted out LSB first
- Mode = 1 data is shifted out MSB first

SHIFTOUT uses values from the interger array *wordList*. *cnt* integers will be shifted out.

Data is shifted out of the device at 800 Kbits/sec.

Example

```
printf("FPU SHIFT test\n");     for (i= 0; i<10; i++) wordMessage[i]= 0xFF;
 wordMessage [i] = 0;
 SHIFTOUT (14,15,MSBFIRST,11,wordMessage, 8); // reset FPU
 WAIT (10);    wordMessage [0] = 0xF0;       // sync character
 SHIFTOUT (14,15,MSBFIRST,1,wordMessage, 8);  // sync FPU
 save_time = TIMER;
 while ((TIMER - save_time) < 15) ;    // wait 15 uSec

 SHIFTIN (14,15, MSBPRE,  1,wordMessage, 8);  // get 1 byte status back
 if (wordMessage[0] != 0x5C) {
  printf("%x No FPU found\n",shortResponse[0]);      break;
 }
 printf("FPU found\n");
```

See also

- **SHIFTIN**
- **SPIIN**

SPIIN

Syntax

void SPIIN (int *CS_pin*,int *in_pin,*int *clk_pin,*int *out_pin,* int *out1,* int *out2,* int *out3,* int *cnt, char *InputList*);

Description

SPIIN supports the loosely defined serial protocol used by a variety of manufacturers. The desired device is selected by asserting *CS_pin* LOW. If there is no *CS_pin* , the value should be set to -1.

In the simplest case, *in_pin* is used to input data clocked by *clk_pin*, to fill the character array *InputList* with *cnt* bytes.

In bi-directional cases, *out1..out3* byte values will be output on *out_pin* before reading the *InputList* . If not used, those should be set to -1. It is also allowable to have *in_pin* equal to *out_pin* , in which case that pin will be driven for the *out1..out3* and then converted to an input for *in_pin* .

Data is shifted in LSB first and each element of the *InputList* is filled with a byte of data. To use negative edge clocks or MSB first, the default SPImode may be changed.

Data is shifted in at 600 Kbits/sec.

Example

```
case 42:   // check MicroMega FPU status
           // clock on pin 15 and bi-directional data on 14, no CS used
 SPImode = 0; // FPU uses MSB first -- positive clock
 for (i=0; i<10; i++) shortMessage[i]= 0xFF;
 shortMessage [i] = 0;
 SPIOUT (-1,14,15,11,shortMessage);  // reset FPU
 WAIT (10);

 shortMessage [0] = 0xF0;    // sync character
 SPIOUT (-1,14,15,1,shortMessage);  // sync FPU
 save_time = TIMER;
 while ((TIMER - save_time) < 15) ;     // wait 15 uSec
 SPIIN (-1,14,15, -1,-1,-1,-1, 1,shortResponse);  // get 1 byte status back

 if (shortResponse[0] != 0x5C) {
  printf("%x No FPU found\n",shortResponse[0]);
  break;
 }
 printf("FPU found\n");
```

See also

- **SPIOUT**
- **SPImode**

SPImode

Syntax

extern char SPImode;

Description

SPImode will change the bit order and clock sense for all suceeding SPI commands or until SPImode is changed again.

- SPImode= 0 data is shifted in MSB first, and sampling starts before the first rising clock edge (positive true clock)
- SPImode= 1 data is shifted in LSB first, and sampling starts before the first rising clock edge. (positive true clock) **(default value)**
- SPImode= 2 data is shifted in MSB first, and sampling starts before the second clock pulse
- SPImode= 3 data is shifted in LSB first, and sampling starts before the second clock pulse

Example

```
case 42:   // check MicroMega FPU status
           // clock on pin 15 and bi-directional data on 14, no CS used
 SPImode = 0; // FPU uses MSB first -- positive clock
 for (i=0; i<10; i++) shortMessage[i]= 0xFF;
 shortMessage [i] = 0;
 SPIOUT (-1,14,15,11,shortMessage);  // reset FPU
 WAIT (10);

 shortMessage [0] = 0xF0;    // sync character
 SPIOUT (-1,14,15,1,shortMessage);  // sync FPU
 save_time = TIMER;
 while ((TIMER - save_time) < 15) ;     // wait 15 uSec
 SPIIN (-1,14,15, -1,-1,-1,-1, 1,shortResponse);  // get 1 byte status back

 if (shortResponse[0] != 0x5C) {
   printf("%x No FPU found\n",shortResponse[0]);      break;
 }
 printf("FPU found\n");
```

See also

- **SPIOUT**
- **SPImode**

SPIOUT

Syntax

void SPIOUT (int *CSpin*, int *out_pin,* int *clk_pin,* int *cnt,* char **OutputList*);

Description

SPIOUT supports the loosely defined serial protocol used by a variety of manufacturers. The desired device is selected by asserting *CSpin* LOW. If there is no *cspin*, the value should be set to -1.

In the simplest case, *out_pin* is used to output data clocked by *clk_pin*, from the *OutputList*. *CSpin*, *out_pin* and *clk_pin* are left as and outputs.

Data is shifted out LSB first and each element of the *OutputList* is treated as a byte. This order can be changed with SPImode.

Data is shifted out at 800 Kbits/sec

Example

```
shortMessage [0] = 0xF3;    // get version
 SPIOUT (-1,14,15,1,shortMessage);  // sync FPU
 INPUT (14);        // allow FPU to drive this bidirectional line
 while (IN(14));       // wait for FPU to drive that line low
 shortMessage [0] = 0xF2;    // get string back
 SPIOUT (-1,14,15,1,shortMessage);  // sync FPU
 save_time = TIMER;
 while ((TIMER - save_time) < 15) ;     // wait 15 uSec
 while (1) {
  SPIIN (-1,14,15, -1,-1,-1,-1, 1,shortResponse); // get 1 byte at a time back and print it
  if (shortResponse[0] == 0) break;
  printf("%x\n",shortResponse[0]);
 }
```

See also

- **SPIIN**

TIMER

Syntax

TIMER

Description

TIMER is a free running timer that increments every microsecond. It is readable using this keyword. You can also write to the TIMER register.

Operations that require more precise timing should use the dedicated hardware routines, as interupts that are occuring for other time functions and serial input may make times using TIMER look longer than actual.

Example

```
int save_time;  // must be signed to handle roll-over

save_time = TIMER;
while ((TIMER - save_time) < 15) ;     // wait 15 uSec
```

```
TIMER = 0;   // reset the free running timer
```

See also

- DAY
- HOUR
- MINUTE
- MONTH
- SECOND
- TIMER
- WEEKDAY
- YEAR

TXD

Syntax

void TXD (int *pin,* int *byte*); // for bit-banged version on any pin

void TXD0(int byte); // for hardware support

void TXD1(int byte); // for UART1

Description

TXD (*pin, byte*) will send a single byte of data that is shifted out as an asynchronous serial stream on *pin* . This function is similar to SEROUT, but is a more efficient implementation. The baudrate for the pin should be set before using TXD, that is done setting the SERbauds[] array.

These routines are "bit-banged" by the processor, so the processor is consumed during these operations. Interupts are also disabled during each byte for these operations.

UART0 UART1 support-

The hardware serial port routines are used, so the CPU is not tied up. So when a byte is sent it is placed into the UART FIFO, but if the 16 byte FIFO is full then the CPU will wait until space is available.

For ARMexpress modules baudrates for RXD0 can be upto 19.2 Kbaud as it is limited by the level translators. No limit for ARMmite/ARMweb Use setUART0baud() routine to set this baud.

Example

```
TXD(3,'a'); TXD(3,'b'); TXD(3,'c');  // send out   abc
```

See also

- **setbaud**
- **RXD**
- **SEROUT**

WAIT

Syntax

WAIT(*milliseconds*) ;

Description

Delay program execution a number of milliseconds.
1000 milliseconds is one second

Example

```
case 42:   // check MicroMega FPU status -- clock pin 15 and data on 14, no CS used
 printf("FPU SPI test\n");

 SPImode = 0; // FPU uses MSB first -- positive clock
 for (i=0; i<10; i++) shortMessage[i]= 0xFF;
 shortMessage [i] = 0;
 SPIOUT (-1,14,15,11,shortMessage);  // reset FPU
 WAIT (10);                                  // allow recovery time - 10 milliseconds
```

See also

- **TIMER**
- **SLEEP**

WEEKDAY

Syntax

RTC_DOW

Description

This hardware register maintains the day of the week. 0 corresponding to Sunday through 6 corresponding to Saturday

This value will be set when the setYEAR routine is called, it is counted from a known date.

Example

```
printf("This is: ");
switch (RTC_DOW) {
 case 0: printf("Sunday "); break;
 case 1: printf("Monday "); break;
 case 2: printf("Tuesday "); break;
 case 3: printf("Wednesday "); break;
 case 4: printf("Thursday "); break;
 case 5: printf("Friday "); break;
 case 6: printf("Saturday "); break;
}
```

The output would look like:

```
This is Friday
```

See also

- DAY
- HOUR
- MINUTE
- MONTH
- SECOND
- TIMER
- WEEKDAY
- YEAR

YEAR

Syntax

RTC_YEAR

void setYEAR (x};

Description

Function setting or returning the year.

When setting the date, the year should be set last, as it will calculate the day of the week and day of the year.

Example

```
printf("This is: ");
switch (RTC_DOW) {
 case 0: printf("Sunday "); break;
 case 1: printf("Monday "); break;
 case 2: printf("Tuesday "); break;
 case 3: printf("Wednesday "); break;
 case 4: printf("Thursday "); break;
 case 5: printf("Friday "); break;
 case 6: printf("Saturday "); break;
}
printf(" %d/%d/%d \n",RTC_MONTH,RTC_DAY,RTC_YEAR);
```

The output would look like:

```
This is Friday 4/14/2006
```

See also

- DAY
- HOUR
- MINUTE
- MONTH
- SECOND
- TIMER
- WEEKDAY
- YEAR

Hardware Specs

Hardware Specs

- ARMmite Pin Diagram
- ARMmite Schematic
- ARMexpLITE Pin Diagram
- ARMexpLITE Schematic
- ARMexpress Pin Diagram
- ARMexpress Schematic
- EvalPCB Schematic
- Suggested RS232 connection
- TTL and other interfacing
- Serial Configuration
- Power
- Timing
- SPI,Microwire
- Using the I2C Bus
- ARM Peripheral Use

ARMmite Pin Description

24 pins available to the user, 8 of which can be analog inputs

IO0 IO1 IO2 IO3 IO4 IO5 IO6 IO7 IO8 IO9 IO10 IO11 IO14 IO15	P0.9 P0.8 P0.30 P0.21 P0.20 P0.29 P0.4 P0.5 P0.6 P0.7 P0.13 P0.19 P0.16 P0.15	RXD1 TXD1 EINT0 EINT2	PWM1 PWM2 PWM3 PWM4 PWM5 PWM6 PWM7 PWM8	Input/Outputs -- user controlled 0-3.3V level 4mA drive when configured as Outputs 5V tolerant - use limiting resistor when connecting to a 5V supply IO15 connected to LED
IO12 IO13	P0.18 P0.17			Input/Outputs -- user controlled Open drain 4mA pulldown when configured as Outputs 5V tolerant
AD0 AD1 AD2 AD3 AD4 AD5 AD6 AD7	P0.22 P0.23 P0.24 P0.10 P0.11 P0.12 P0.25 P0.26	IO16 IO17 IO18 IO19 IO20 IO21 IO22 IO23		10 bit A/D inputs may also be used as digital Input/Outputs IO(16-23) -- user controlled when used as analog lines, voltage levels should not exceed 3.3V

Dual Use AD pins

On reset or power up the AD pins are configured as AD inputs. To change those to digital IOs, the user must individually specify a control direction using INPUT x, OUTPUT x, DIR(x), or IO(x) commands. After that they will remain digital IOs until the next reset or power up.

PWM pins

All pins can be used for the software PWM function, and 8 pins can be used for the hardware driven HWPWM function.

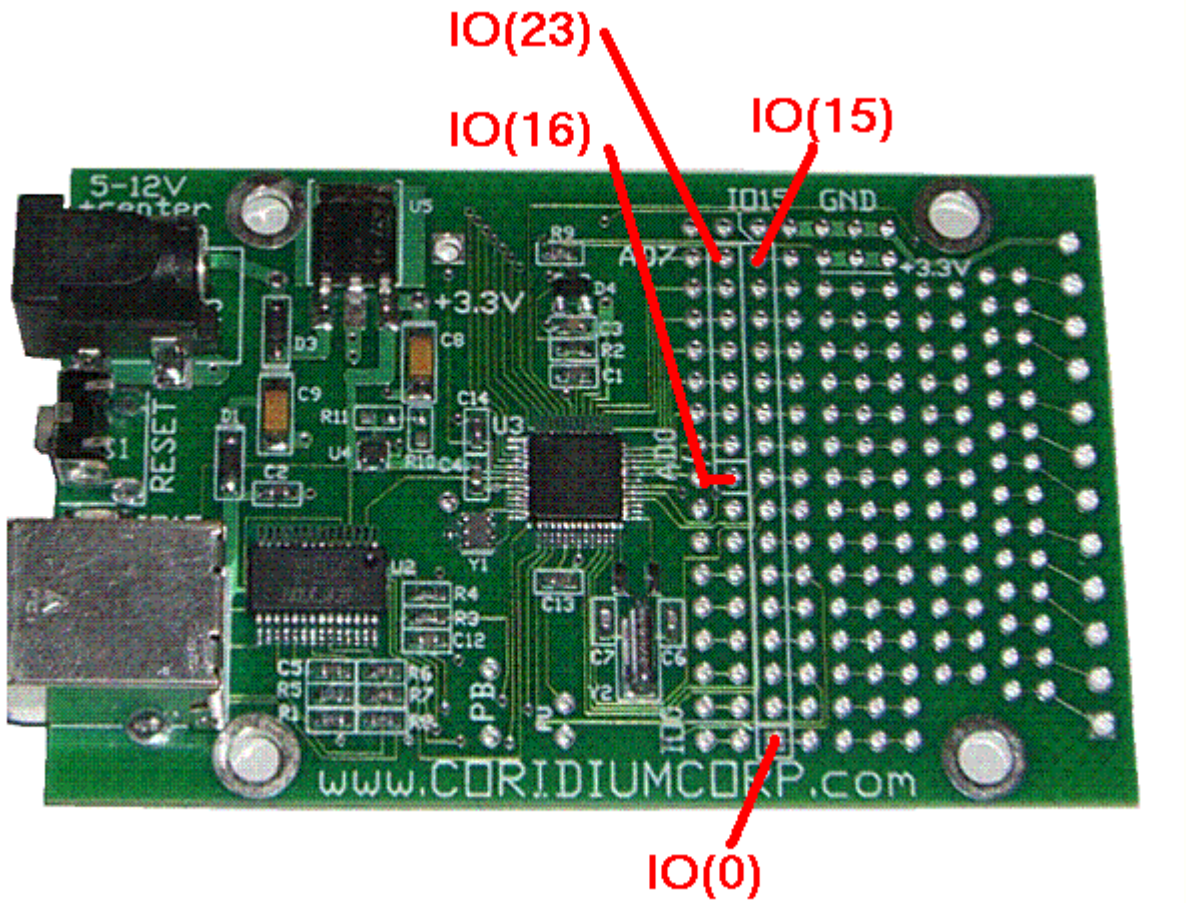
Battery Real Time Clock

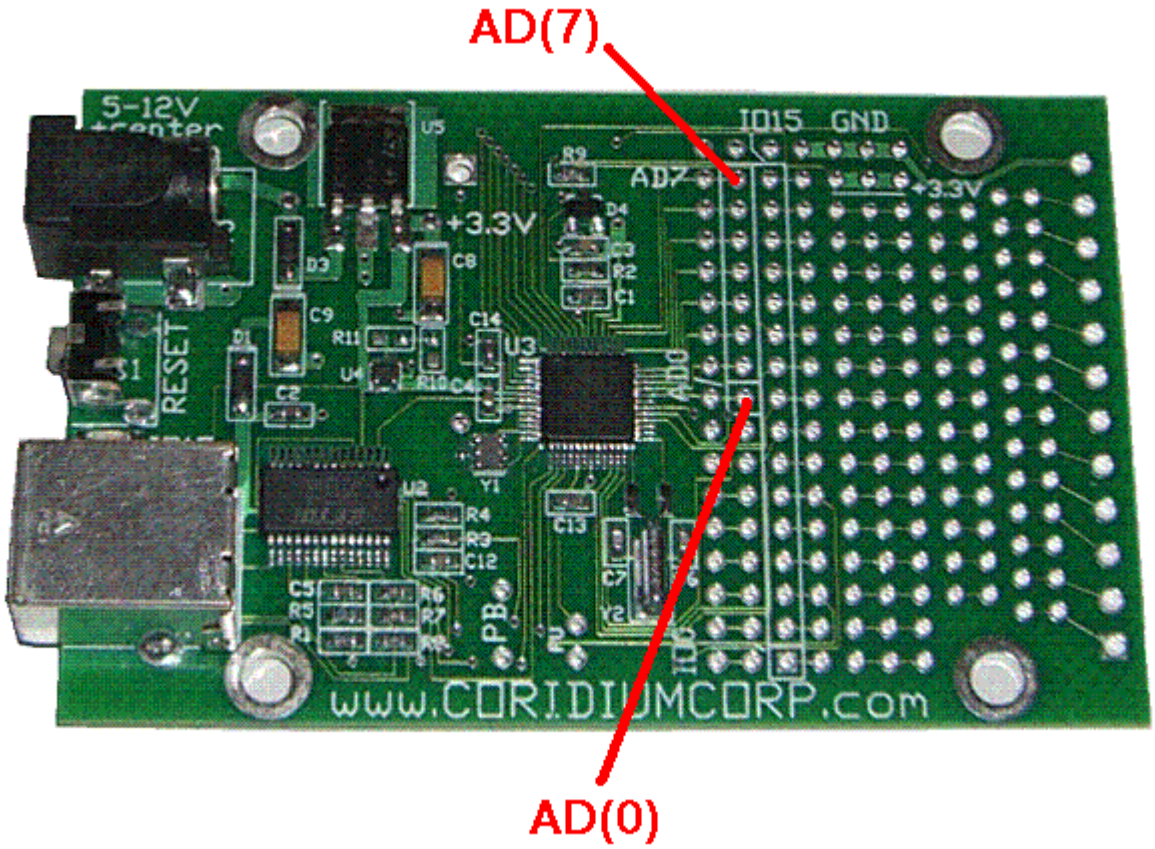
The ARMmite board is designed to accept a Panasonic ML2020/H1C rechargeable Lithium battery at position BT1. This battery powers the real time clock of the LPC2103. The contents of RAM is not kept alive while running on battery, and the CPU restarts the user program in Flash when power is restored. This battery is designed to maintain power for a few days without power, and will recharge fully in about 1 day.

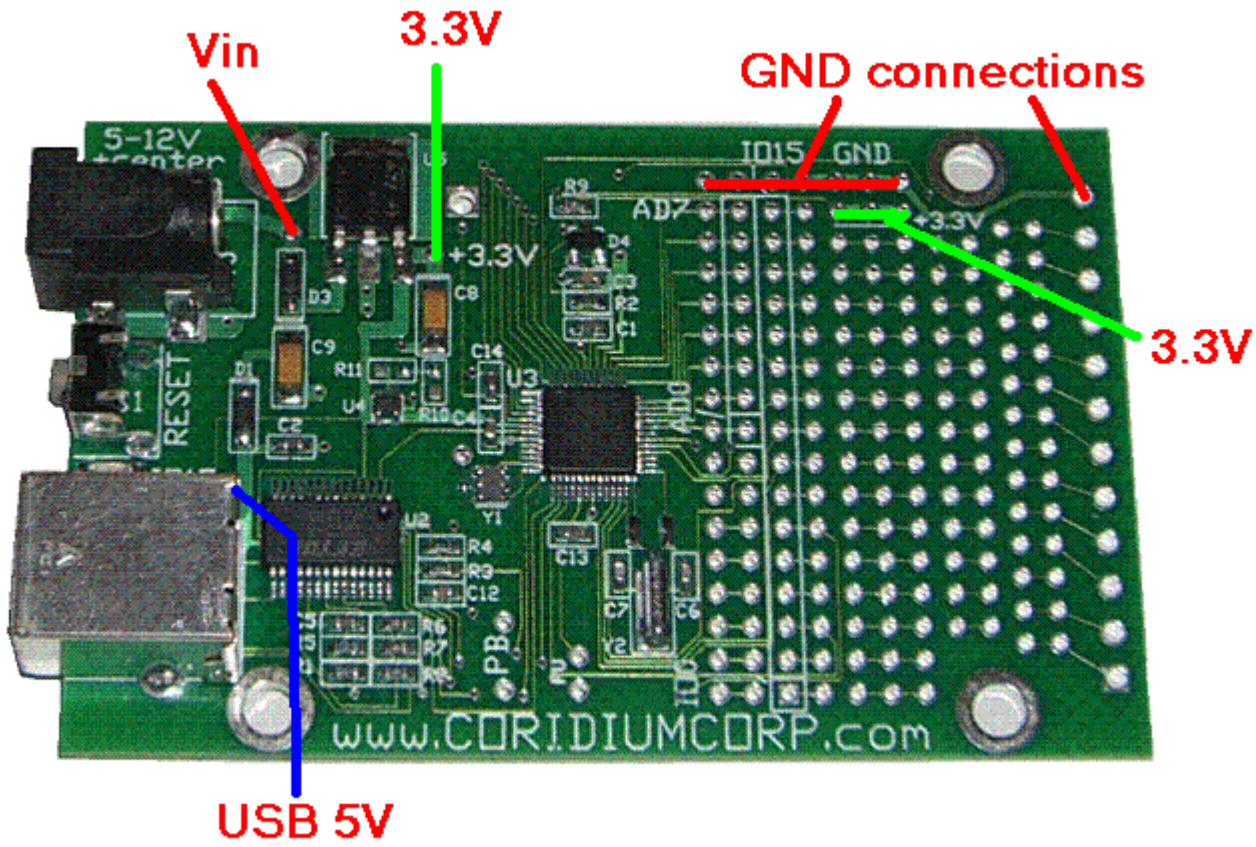
Power connection

Power when not being supplied by a USB connection uses a 2.1mm barrel connector (Cui PJ-002A). Diodes allow both USB and seperate power to be connected simultaneously. If you are using an unregulated wall transformer, you must check the open circuit voltage and it MUST be less than 12V.

REV 3

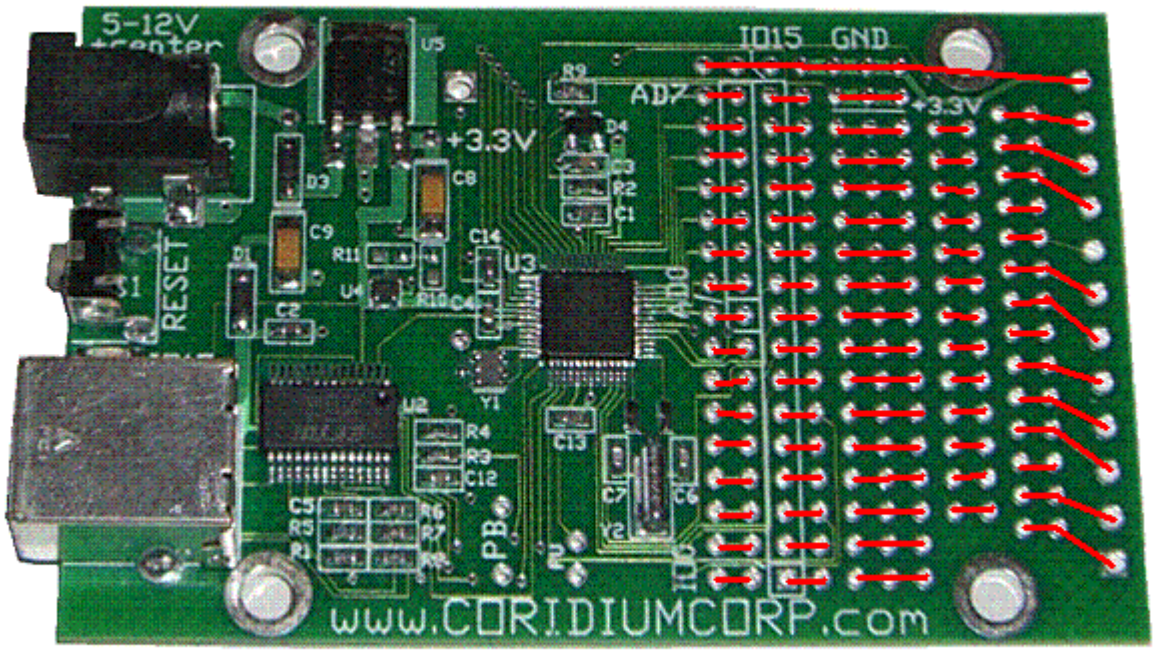


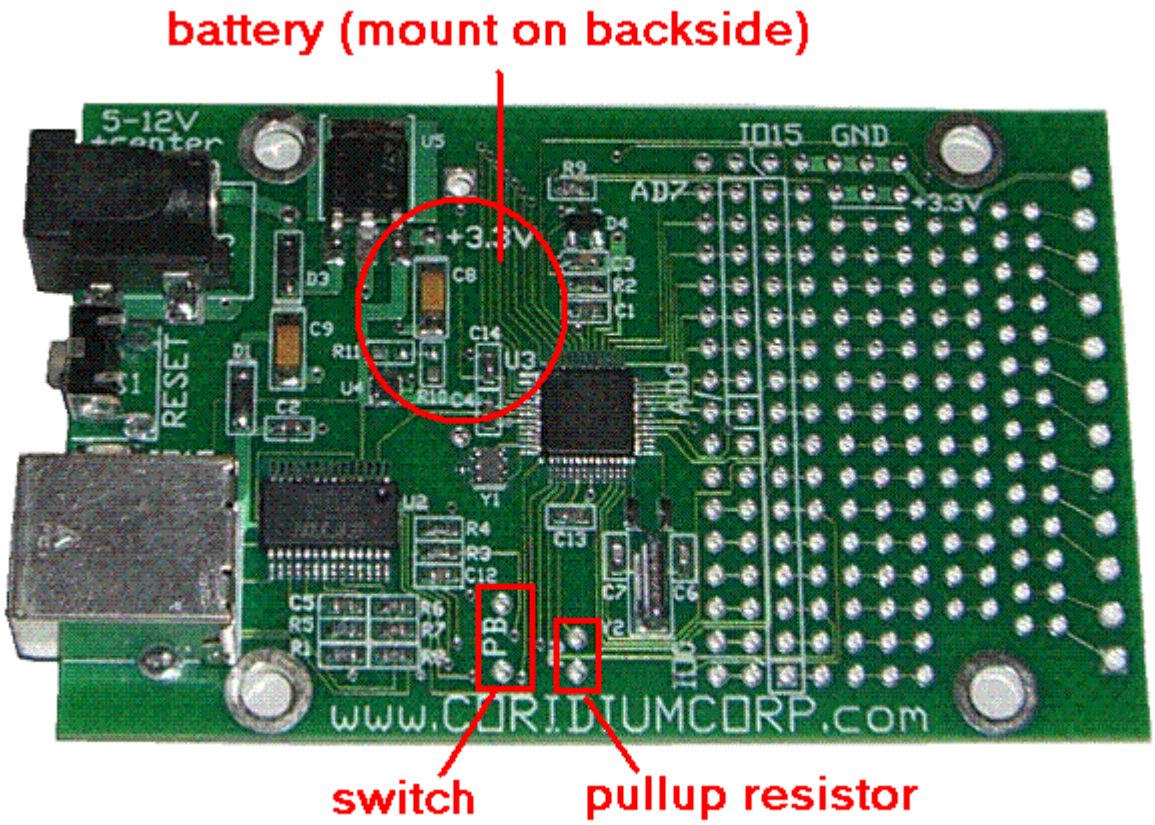

AD(7)
5-12V
IO15 GND
+3.3V
RESET
www.CORIDIUMCORP.com
AD(0)



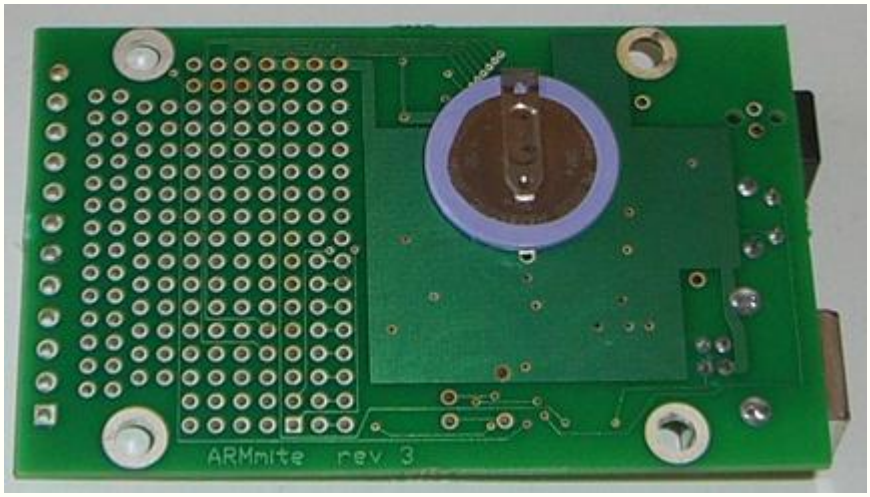
When USB power is not used, a 5-12V supply is required. If 5V is required for some portion of your circuit, it is suggested that a regulated 5V supply be used for input power. These are available from **SparkFun**.

PROTOTYPE Connections

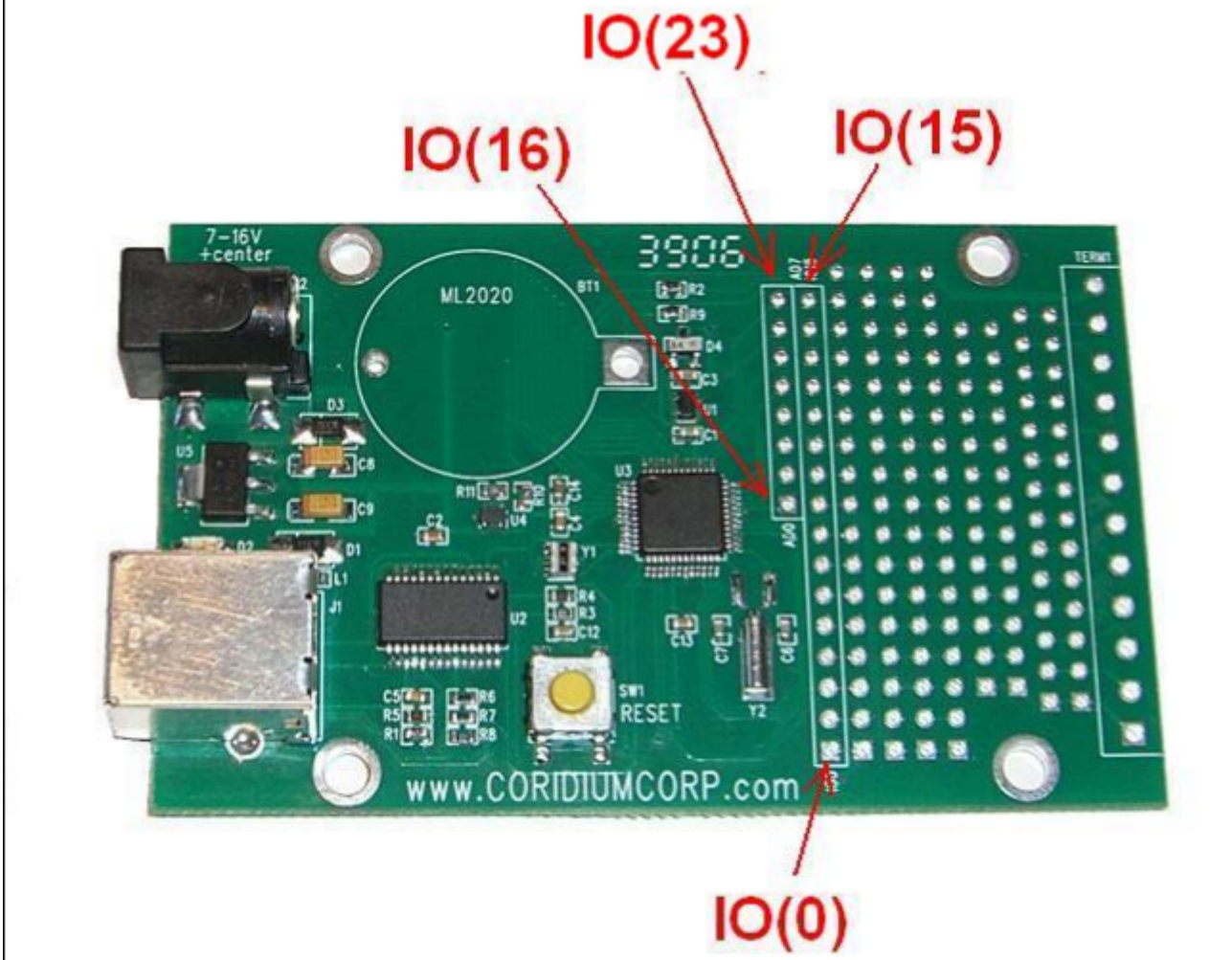


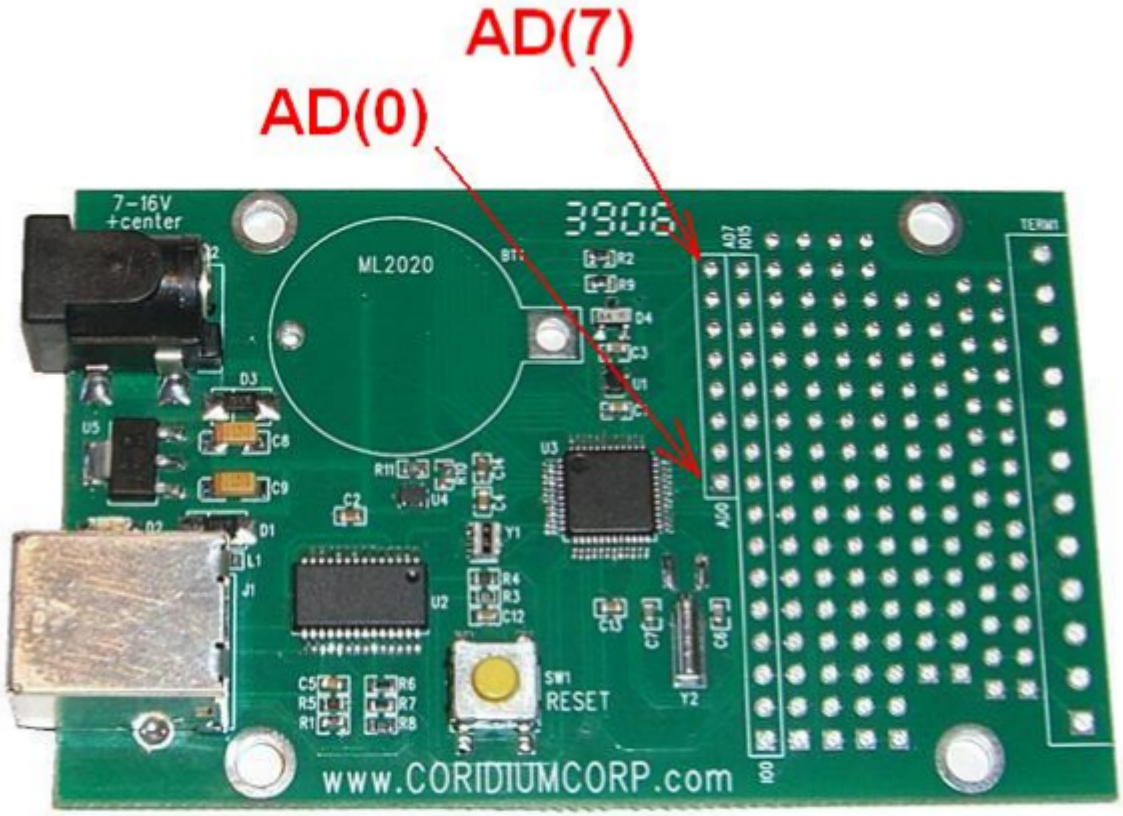


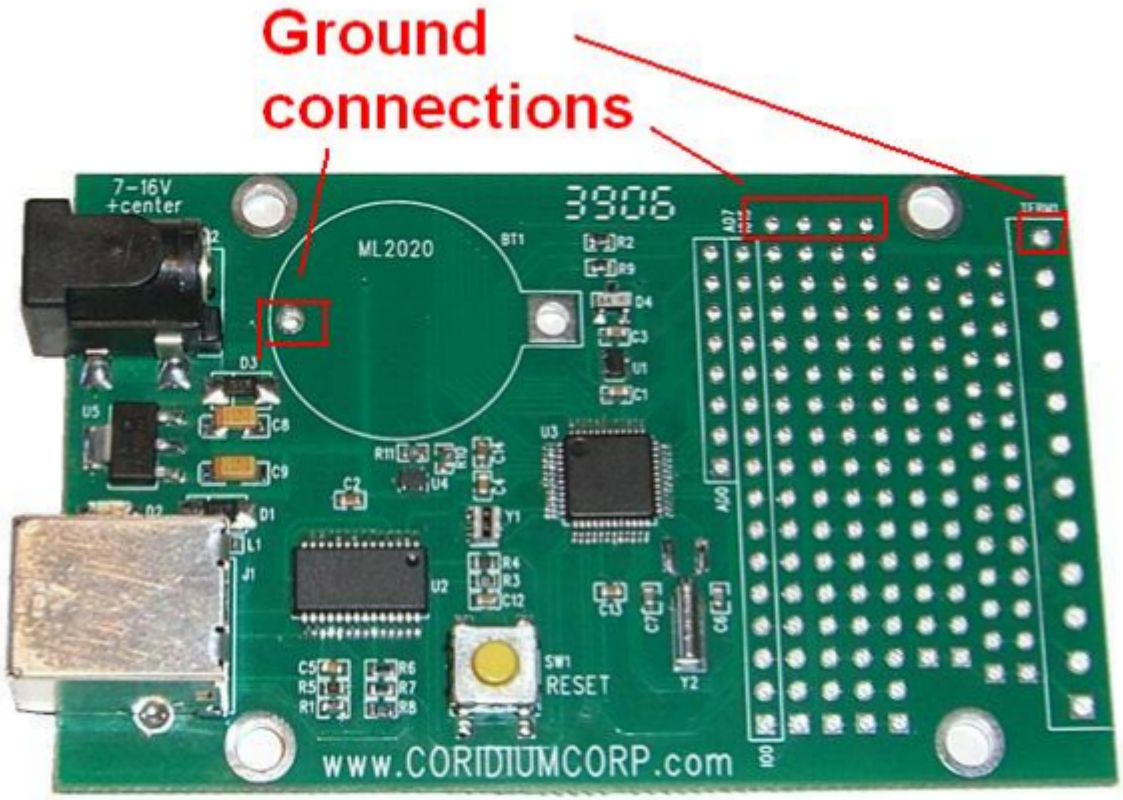
A push button switch and pullup resistor can also be mounted (connected to IO(2)). The optional battery for the real time clock (Panasonic ML2020) can be mounted on the back of the PCB. The VL2020/HFN will also work, though it is more expensive and has less power.

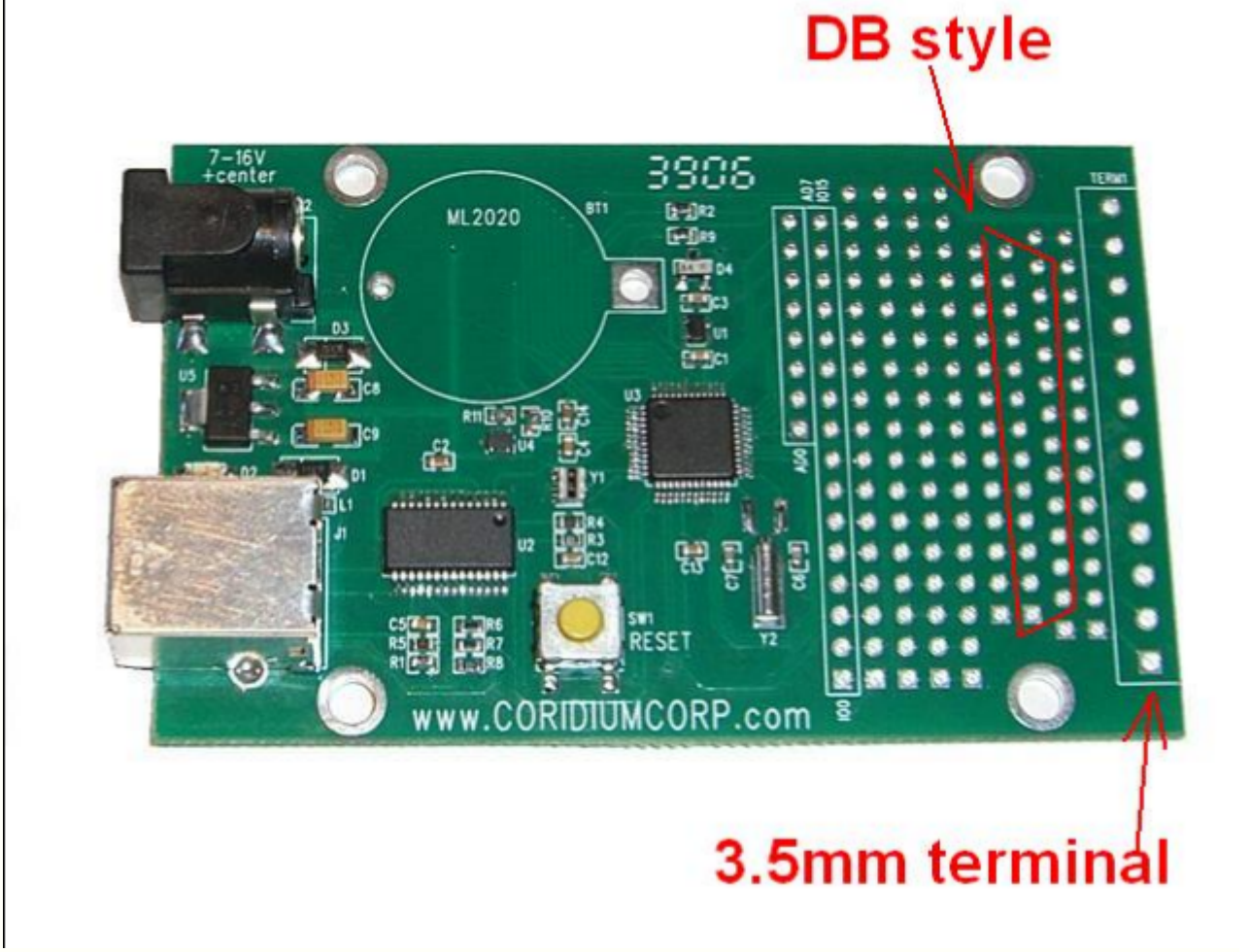


REV 2


IO(23)
IO(16)
IO(15)
7-16V
+center
ML2020
3906
RESET
www.CORIDIUMCORP.com
IO(0)


AD(7)
AD(0)
7-16V
+center
ML2020
3906
RESET
www.CORIDIUMCORP.com


Ground
connections
7-16V
+center
ML2020
3906
RESET
www.CORIDIUMCORP.com



suggested terminal strip On Shore Tech ED550/12DS or equivalent 3.5mm pitch connector (available at Digikey)

Prototype Connections

ARMmite PRO Pin Description

The ARMmite PRO is footprint and pin compatible with the Arduino PRO. In addition it has an onboard 5V regulator so it is compatible with 5V shield boards.

BASIC or C programs can be downloaded using the installed test connector using the USB dongle contained in Coridium's evaluation kit or using the **SparkFun USB Basic Breakout board** or FTDI cable from **MakerShed**. More details on **these connections here**.

Pins available to the user, 7 of which can be analog inputs

IO0 IO1 IO2 IO3 IO4 IO5 IO6 IO7 IO8 IO9 IO10 IO11 IO12 IO13 IO14	P0.9 P0.8 P0.27 P0.19 P0.28 P0.21 P0.5 P0.29 P0.30 P0.16 P0.7 P0.13 P0.4 P0.6 P0.20	RXD1 TXD1 EINT0 	PWM1 PWM2 PWM8 PWM4 PWM3 PWM6 PWM7 PWM5	Input/Outputs -- user controlled 0-3.3V level 4mA drive when configured as Outputs 5V tolerant - use limiting resistor when connecting to a 5V supply
IO15	P0.15	EINT2		IO15 connected to LED -- no other connection
AD0 AD1 AD2 AD3 AD4 AD5 AD6 AD7*	P0.22 P0.23 P0.24 P0.10 P0.11 P0.12 P0.25 P0.26	IO16 IO17 IO18 IO19 IO20 IO21 IO22 IO23		10 bit A/D inputs may also be used as digital Input/Outputs IO(16-23) -- user controlled when used as analog lines, voltage levels should not exceed 3.3V AD6 connected to Arduino AREF pin AD7 connected to a via

Dual Use AD pins

On reset or power up the AD pins are configured as AD inputs. To change those to digital IOs, the user must individually specify a control direction using INPUT x, OUTPUT x, DIR(x), or IO(x) commands. After that they will remain digital IOs until the next reset or power up.

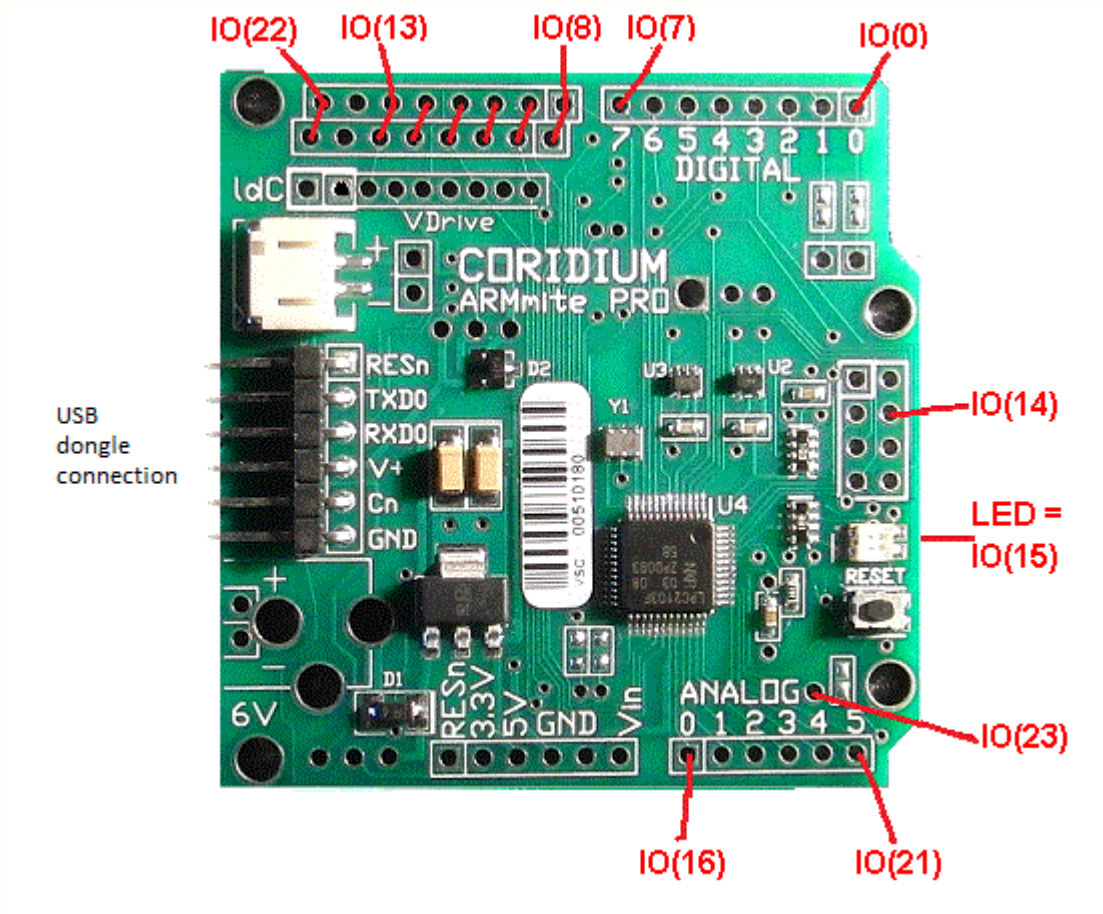
The LPC2103 does not support an external reference for the A/D converters, so the Arduino AREF pin is connected to a seventh converter, AD(6).

PWM pins

All pins can be used for the software PWM function, and 8 pins can be used for the hardware driven HWPWM function.

Digital IO connections

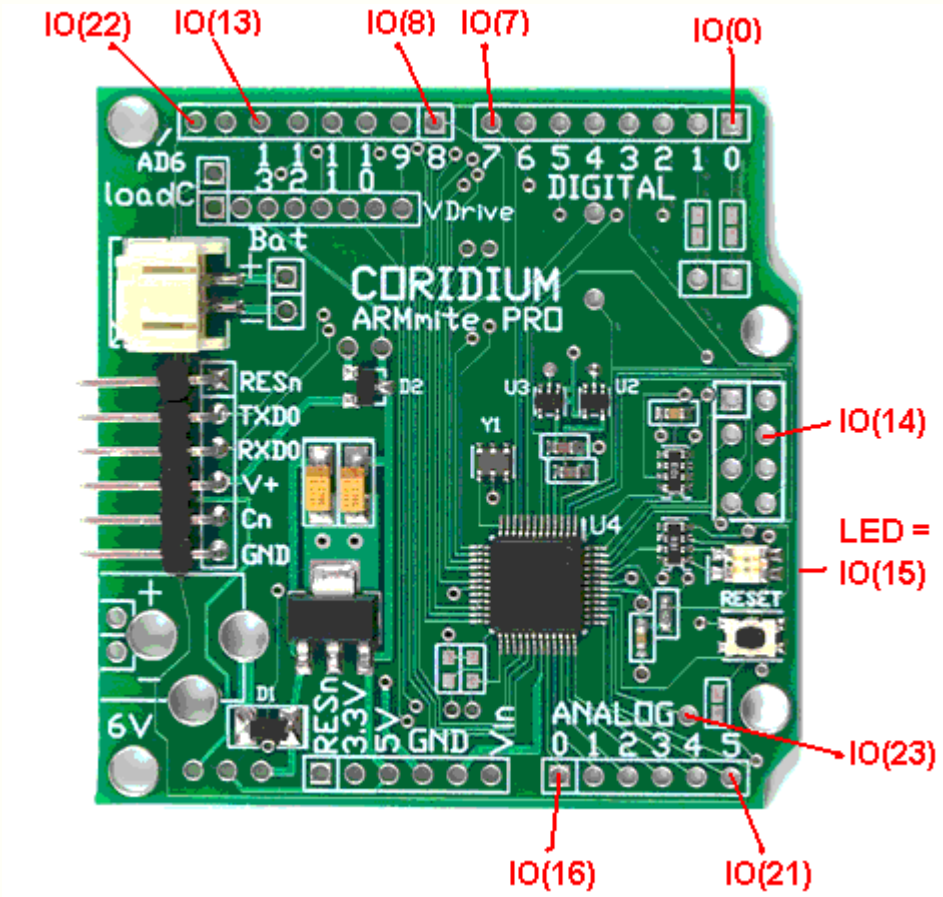
REV4



The major change for rev 4 is to add a parallel connection for the 8 IOs IO(8)-IO(13), GND and IO(22) that is on 0.1" centers in relation to the other connections.

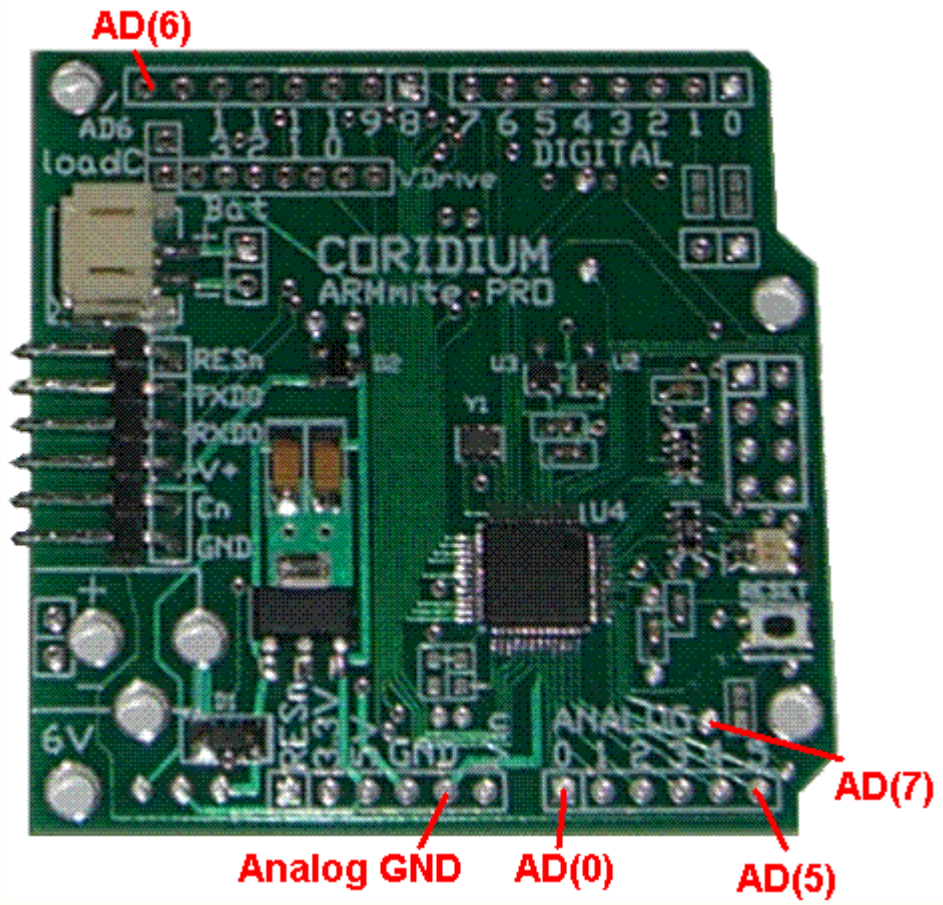
In addition the loadC jumper was rotated 90 degrees to make room for this extra connection. And it is also easier to add a battery to the board, by making 1 cut, and adding a diode, resistor and battery (details below).

REV 3



Picture is for the Rev 3 production board. On the Rev 1, IO(23) is available on the via next to AD(5)/IO(21).

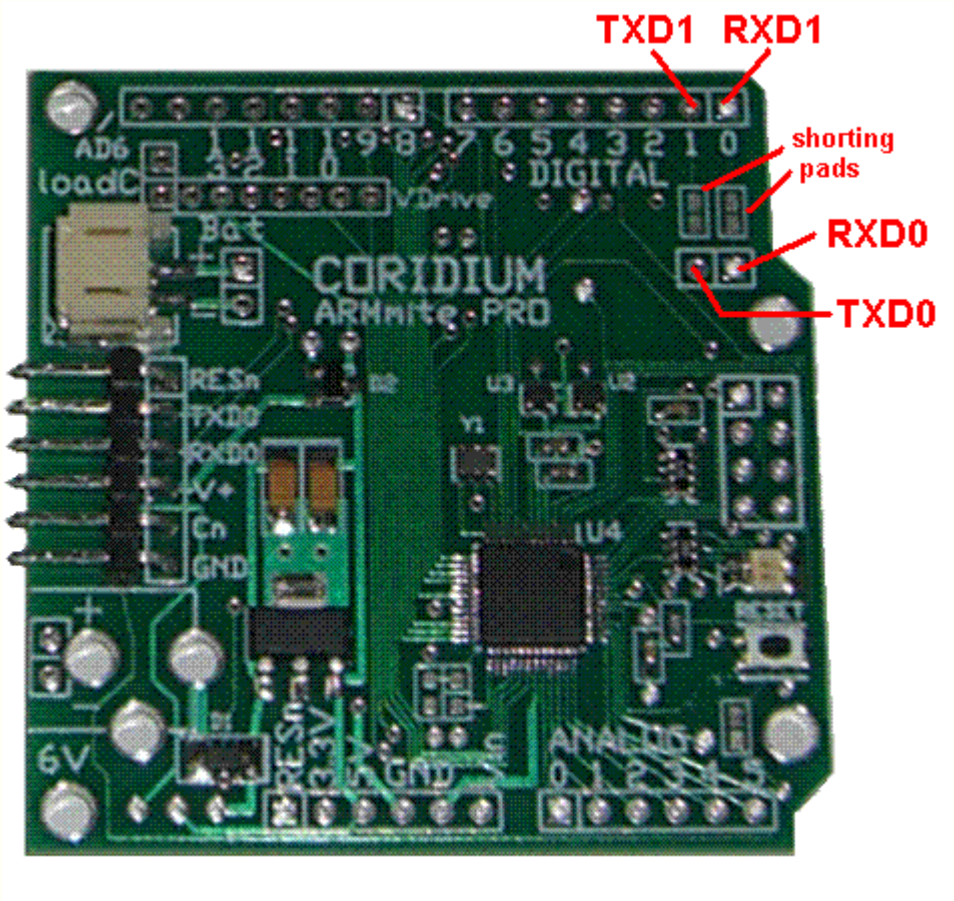
Analog connections



Picture is for the Rev 3 production board. On the Rev 1, AD(7) is available on a via next to AD(5).

Dual Serial Ports

Where the Arduino has only a single serial port, the ARMmite PRO has 2 UARTs. The second UART is connected to IO pins 0 and 1. This allows it to be used simultaneously with the first UART acting as a debug port. In the Arduino, the debug port is connected to these 2 IOs. To allow for this connection as well, the ARMmite PRO has 2 shorting bridges that can be shorted to make this connection.



Power connections

The board is shipped with a **2mm power jack compatible** with a JST PHR/S2B or **SparkFun PRT8671** or various battery packs from SparkFun.

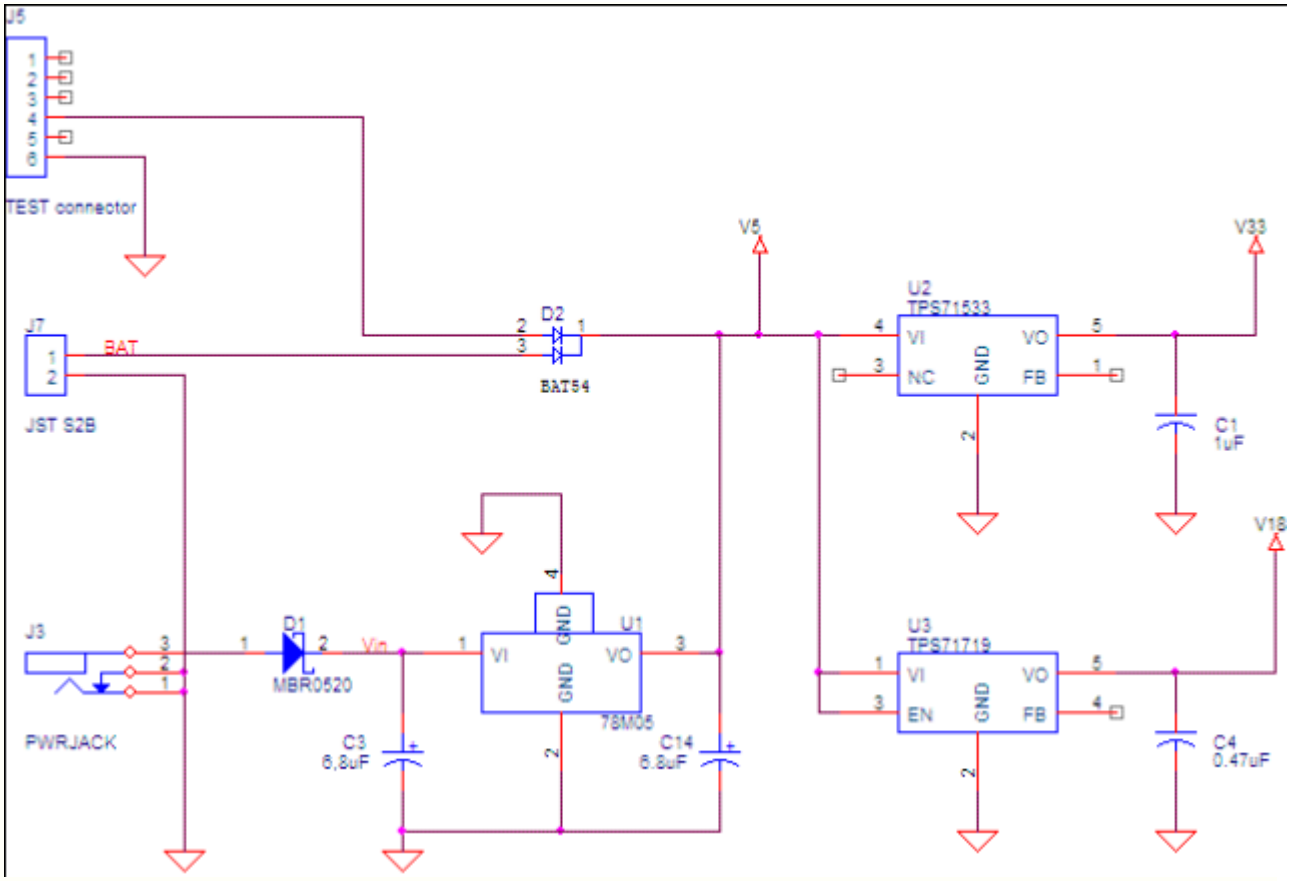
Pads for a Cui PJ-002A or **SparkFun PRT-119** power connector are available in the lower left hand corner.

For both battery and 6V input, 2 pin 0.1" spaced holes are available for wires or headers. When using the battery connector, total current draw for the board must be limited to 200mA. If you want to use more currrent, you should install a jumper around the D2 diode (holes are available above D2).

Diode steering allows power to be supplied from a barrel connector from a 6V unregulated source, 5V USB test connector, or the battery connector. Because of the Schottky diodes, all 3 power sources can be connected simultaneously. If you are using an unregulated wall transformer, you must check the open circuit voltage and it MUST be less than 12V.

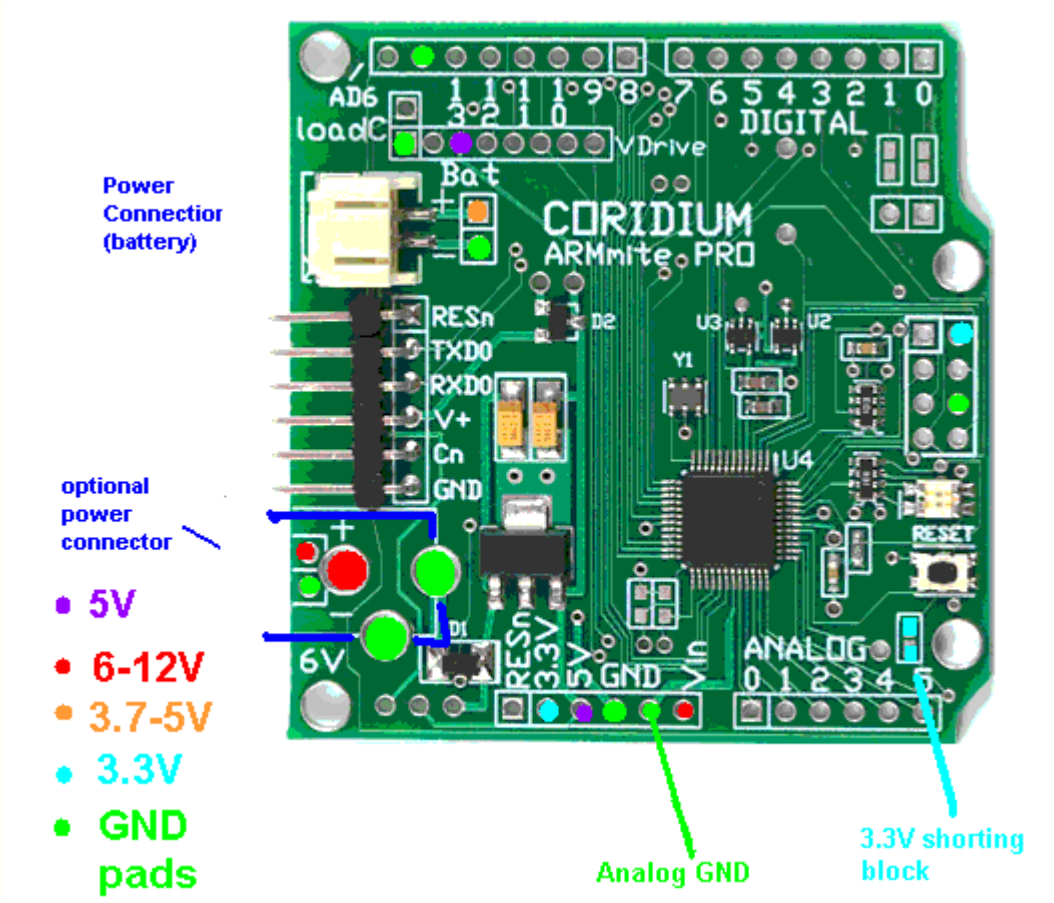
When the 6V source is used, 5V Arduino shields can be powered from the ARMmite PRO.

The schematic describes this circuit



The **full schematic can be seen here**

Power connections details

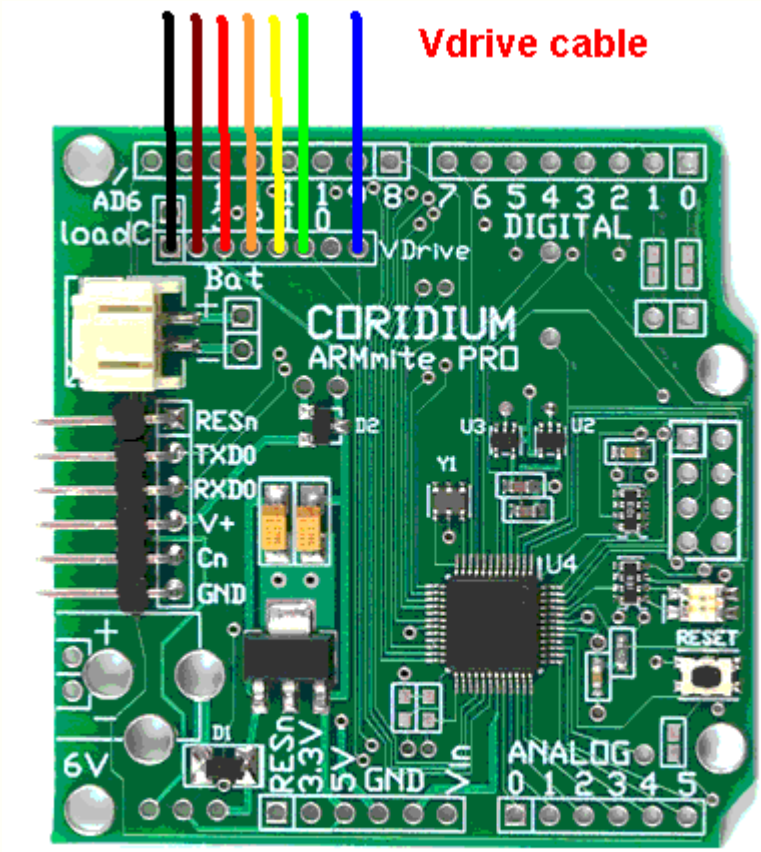


The 3.3V regulator can supply 50 mA, with most being used by the LPC2103. The 3.3V connection next to RESn on the lower power connector is only connected if the shorting pads are shorted (NOT the factory default).

The analog GND should be used to connect to the GND of analog inputs. Digital and Analog GNDs are connected together with a small trace, but to minimize noise you should use the analog GND only for analog signals.

Vdrive connection (added in rev 2)

A connection for the Vdrive has been added so it is easy to use an ARMmite PRO to do data logging to a USB Flash. So all that is required is a **Vdrive** and a **2mm header** .

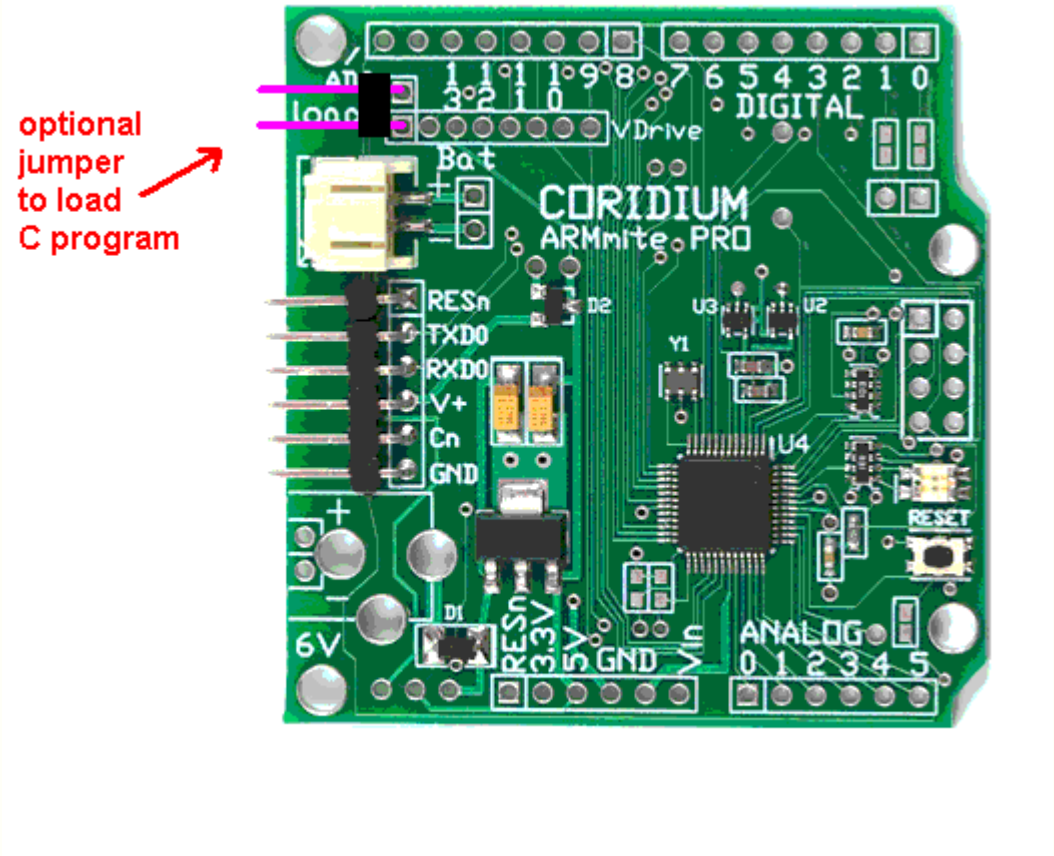


Jumpers and test connector for Program Download

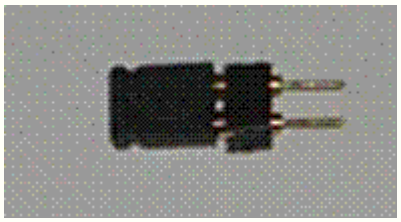
The USB Dongle from Coridium will supply 5V from the USB to power the ARMmite PRO. It also controls the RESET and BOOT signals to automatically load C or BASIC programs using MakeItC or BASICtools. **It is NOT necessary to install the load C jumper when using the USB Dongle from Coridium.**

When using the SparkFun FTDI Basic Breakout Board, a limited amount of power can be supplied from the BBB, but this is limited to 50 mA and after diode drops, its about 2.8V to the LPC2103. In practice this will run, but it is outside the part specifications, so it should be limited in use.

Also with the SparkFun FTDI Basic Breakout Board to load a C program, the LOAD C jumper needs to be installed, then removed to run the program. BASIC programs can be loaded and controlled using the SparkFun board, with no additional steps/jumpers.

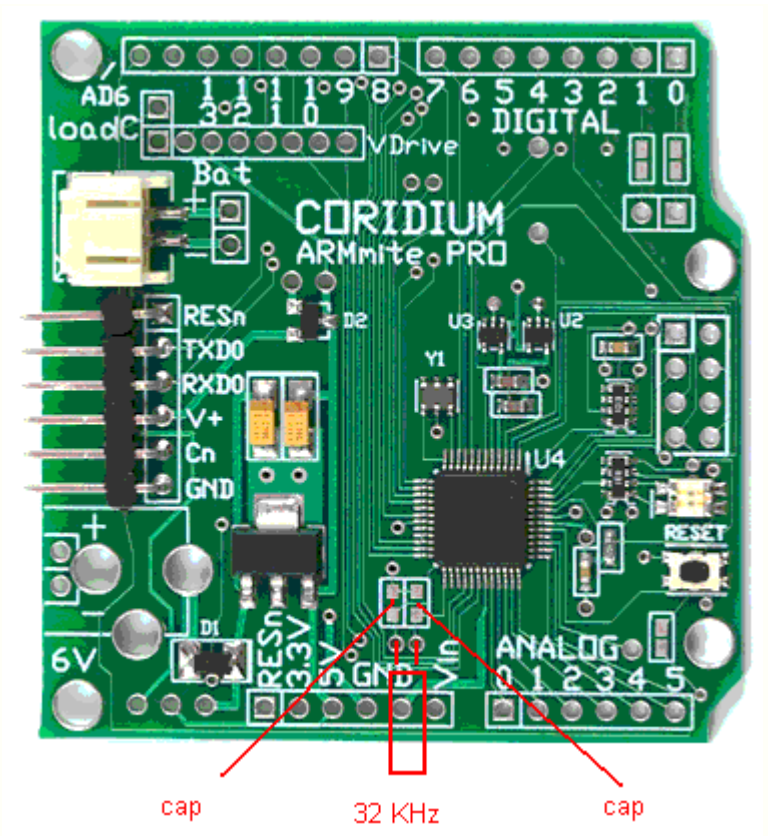


An alternative is to use a 2 pin header with a shorting block (pictured below)



Real Time Clock Oscillator

The ARMmite PRO uses ceramic resonator, which has a 1% accuracy. But there is a provision to load a 32 KHz cyrstal and 2 cap to use that for the Real Time Clock.



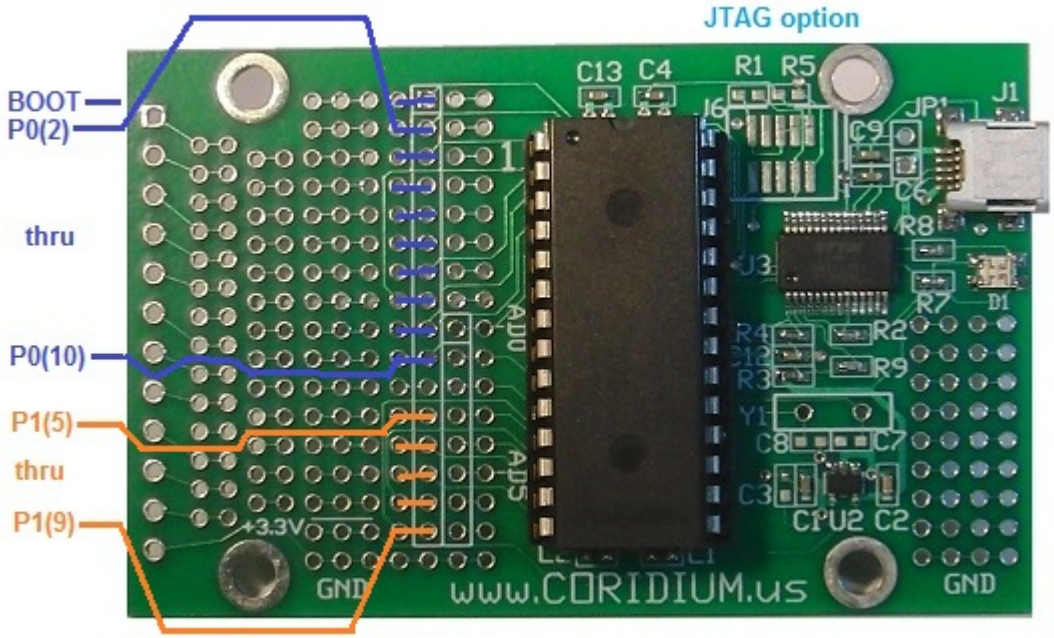
The crystal should be a 32.768 KHz can type, and depending on the rating the capacitors are 0603 size 18-27pF.

If you install this, include the following at the start of your program.

```
#define RTC_CCR   * &HE0024008
RTC_CCR = &H11                          ' clock the RTC with the 32 KHz crystal
```

Rev 4 version of the board makes it easier to add a battery. First cut the trace indicated below, then install a Schottky Diode, 180 ohm resistor and Panasonic ML2020H as shown below. The VL2020/HFN will also work, though it is more expensive and has less power.

BASICboard Pin Diagram



The BASICchip is a complete System on a Chip, all that is required is 2.5 through 3.3V power and GND. Then just wire the available IOs into your application. No extra crystals, external memories, or second supplies required.

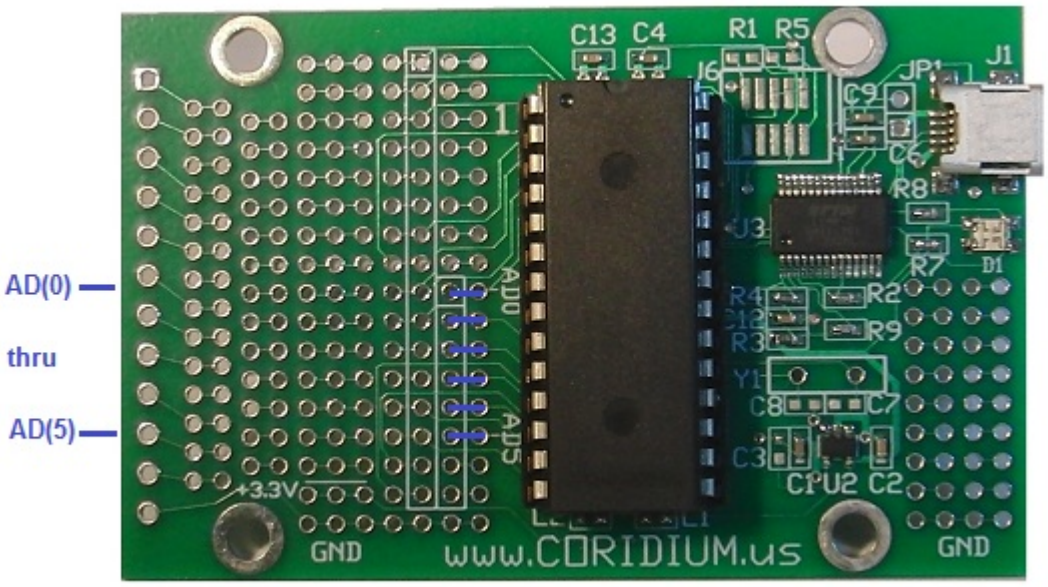
BASIC	function	pin #	alt	notes
IO(39)	TXD(0)	16	P1(7)	Serial Output, TTL compatible (active high) -- debug connection
IO(38)	RXD(0)	15	P1(6)	Serial Input, TTL compatible (active high) -- debug connection
	/RES	23	P0(0)	RESET (internal pull-up) (active low)
IO(1)	BOOT	24	P0(1)	when LOW during reset, ISP is started which disables BASIC (connects to LED and resistors on board)
IO(4) IO(5)	SCL SDA	27 5		open drain outputs, can only pull down, require a pull-up resistor to drive high
IO(2) IO(3) IO(6) IO(7) IO(8) IO(9) IO(10) IO(11)	P0(2) P0(3) P0(6) P0(7) P0(8) P0(9) P0(10) P0(11)	25 26 6 28 1 2 3 4	AD(0)	Input/Outputs -- user controlled - 0-3.3V level 4mA drive when configured as Outputs P0.7 has a 20 mA driver 5V tolerant - use limiting resistor when connecting to a 5V supply
IO(32)	P1(0)	9	AD(1)	Input/Outputs -- user controlled

IO(33) IO(34) IO(35) IO(36) IO(37) IO(40) IO(41)	P1(1) P1(2) P1(3) P1(4) P1(5) P1(8) P1(9)	10 11 12 13 14 17 18	AD(2) AD(3) AD(4) AD(5)	0-3.3V level 4mA drive when configured as Outputs 5V tolerant - use limiting resistor when connecting to a 5V supply
	XTAL	19 20		optional crystal connection -- do not exceed 1.8V
	VDD	21		Power 2.5-3.3V input power -- do not exceed 3.3V
	GND	22		Ground (0V)
	AVDD	7		Analog power, must be equal to or less than VDD
	AGND	8		Analog Ground (0V)

[1]These pins P0(4) and P0(5) are open-drain, when configured as outputs they can only pull down.

Port P1(x) pins can be accessed using the P1(x) keyword. They can also be accessed using IO, IN, OUT, and DIR with indexes 32-41.

Analog Inputs



Power Connections

Optional Crystal

The LPC1114 has an internal 12 MHz oscillator that is trimmed to 1% accuracy. This is good enough for most operations, including serial communication. If more accuracy is desired, then add an optional 12 MHz crystal at Y1 and C7 and C8 with 18pF capacitors.

JP1 option

JP1 in the upper right can have a 2 pin header installed by the user, When this jumper is installed (using a 2 pin shorting block), the control lines from the USB are disabled, so that the board can be plugged into a PC and no matter what state the PC is in, as long as the power is on the BASIC program will start up and run on the ARM.

SuperPRO Pin Description

PROplus Pin Description

The SuperPRO is footprint and pin compatible with the Arduino PRO. In addition it has an onboard 5V regulator so it is compatible with 5V shield boards.

BASIC or C programs can be downloaded using the installed test connector using the USB dongle contained in Coridium's evaluation kit or using the **SparkFun USB Basic Breakout board** or FTDI cable from **MakerShed**. More details on **these connections here**.

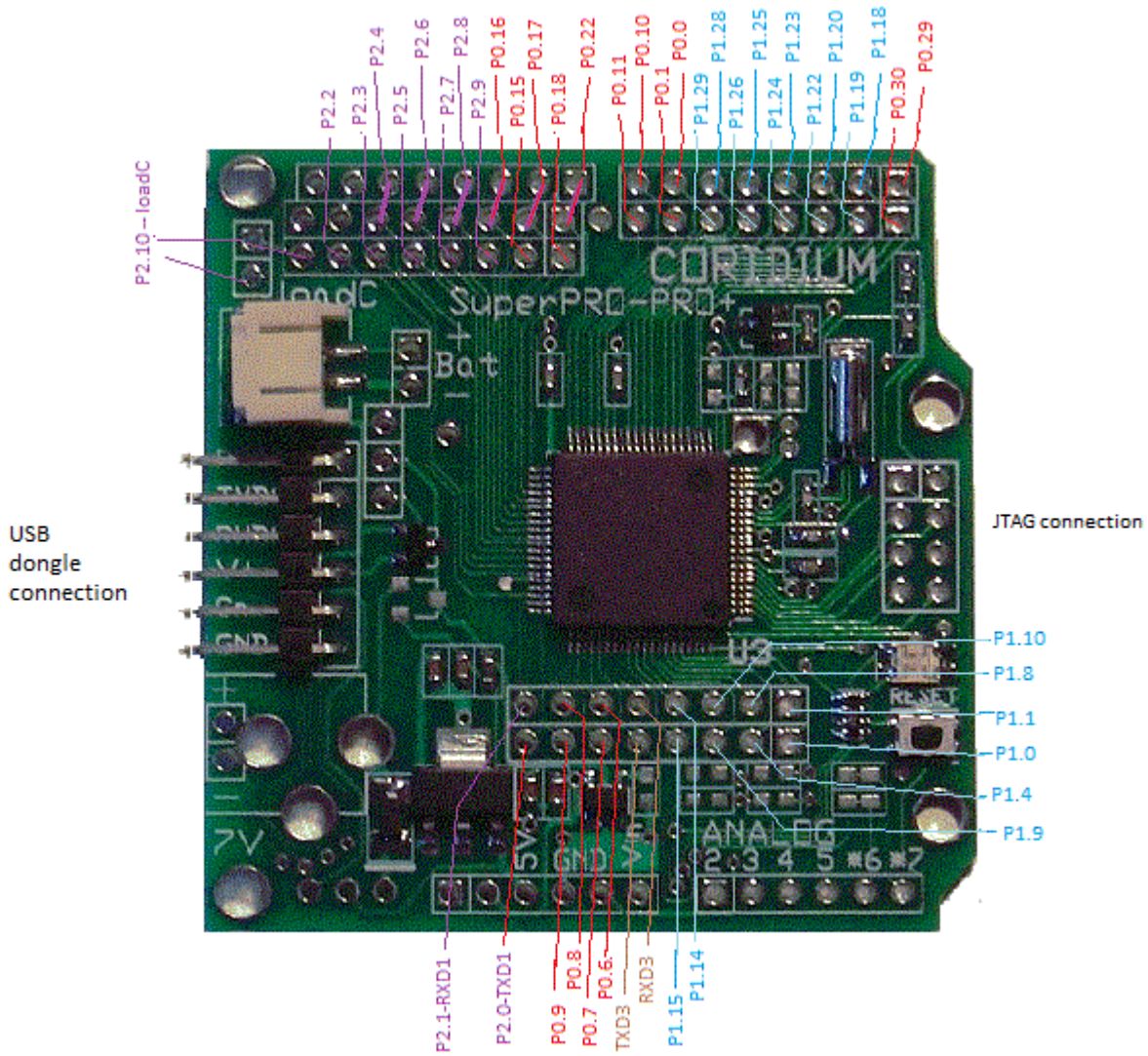
Digital IO connections -- rev 5

The rev 5 adds a parallel connection for pins that are on 0.1" centers. This artwork is also shared with the PROplus version of the board.

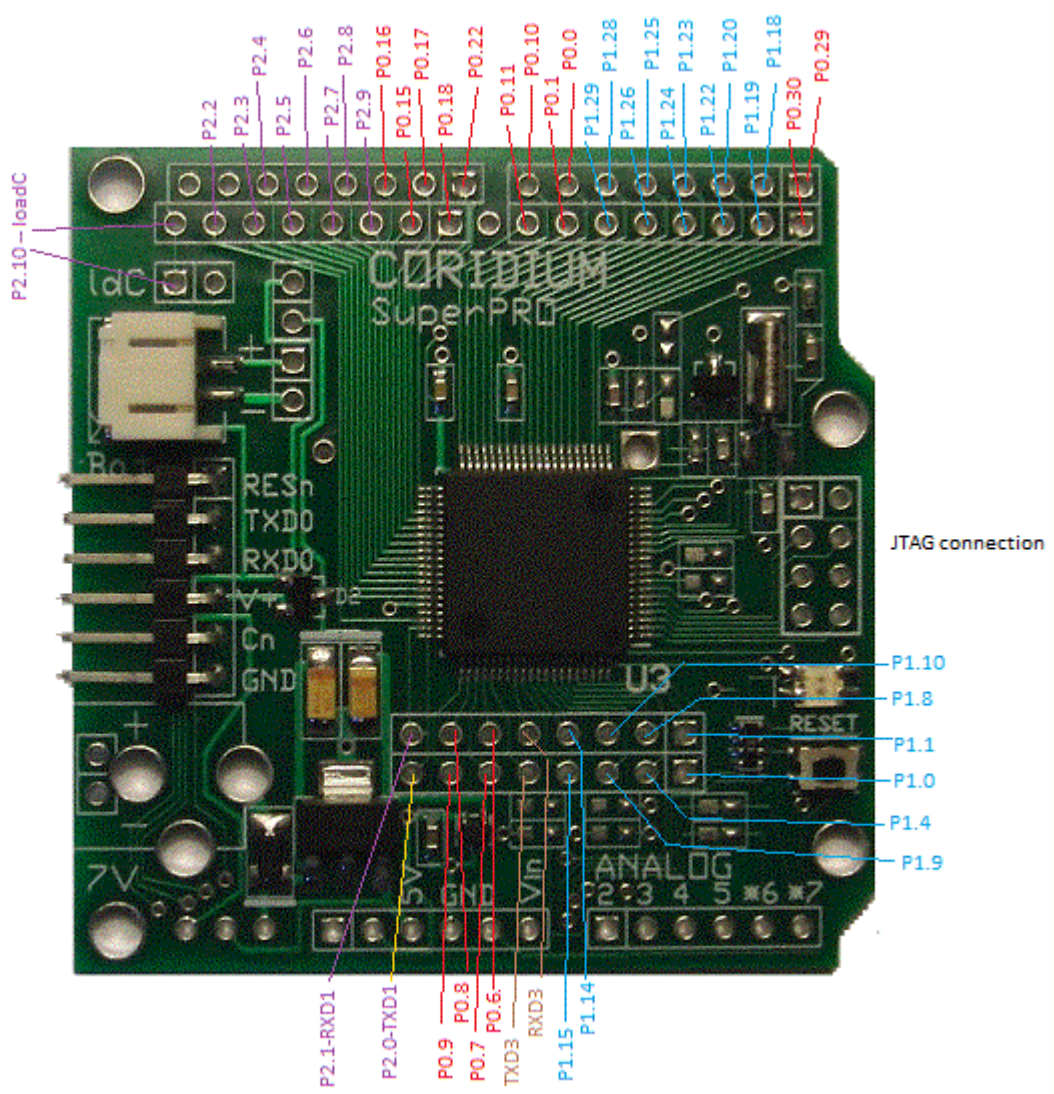
The SuperPRO uses an LPC1756 and has 5V and 3.3V supplies.

The simpler PROplus uses an LPC1751 and has only the 3.3V supply.

Port pins can be controlled by access with the FIOxPIN, FIOxDIR, FIOxSET and FIOxCLR registers. More details on the GPIOs can be found in the NXP User Manual.



Digital IO connections -- rev 4



Special purpose pins

The LPC1756 supports a number of dedicated functions. Those include 4 UARTs, USB, 2 SSPs, 1 SPI, 2 CAN, 2 I2C, I2S, 2 multi-channel PWMs, Quadrature Encoder, dedicated motor control PWM, interrupts, timer counter capture and match.

In addition most can be configured with pullups and default to pullups following reset.

Details can be found in NXP's User manual.

Analog connections

4 A/D converters are readily available. 2 more are available, but share the pins with UART0 -- what was NXP thinking, I have no idea.

1 10 bit DAC is available shared with AD(3) available on the SuperPRO (not on PROplus)

On reset or power up the AD pins are configured by software as AD inputs. To change those to digital IOs, the user must write to the appropriate PINSEL register.

The LPC1756 does support an external reference for the A/D converters, but to use the Arduino AREF pin a jumper is required (details on the schematic)

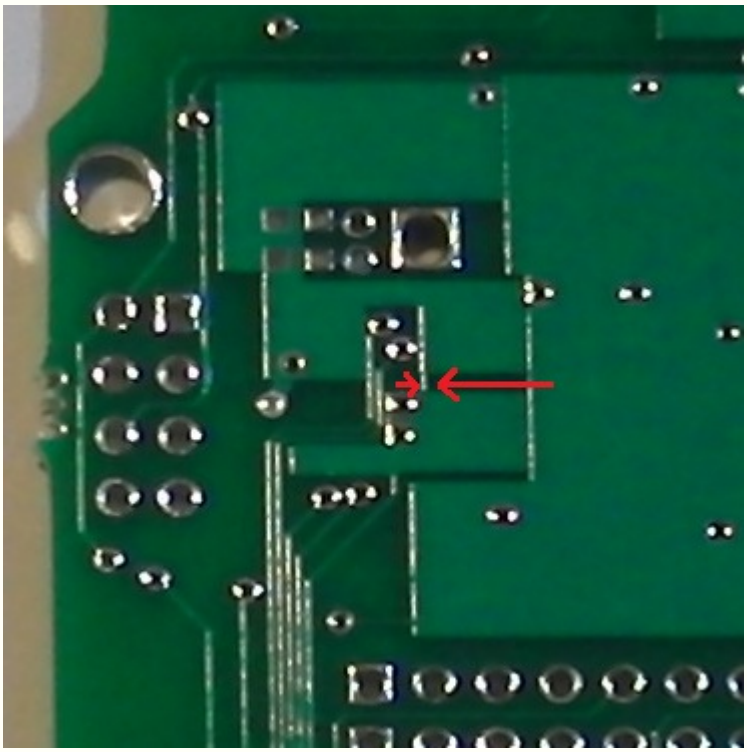
Analog Isolation

The rev 6 and 7 boards isolate both GND and power for the analog section using ferrite beads.

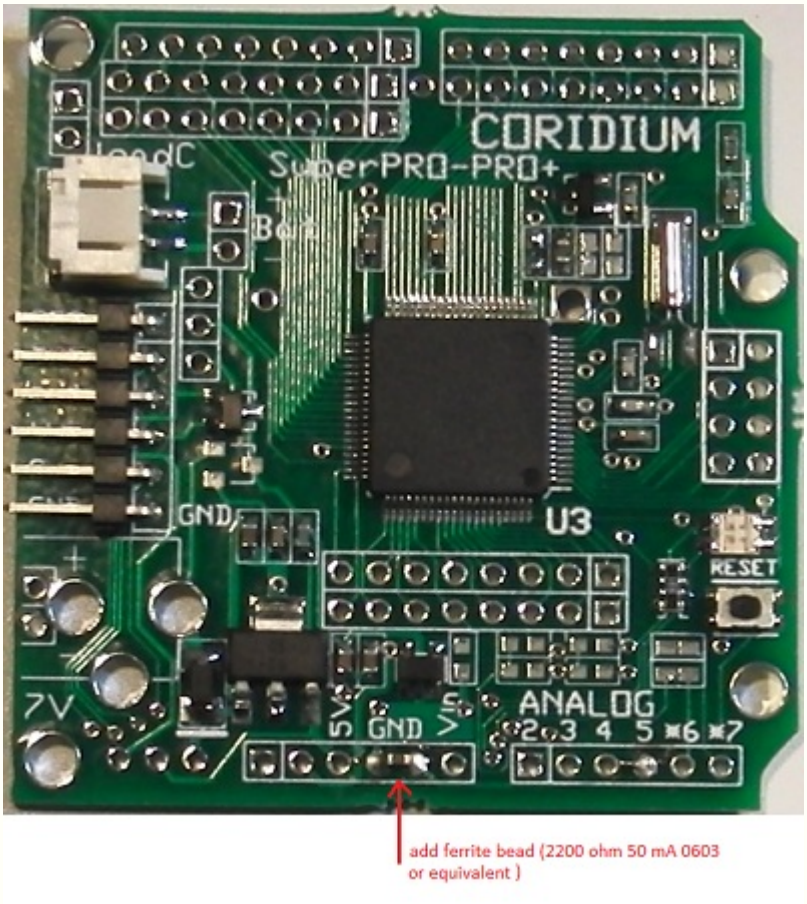
to add isolation to rev 4/5 boards -

The LPC17xx series chips AD converter are sensitive to high frequency noise on the analog GND (Vssa) or on the AD inputs themselves. A symptom that will show up is bits in any bit position turned on/off when the conversion is done. This makes it hard to average out, but conversion can be voted on, choosing 2/3 conversions that agree within a few bits. The occurance of these errors is in less than 1% of the conversions, unless your setup is very noisy.

Another option is to change the analog GND connection on the board. Do this by cutting the trace on the back side between GND under the crystal and the GND connected to Vssa (shown on the picture below)



Then connect digital GND to analog GND using a ferrite bead, a convenient place to do this is on the front side as shown below.



Pin limitations

P0.29 and P0.30 direction control must be done in parallel, they can be both outputs or both inputs, but not mixed.

Power connections -- SuperPRO

The board is shipped with a **2mm power jack compatible** with a JST PHR/S2B or **SparkFun PRT8671** or various battery packs from SparkFun.

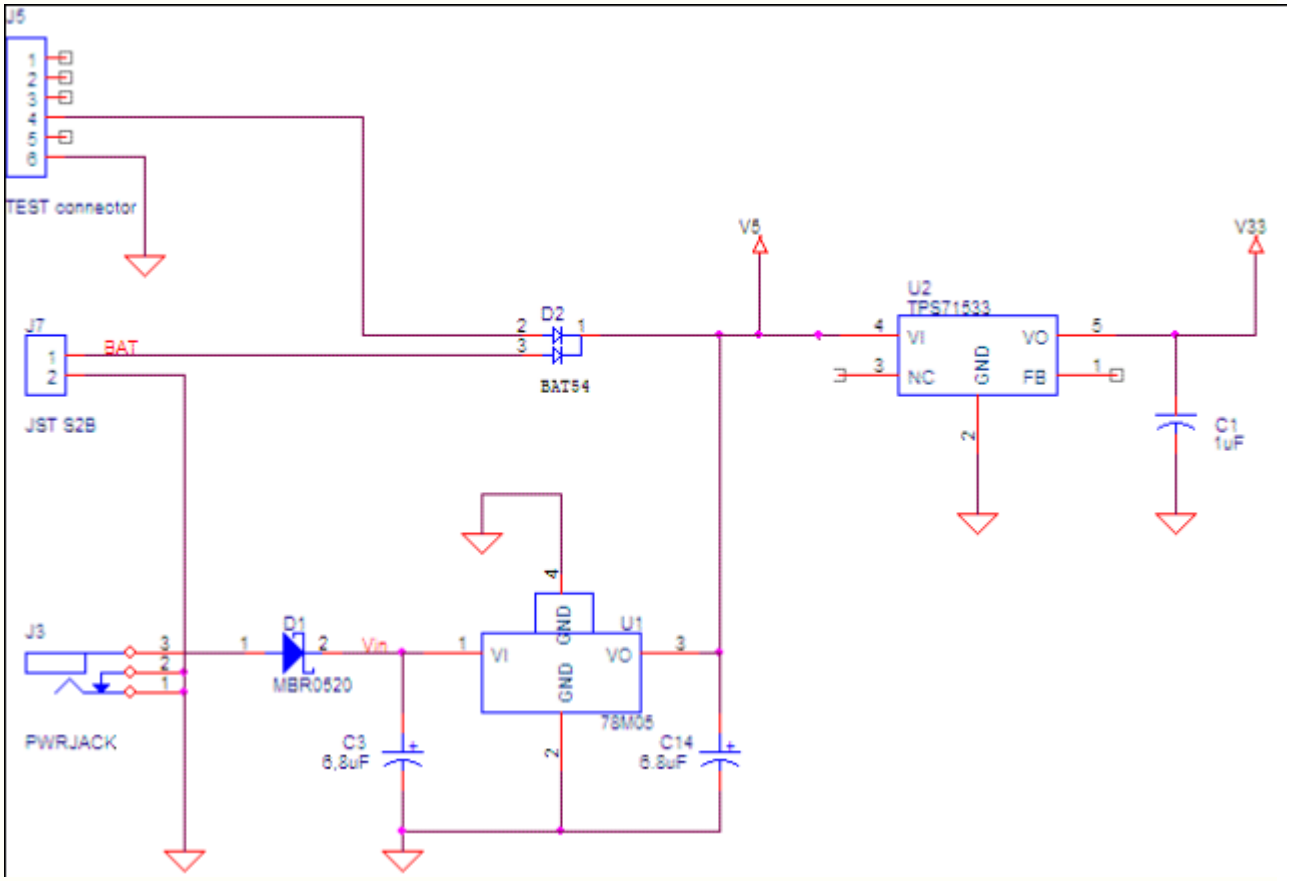
Pads for a Cui PJ-002A or **SparkFun PRT-119** power connector are available in the lower left hand corner.

For both battery and 6V input, 2 pin 0.1" spaced holes are available for wires or headers. When using the battery connector, total current draw for the board must be limited to 200mA. If you want to use more currrent, you should install a jumper around the D2 diode (holes are available above D2).

Diode steering allows power to be supplied from a barrel connector from a 6V unregulated source, 5V USB test connector, or the battery connector. Because of the Schottky diodes, all 3 power sources can be connected simultaneously. If you are using an unregulated wall transformer, you must check the open circuit voltage and it MUST be less than 12V.

When the 6V source is used, 5V Arduino shields can be powered from the SuperPRO.

The schematic below describes this circuit on the SuperPRO



Power connections -- PROplus

The board is shipped with a **2mm power jack compatible** with a JST PHR/S2B or **SparkFun PRT8671** or various battery packs from SparkFun.

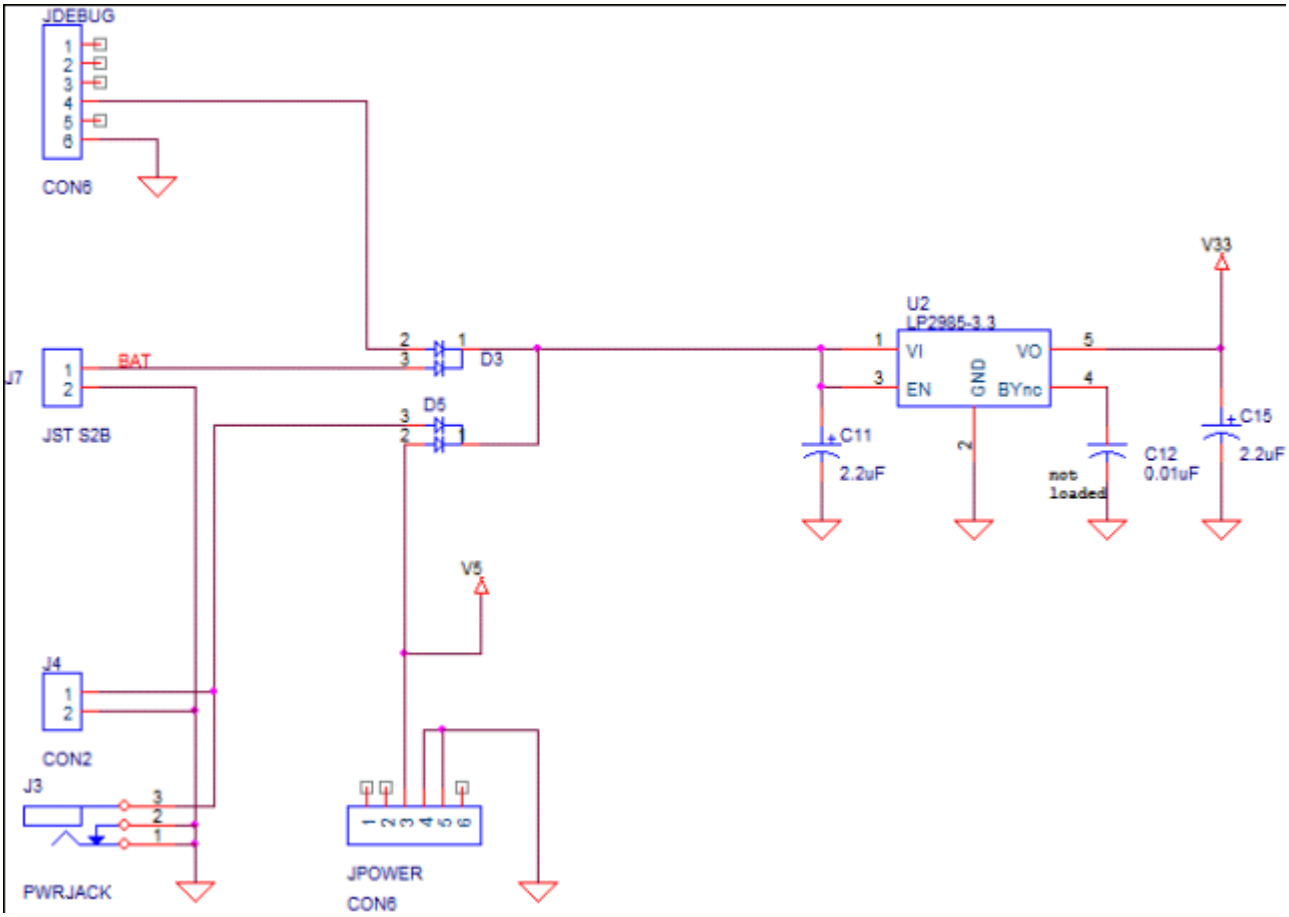
Pads for a Cui PJ-002A or **SparkFun PRT-119** power connector are available in the lower left hand corner.

For both battery and 6V input, 2 pin 0.1" spaced holes are available for wires or headers. When using the battery connector, total current draw for the board must be limited to 200mA. If you want to use more currrent, you should install a jumper around the D2 diode (holes are available above D2).

Diode steering allows power to be supplied from a barrel connector from a 6V unregulated source, 5V USB test connector, 5V from a shield or the battery connector. Because of the Schottky diodes, all 3 power sources can be connected simultaneously. If you are using an unregulated wall transformer, you must check the open circuit voltage and it MUST be less than 12V.

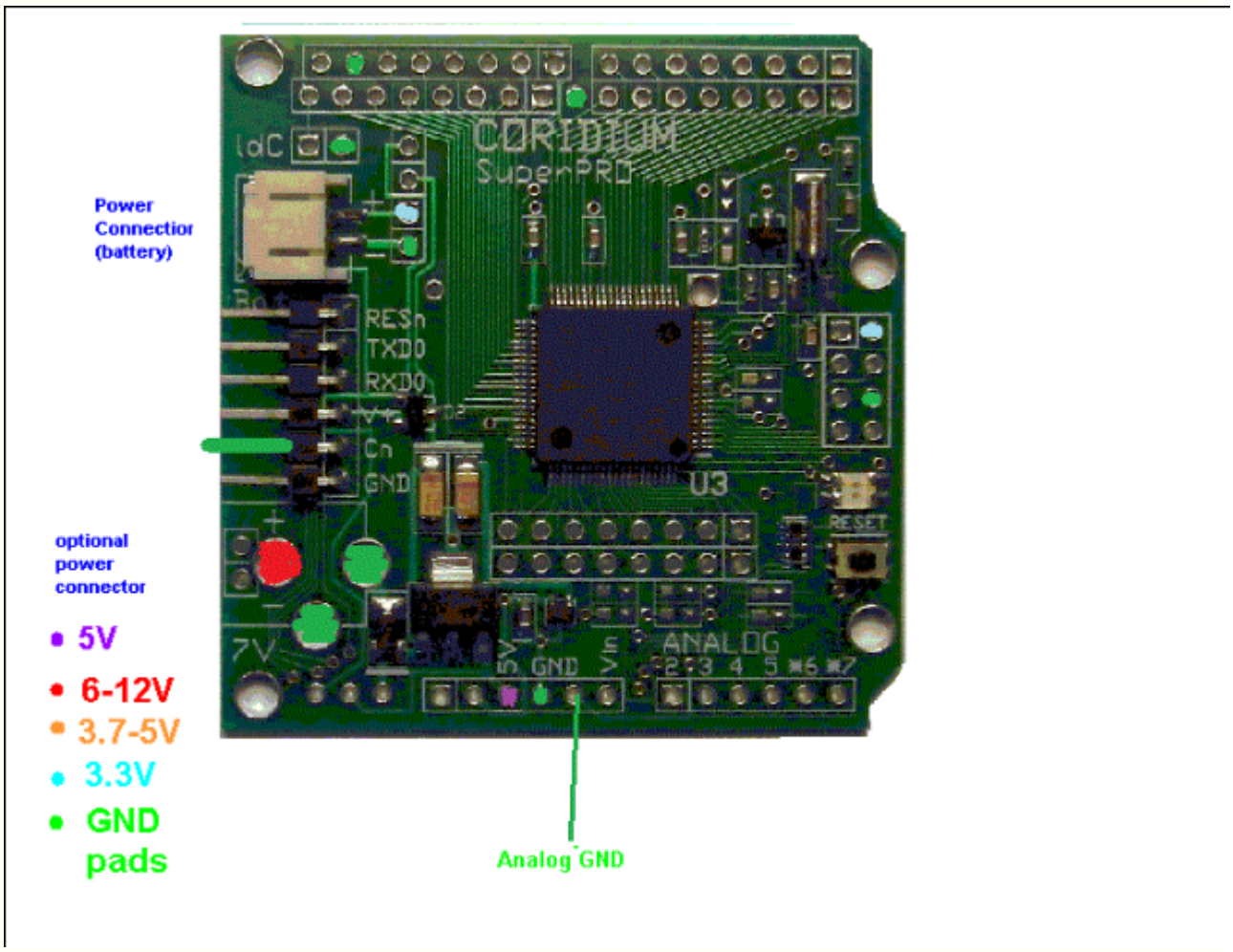
The PROplus only has the 3.3V regulator, so it cannot supply power to a 5V shields.

The schematic below describes this circuit on the PROplus



The **full schematic can be seen here**

Power connections details



The 3.3V regulator can supply 50 mA, with most being used by the LPC2103. The 3.3V connection next to RESn on the lower power connector is only connected if the shorting pads are shorted (NOT the factory default).

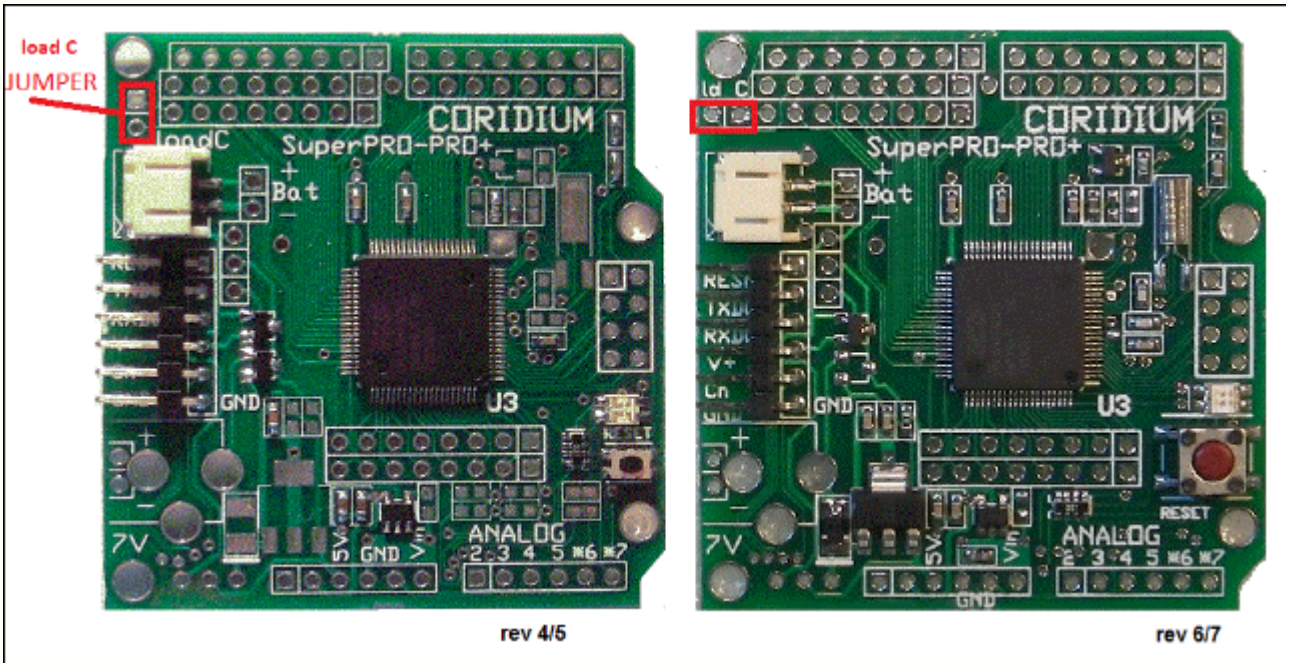
The analog GND should be used to connect to the GND of analog inputs. Digital and Analog GNDs are connected together with a small trace, but to minimize noise you should use the analog GND only for analog signals.

Jumpers and test connector for Program Download

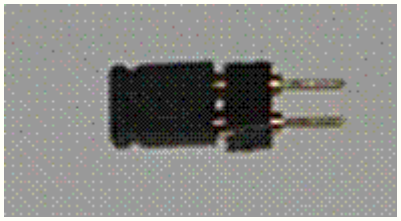
The USB Dongle from Coridium will supply 5V from the USB to power the ARMmite PRO. It also controls the RESET and BOOT signals to automatically load C or BASIC programs using MakeItC or BASICtools. Remember, if you load a C program, it will erase the BASIC firmware and you will not be able to load BASIC programs after that. **It is NOT necessary to install the load C jumper when using the USB Dongle from Coridium.**

When using the SparkFun FTDI Basic Breakout Board, a limited amount of power can be supplied from the BBB, but this is limited to 50 mA and after diode drops, its about 2.8V to the LPC2103. In practice this will run, but it is outside the part specifications, so it should be limited in use.

Also with the SparkFun FTDI Basic Breakout Board to load a C program, the LOAD C jumper needs to be installed, then removed to run the program. BASIC programs can be loaded and controlled using the SparkFun board, with no additional steps/jumpers.



An alternative is to use a 2 pin header with a shorting block (pictured below)

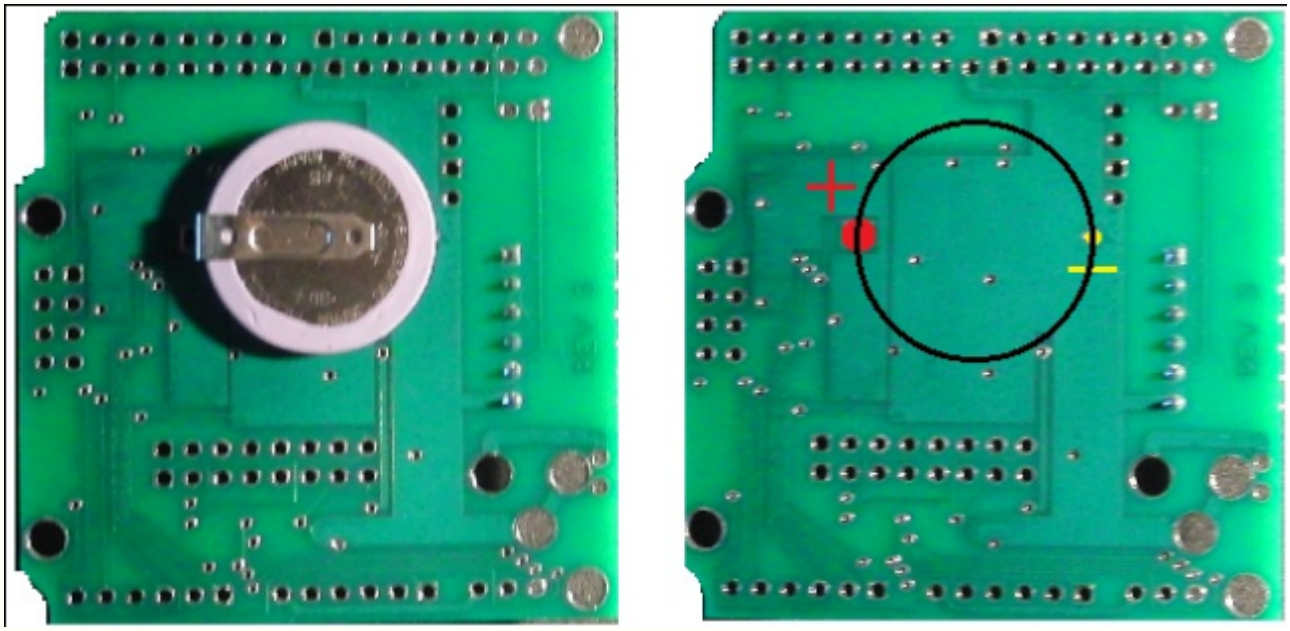


Real Time Clock Oscillator

The RTC oscillator of the LPC17xx parts has been resolved. The first generation parts which were shipped in early 2011 had an unreliable oscillator and this has been corrected by NXP.

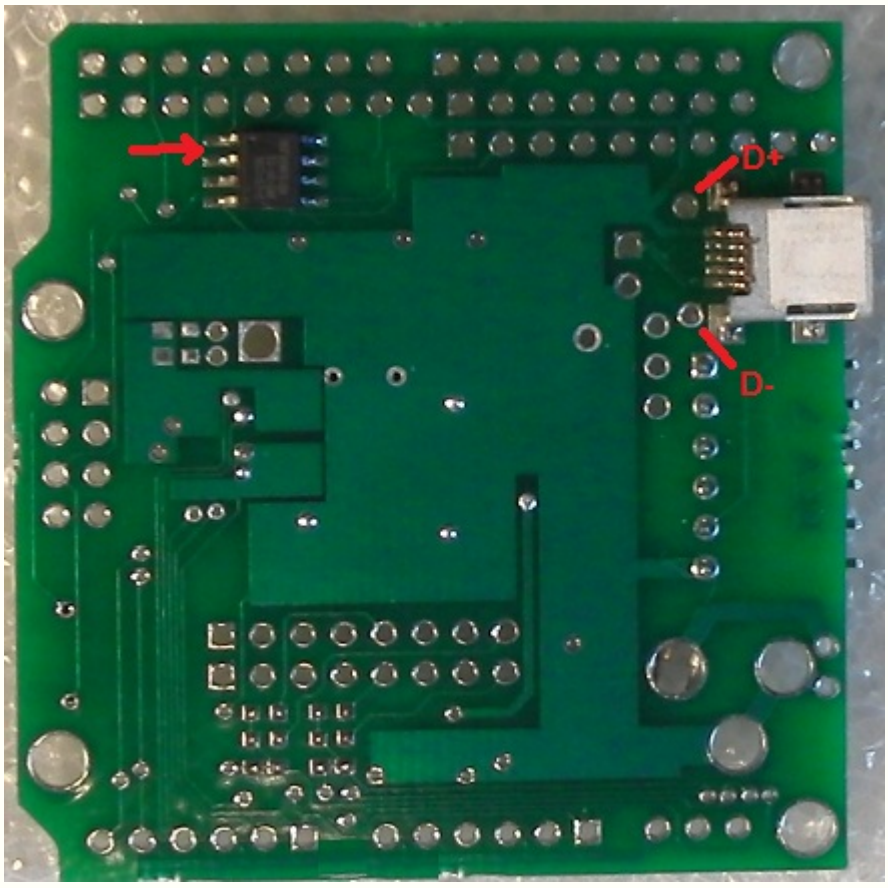
A 32 KHz crystal and diode for battery backup with an optional ML2020 rechargeable Li battery.

A Panasonic ML2020H rechargeable battery may be added to keep the real time clock running when power is removed. The battery is mounted on the back of the board as shown below. The VL2020/HFN will also work, though it is more expensive and has less power.



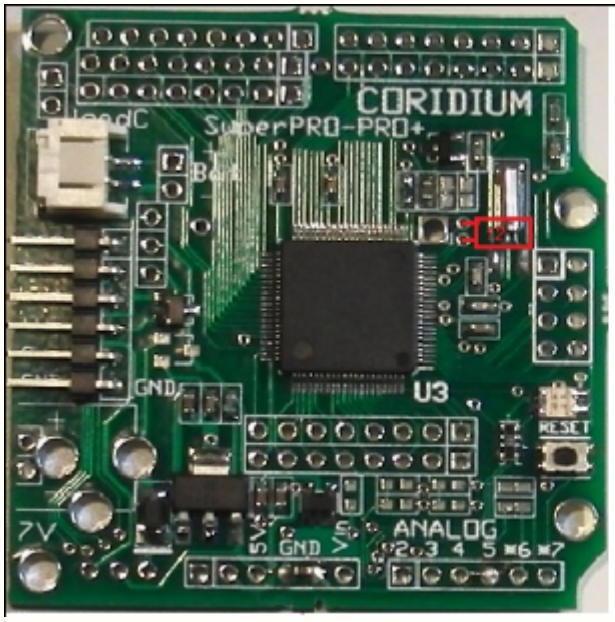
USB connector option for power and SPI Flash option

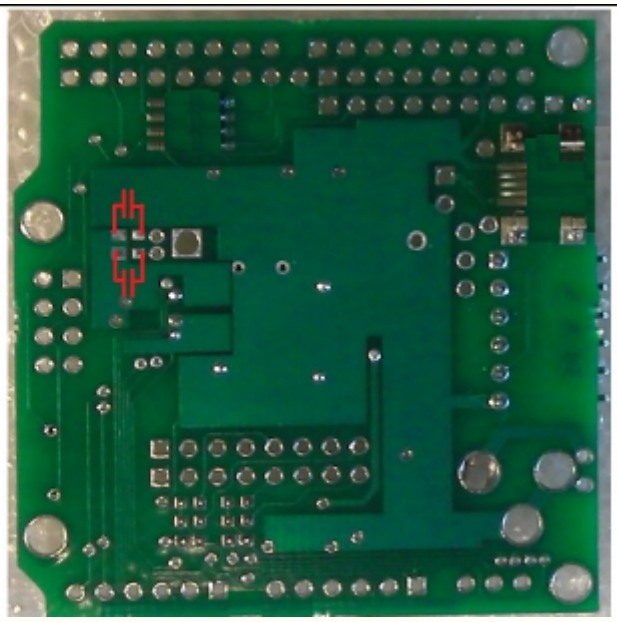
The rev 6/7 boards add pads for an optional SPI Serial Flash (note pin 1 location). Also pads for a USB mini-B connector have been added, this is intended primarily to supply power, and the data lines are connected to pads. These options can be installed at Coridum for orders of 10 boards or more.



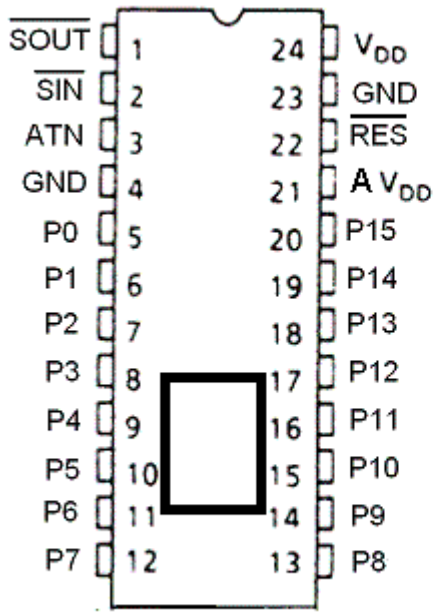
Main Clock Crystal option

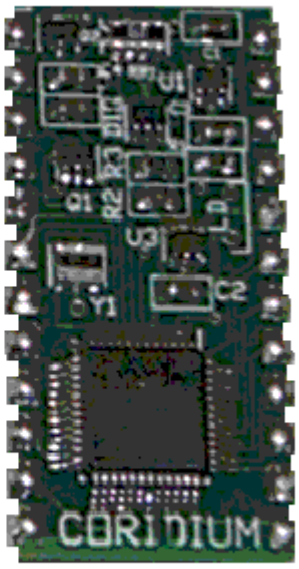
You can add a **12 MHz crystal** with 39pf 0603 load caps, for use as a more accurate clock source.
Locations marked below





ARMexpress Pin Diagram





/SOUT	1		Serial Output, RS-232 compatable (active low)
/SIN	2		Serial Input, RS-232 compatable (active low)
ATN	3		connect to DTR with RS-232, when HIGH reset the Node (active high)
/RES	22		TTL level RESET (open collector with 2.7K pullup) (active low)
P0 P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15	5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20	PWM3 PWM1, RXD1 PWM2, TXD1 AD0 AD2 AD5 AD1 AD6 AD7 PWM7 PWM5 PWM8	Input/Outputs -- user controlled 0-3.3V level 4mA drive when configured as Outputs 5V tolerant
GND	4,23		Ground (0V)
VDD	24		Power 5-12V input power

Alt-VDD	21		Alternate 5-12V input power (for backward compatability) connection to pin 24 is preferred reserved for future expansion

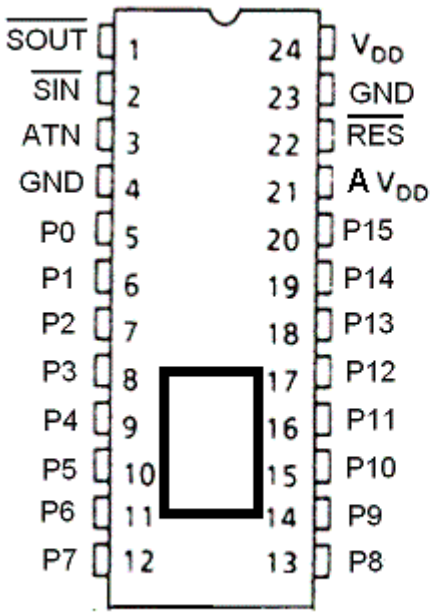
Dual Use AD pins

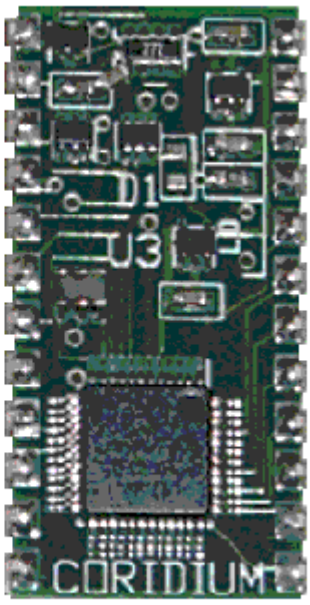
On reset or power up the AD pins are configured as digital IOs on the ARMexpress LITE. When the BASIC accesses these pins they are changed to analog inputs. After that they will remain analog inputs until the next reset or power up.

PWM pins

All pins can be used for the software PWM function, and 6 pins can be used for the hardware driven HWPWM function (HWPWM channels 4 and 6 are not connected).

ARMexpress Pin Diagram





/SOUT	1	Serial Output, RS-232 compatable (active low)
/SIN	2	Serial Input, RS-232 compatable (active low)
ATN	3	connect to DTR with RS-232, when HIGH reset the Node (active high)
/RES	22	TTL level RESET (open collector with 2.7K pullup) (active low)
P0 P1 P2 P3 P4 P7 P8 P9 P10 P11 P12 P13 P14 P15	5 6 7 8 9 12 13 14 15 16 17 18 19 20	Input/Outputs -- user controlled 0-3.3V level 4mA drive when configured as Outputs 5V tolerant
P5 P6	10 11	Input/Outputs -- user controlled open drain when configured as outputs (can only drive low)
GND	4,23	Ground (0V)
VDD	24	Power 5-12V input power

Alt-VDD	21	Alternate 5-12V input power (for backward compatability) connection to pin 24 is preferred reserved for future expansion

DIN rail Pin Description

USB connection shown. Details on the enclosure at **OKW enclosures** .

The ethernet version is software compatible with the **ARMweb, refer to those pages** for more information.

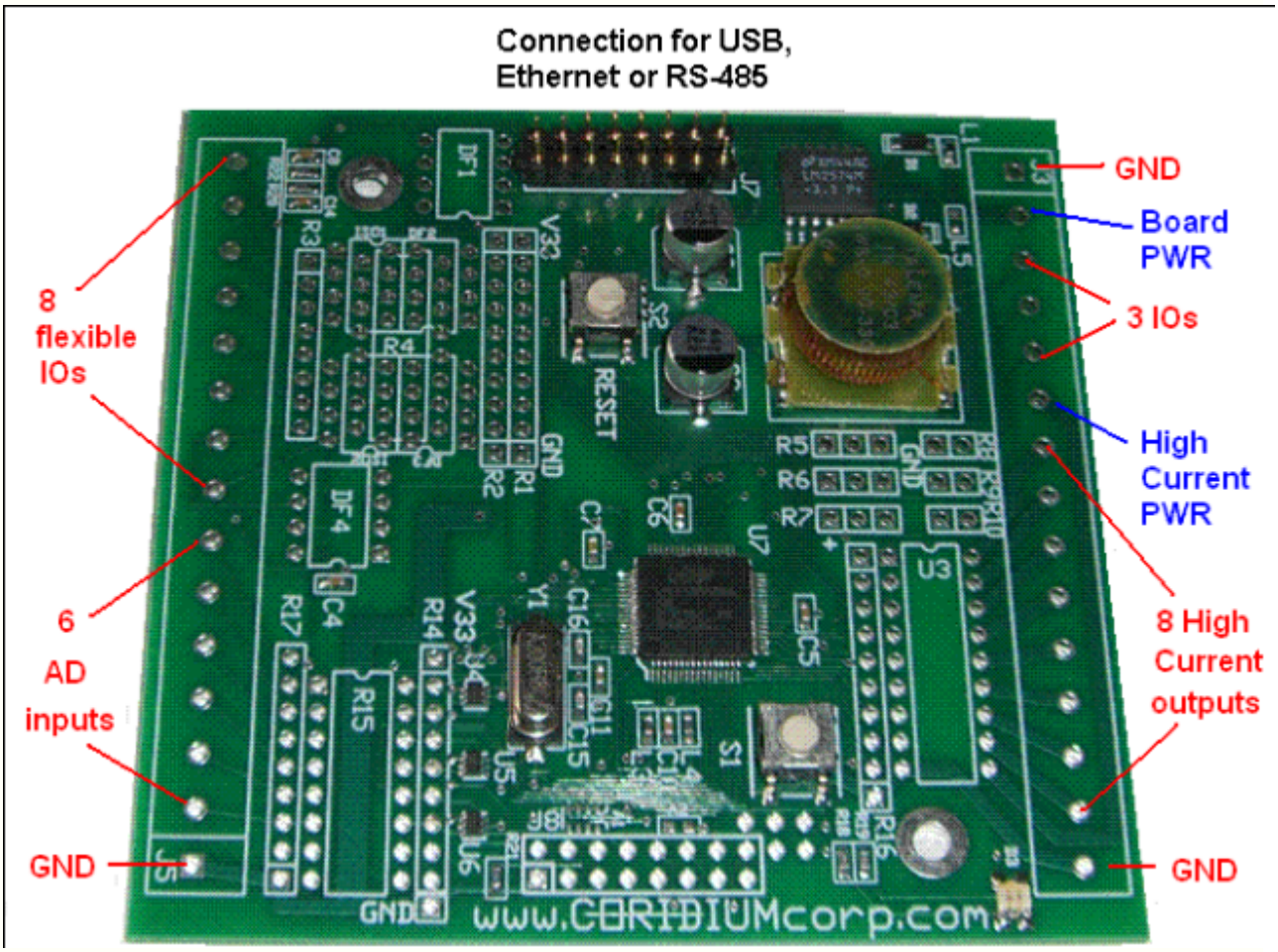
The USB version uses the standalone ARMbasic compiler on the PC.

Rev 1

25 pins available to the user, 6 of which can be analog inputs, 8 high current drivers, 3 digital IOs, and 8 flexible IOs

The LPC2138 is used with 512K Flash and 32K of SRAM.

Optional connections to USB, 10Mb Ethernet, or RS-485 (with optional isolation)



picture shown without screw terminals for clarity

Power Inputs

Board 7-40V DC. This voltage is reduced with a switching regulator for the 3.3V internal board supply.

High Current Driver (**ULN2803**) 5-50V. This can be a seperate supply from the Board input power, or can be the same supply. It is a required connection for relay drivers to provide a path for current when the relay coil is turned off, it does not have to be the power supply for the board in this case, but it can be.

For volume customers the power supply can be stuffed to accept a regulated 3.3V supply directly, this is done by omitting the switching power supply and adding an appropriate ferrite bead at L5.

Schematic

The schematic is too large to include on this page, but is downloaded into the /Program files/Coridium/Schematic directory. **Is also available here.**.

Enclosure

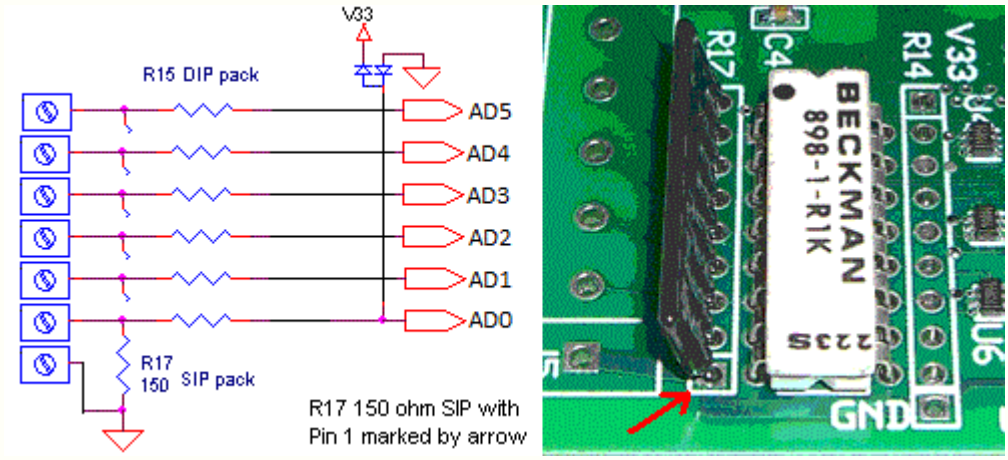
OKW B6704100 The kits include custom cutouts for either Ethernet or USB connections. Mechanical drawing for the **enclosure is here** ,

All the following options can be configured by the user, by optionally stuffing the through-hole components in the DIN rail kit. Coridium will configure boards when 10 or more are ordered.

6 AD pins

These may configured for 4-20 mA sensors, with resistor dividers, or as digital inputs. These inputs have diode clamps to 3.3V and GND.

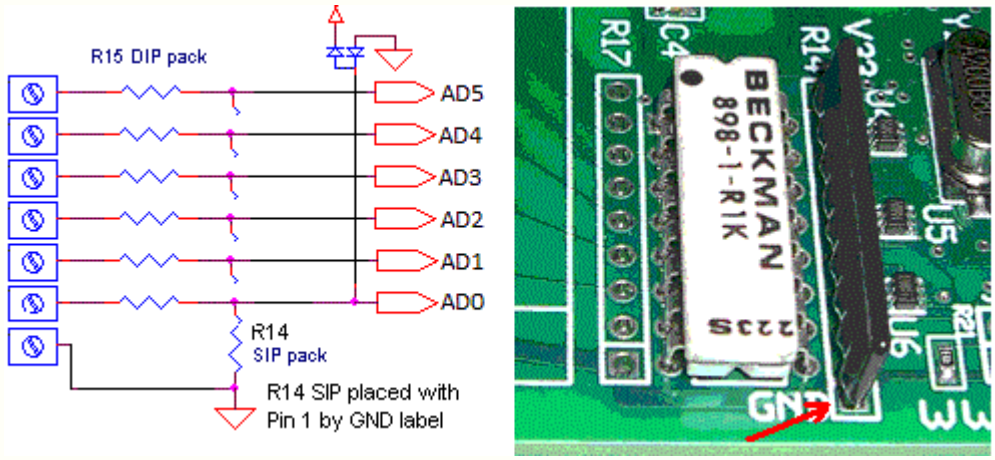
4-20mA sensor --



load 150 ohm SIP into R17

suggested components
  Bourns 4600X Bussed SIP resistor
  Bourns 4100R Isolated DIP resistor

A/D resistor divider --

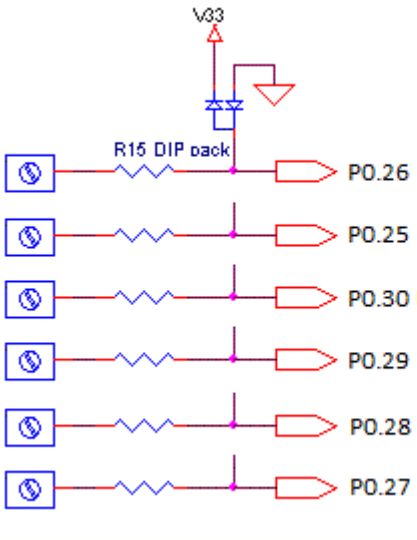


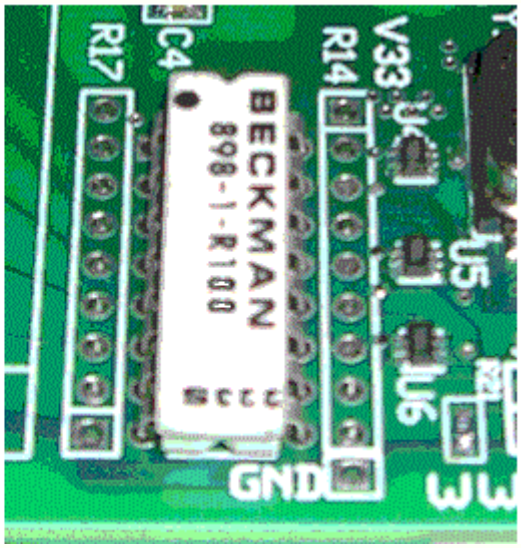
load R15 DIP resistor and R14 SIP with appropriate values

AD = Vin * R14/(R14+R15)

Source impedance to AD should be less than 10K.

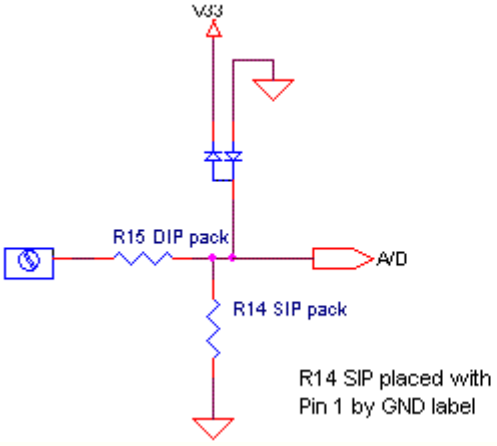
digital IO --

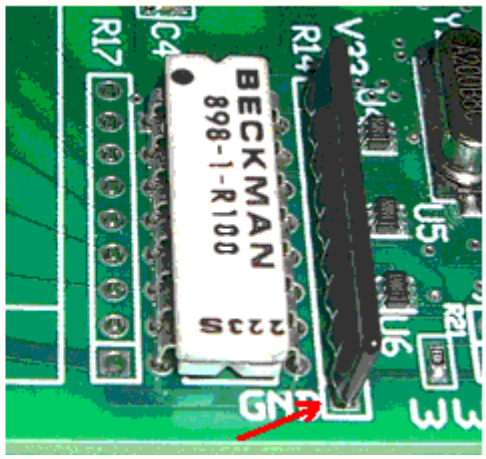




load R15 with 100 or 1K

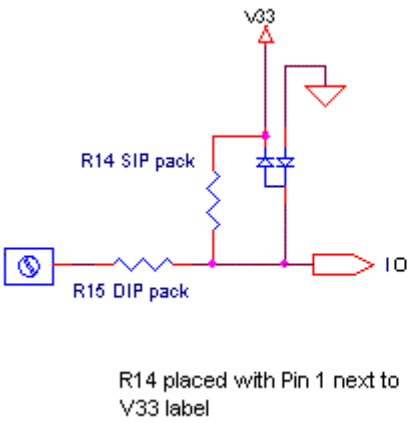
digital IO (pulldown)--





load R15 with 100, R14 with 10K

digital IO (pullup) --

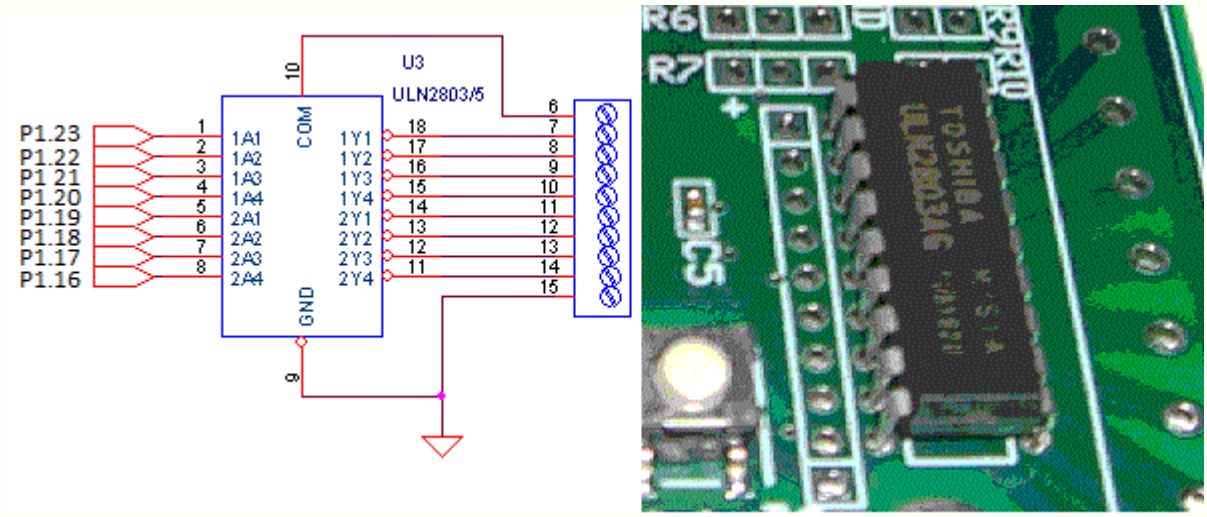


load R15 with 100, R14 with 10K

High Current Drivers

These may use a high sink current driver, or configured as digital IOs with optional pullups or pulldowns

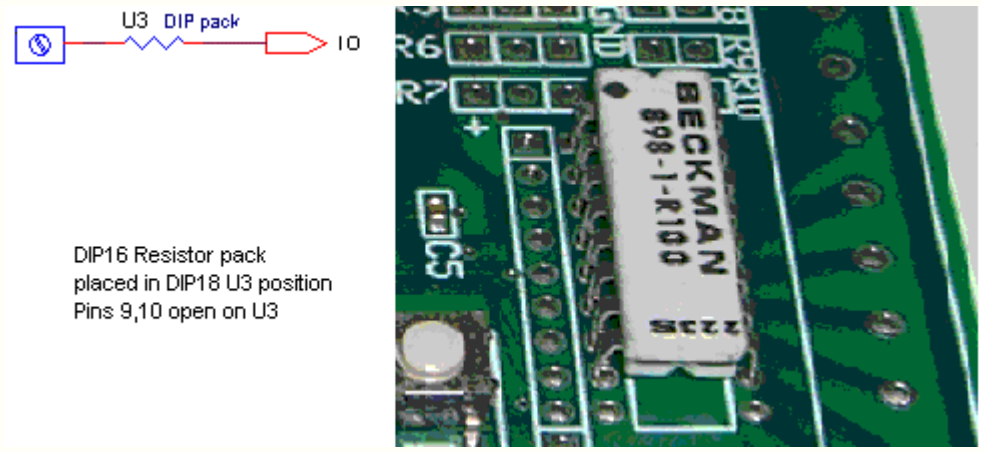
High Current drive --



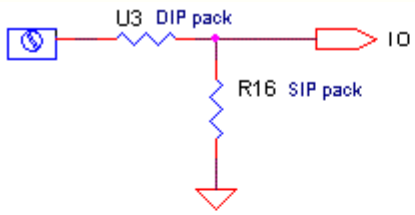
This driver can sink a surge current of 500mA upto 50V, this driver is a **ULN2803** .

suggested components
TI ULN2803AN
Toshiba ULN2803APG
STmicro ULN2803A

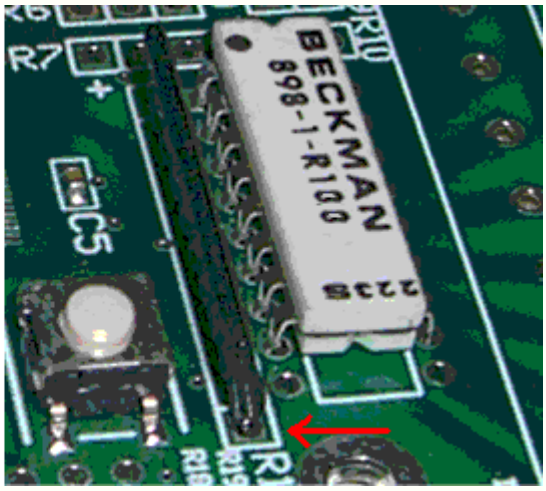
digital IO --



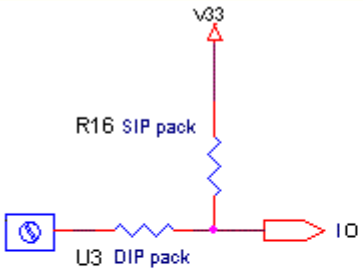
digital IO (pulldown) --



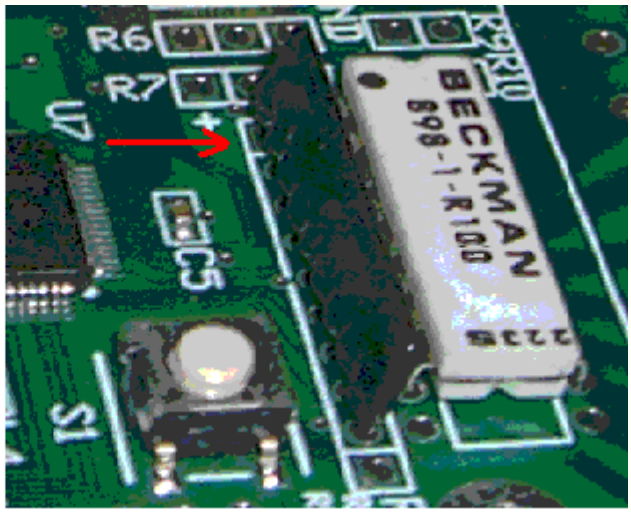
Pin 1 of SIP resistor loaded
near R16 label (arrow)



digital IO (pullup) --



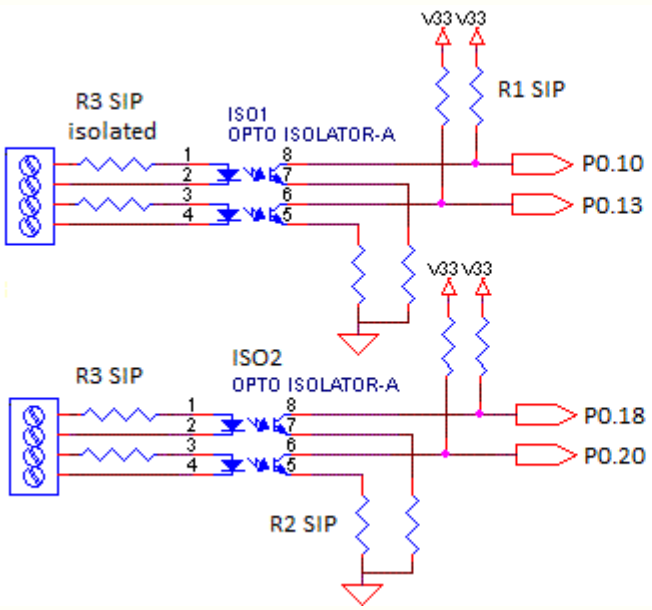
Pin 1 of SIP pullup loaded
near + sign (arrow)

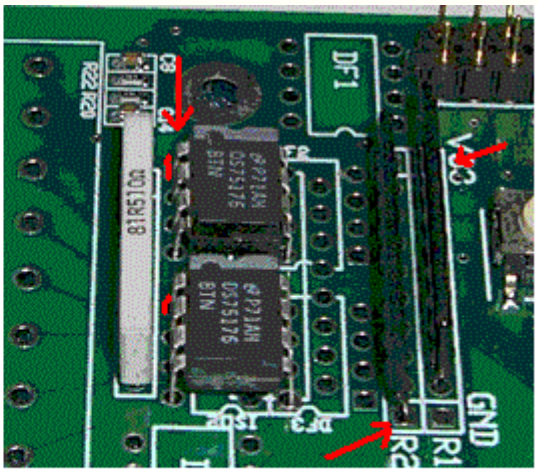


Flexible IOs

These may be configured as 8 digital IOs (with and without pullup/pulldown), opto-isolated inputs or outputs, or differential inputs or outputs. They are arranged in 2 groups of 4 so that there can be 2 opto-isolated input and 2 opto-isolated outputs.

opto-isolated input --

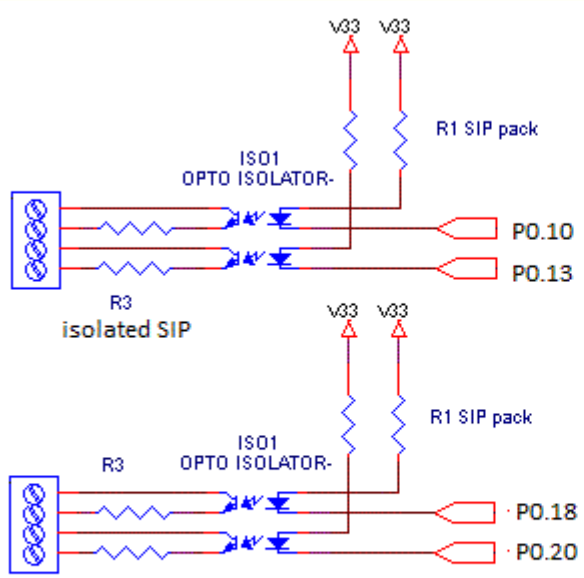


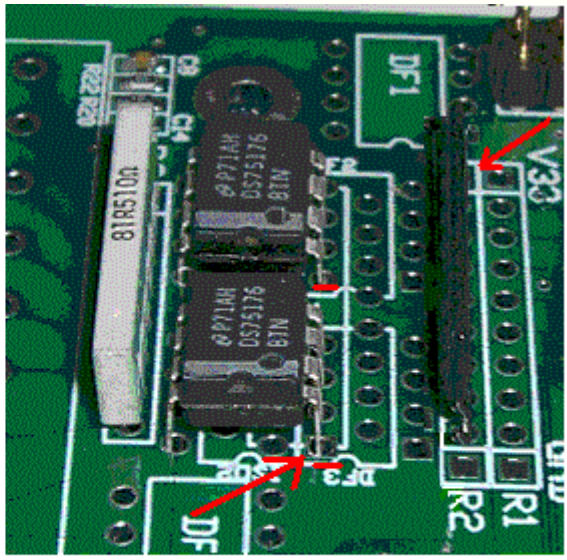


Note Pin 1 orientation of opto-isolators
Pin 1 of R2 near GND, Pin 1 of R1 near V33

suggested components
Liteon LTV-827
Fairchild MCT9001
Toshiba TLP621-2

opto-isolated output --



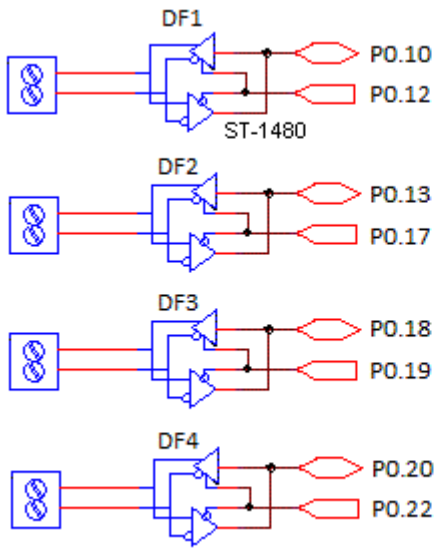


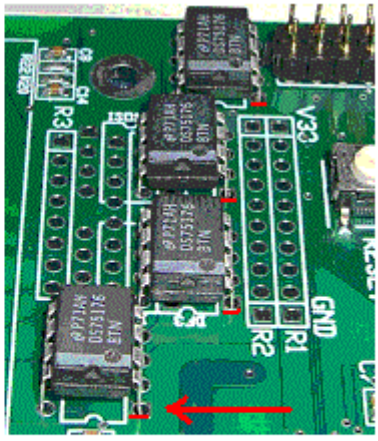
Note Pin 1 orientation of opto-isolators

Pin 1 of R2 near V33 marking

same components as above, rotated 180 degrees

bidirectional RS-422 driver --

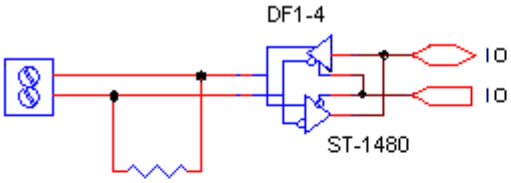




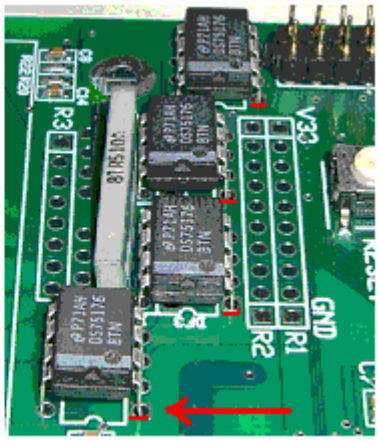
Note Pin 1 orientations

suggested components
  National DS75176BN
  TI SN75176AP

bidirectional RS-422 driver with termination --

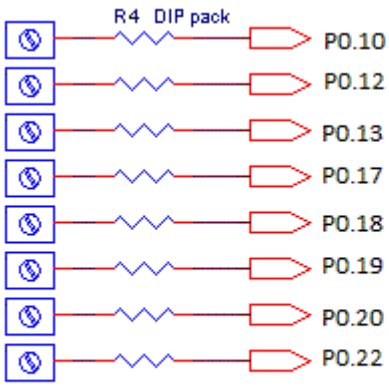


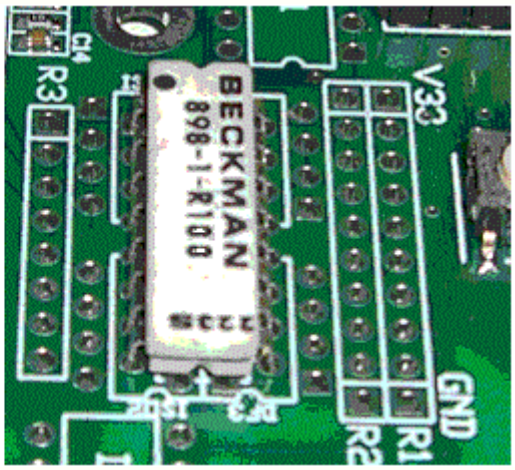
Note Pin 1 orientations



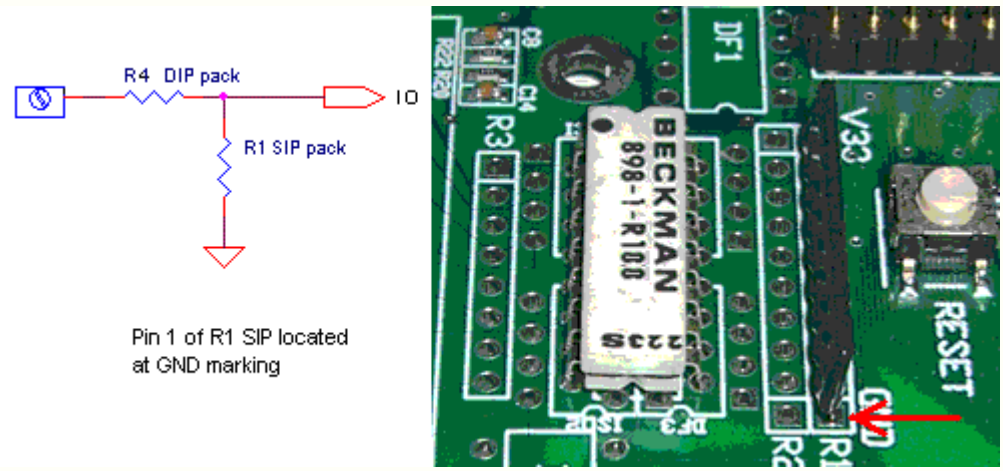
suggested components
  Bourns 4600 Isolated SIP resistor

digital IO --

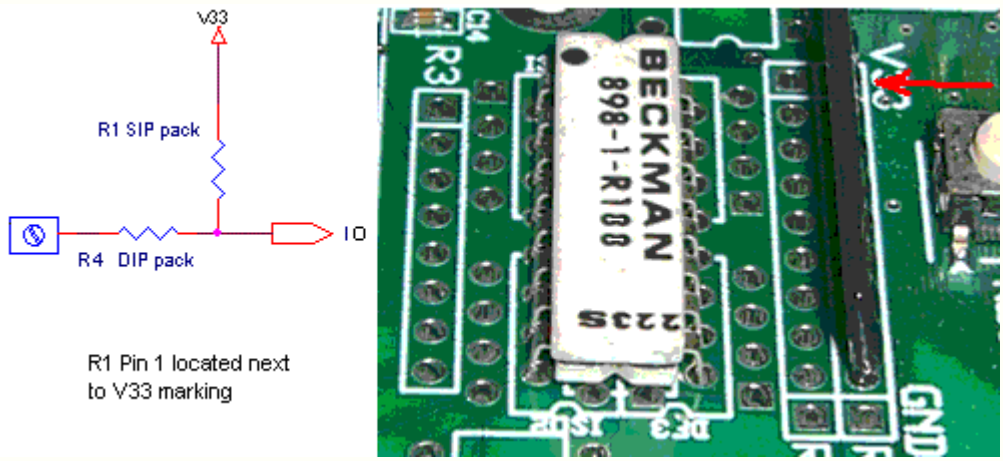




digital IO (pulldown) --



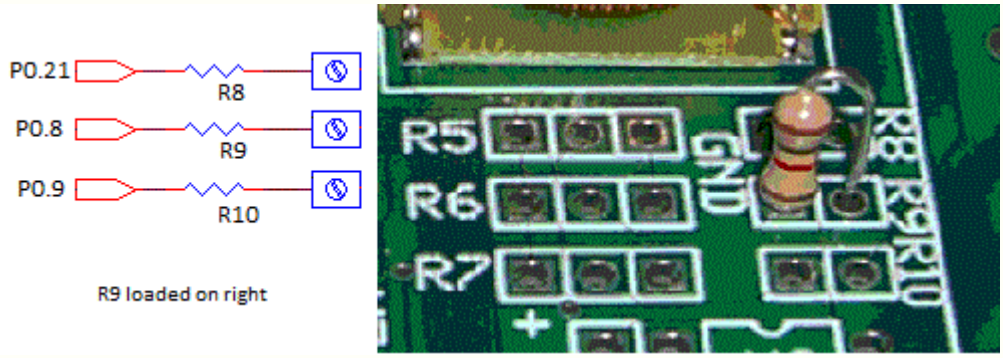
digital IO (pullup) --



3 digital IOs

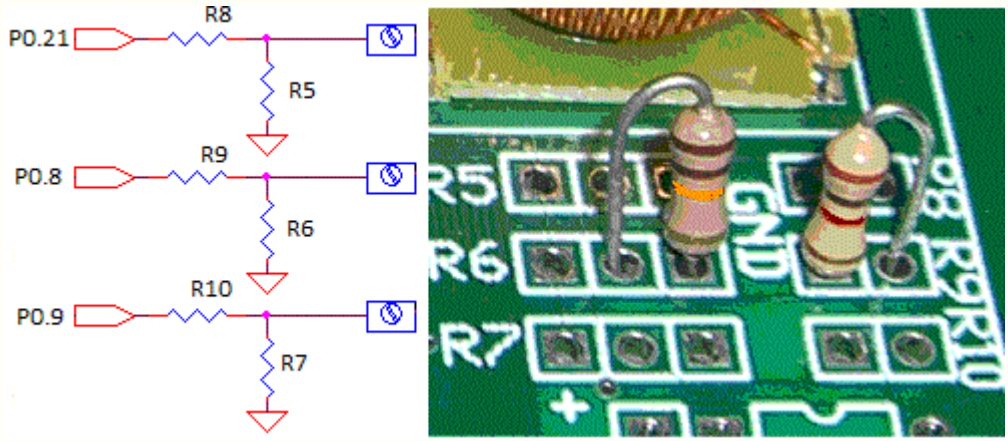
These may be configured as staight thru, or with pullups or pulldowns

digital IO --



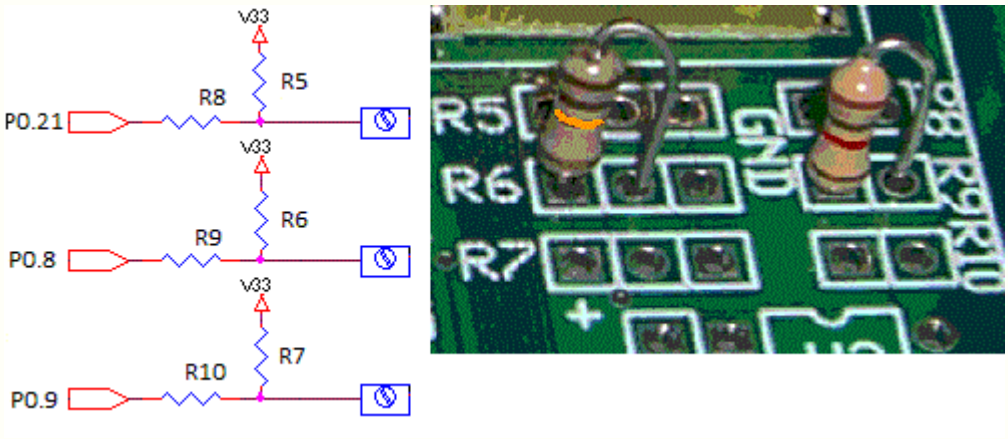
shown with 100 ohm series

digital IO (pulldown) --



shown with 10K pulldown and 100 series

digital IO (pullup) --

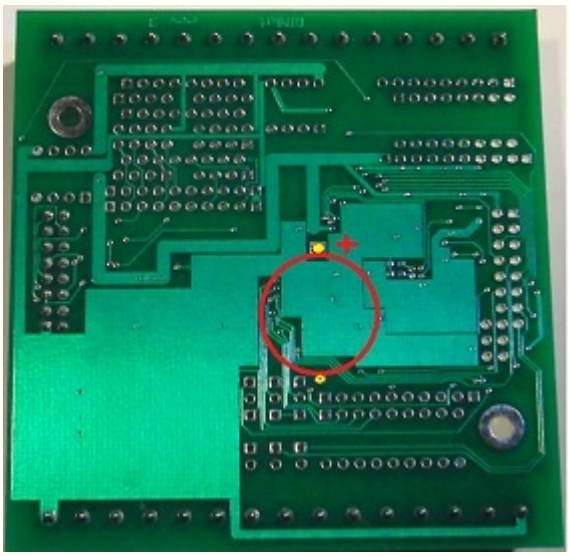


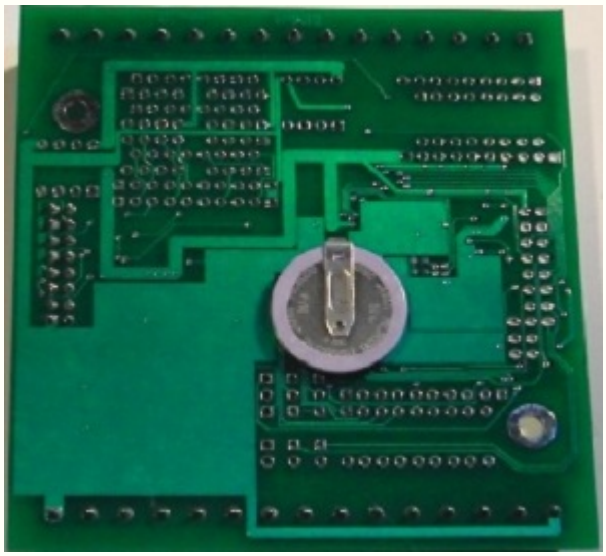
shown with 10K pullup and 100 series

RTC options

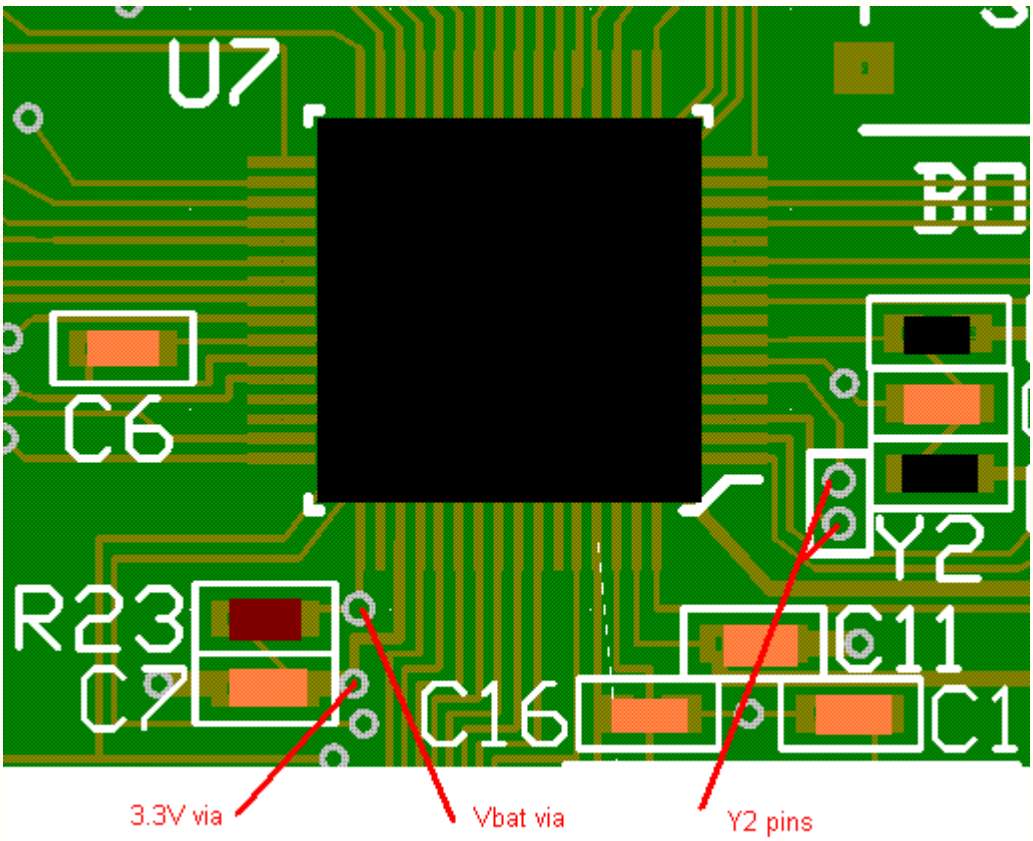
Rev 3

This revision adds the diode and resistor needed for charging an ML2020 battery. That battery can be mounted on the backside of the board as illustrated below



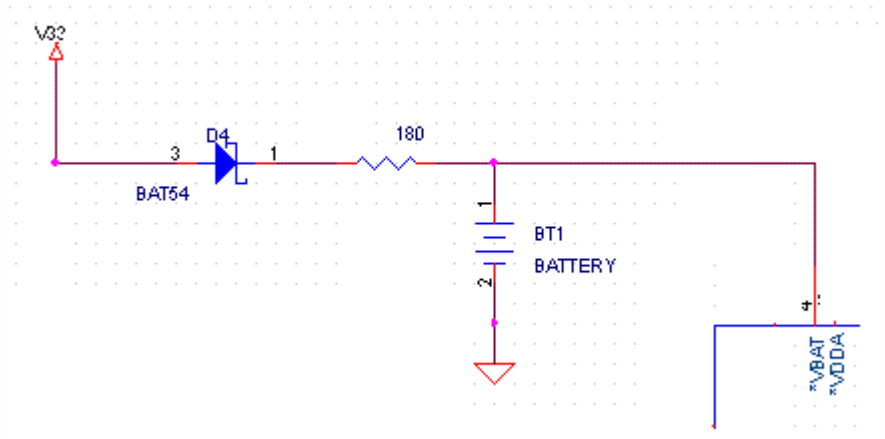


Rev 2



To connect a battery, remove R23, and use the Vbat via to connect, a resistor-Schottky diode-battery connection (suggested schematic below)

GND and 3.3V are available on either side of C7



A 32 KHz crystal (such as the Citizen CMR200TB32.768KDZFTR) can be connected at Y2, with the two 22pF startup caps on the bottom/circuit side of the board.

Schematics

PDF copies of the schematics are copied into the Program Files/Coridium/Schematics directory when you install either the BASIC or C tools.

Or you can follow these links to PDF schematics on the Coridium website.

- ARMmite schematic
  - ARMmite rev 2 schematic
- ARMmite PRO schematic
- PROplus schematic
- Super PRO schematic
  - PROplus / SuperPRO rev 4/5 schematic
  - USB dongle schematic
- ARMexpress LITE schematic
- ARMexpress schematic
  - ARMexpress Eval PCB
- ARMweb schematic
  - ARMweb rev 3 schematic
- DINkit schematic
  - DINkit USB board
  - DINkit Ethernet board

DXF files are mechanical drawings of the boards, they are also available from these links or in the Schematics directory.

- ARMmite mechanical
- ARMmite PRO mechanical
- ARMweb mechanical

CPU details

These are links to detailed documentation for the CPUs used in the ARMexpress and ARMmite products. These files are at the NXP website. The links may move so if they are broken here, search their site www.nxp.com

LPC2103 used in ARMmite and ARMexpress LITE and ARMmite PRO

LPC2103 data sheet

LPC2103 User manual

LPC2106 used in ARMexpress

LPC2106 data sheet

LPC2106 User manual

LPC2138 used in ARMweb

LPC2138 data sheet

LPC2138 user manual

LPC1756 used in Super PRO

LPC1756 data sheet

LPC1756 user manual

Power On Behavior

Initial Power on conditions

On power up all pins are tri-stated on the ARMexpress or ARMmite.

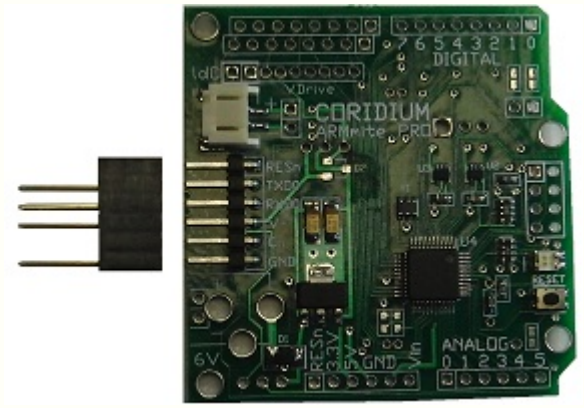
Restarting the program

If the user has entered a BASIC program into the ARMexpress/ ARMmite, that program will be started when the power is applied, or restarted when RESET is asserted either with the pushbutton, or from the BASICtools program via asserting the DTR line (low on ARMmite, high on ARMexpress).

If the user program ends by getting to the last statement of the program or executing an END instruction, the ARMexpress/ARMmite will power down and await either input on the debug serial port, or a RESET.

Reset and Boot for PRO boards

For the PRO, PROplus and SuperPRO boards when connecting a PC to a board that is running, the reset and boot control signals will be toggled by the PC. This is a function of Windows and the Drivers. This will reset the board or possibly put it into a load program state. To avoid this you can disconnect the Reset and Boot signals from the USB dongle, either by cutting pins or making an adapter using a 6 pin female header with long pins**(available from SparkFun**).



ISP (bootloader) checks

ISP bootloader

To load a C program, the NXP ISP bootloader is invoked. This is also used by the Connection test in TclTerm and BASICtools.

Connections

Whether its part of the PRO family or has a builtin USB connection, 6 wires make up the primary connection to an ARM for all Coridium products.

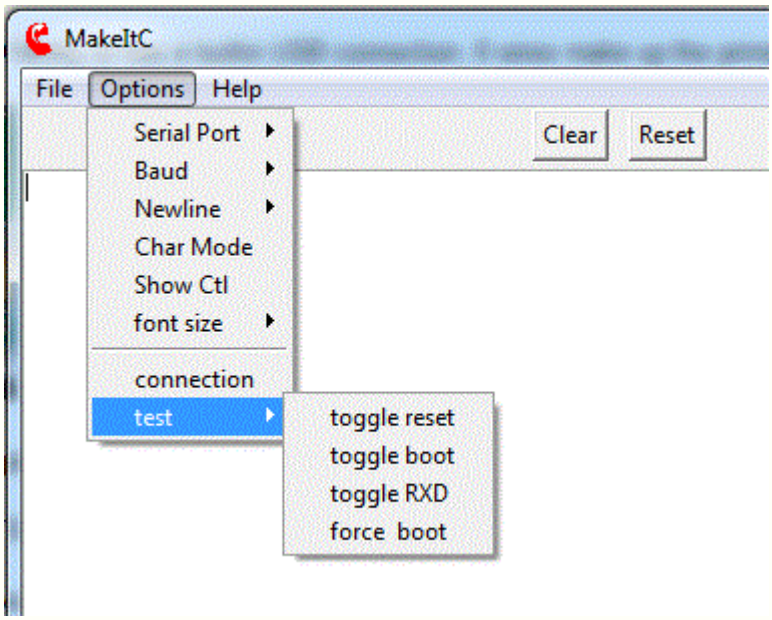
Those connections include power (V+) and GND.

Serial connections to UART0 (TXD0 from the ARM to the PC, RXD0 from PC to the ARM).

RESn is the reset line (active low -- places the device in reset)

Cn is held low on P0.14, P2.10 or P0.1 low during RESET, to place the part in ISP mode, but its normal state is high.

Controlling these pins from TclTerm and BASICtools



You can use a meter to check the states of these pins.

RESn should normally be a logic 1. And you will see it change to a 0 and back when toggle reset is selected.

Toggle boot will drive Cn low and high (high being the normal state).

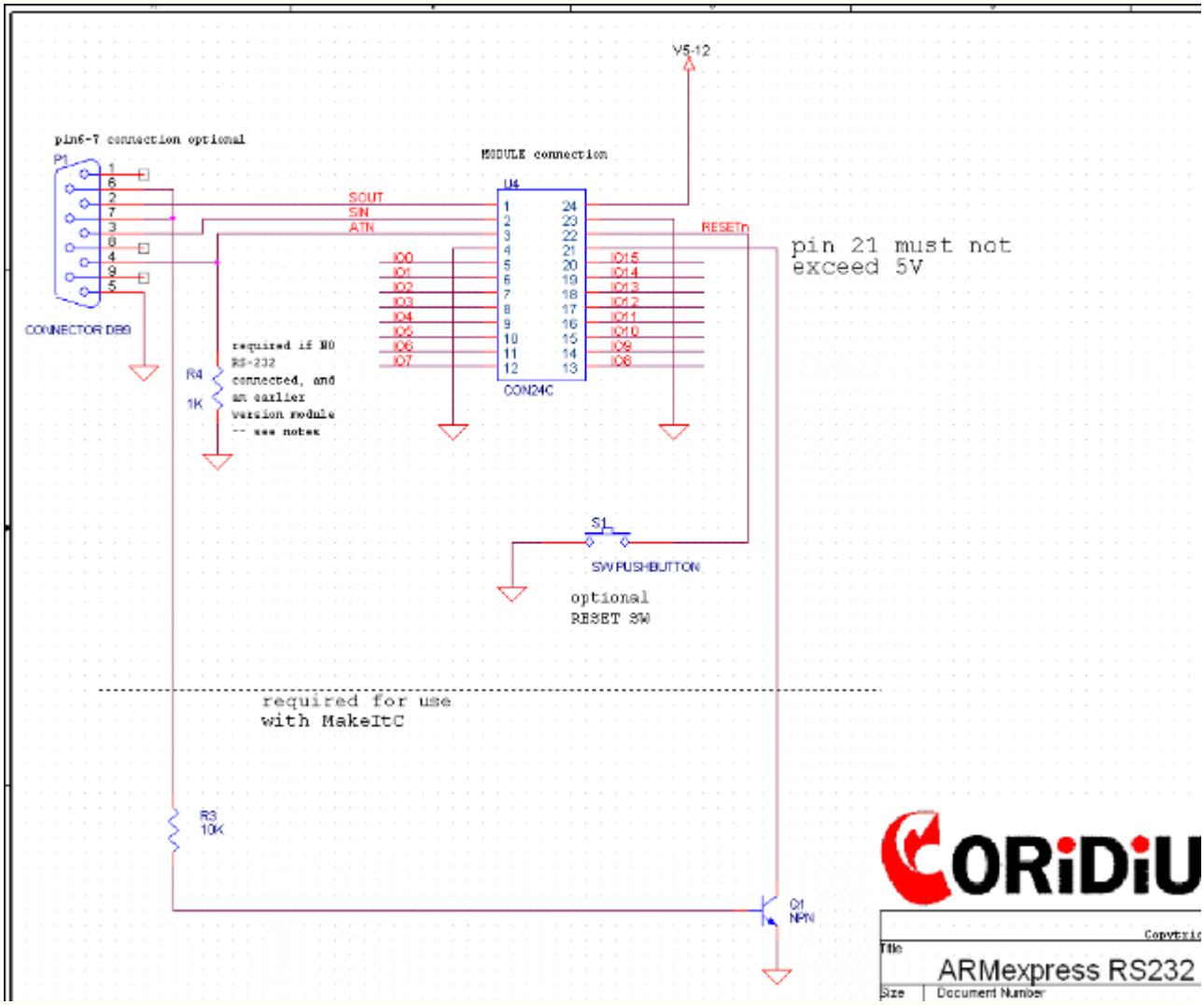
Toggle RXD will send a set of characters out to the UART0 of the ARM (it will be visible even on a simple meter)

force boot will drive Cn low, and can be used to manually put the part into ISP mode. This is done by being in Char Mode (send 1 character at a time, rather than buffereing up a line), toggle reset, send ?, send Synchronized. The ARM will echo Synchronized back.

For a Coridium USB dongle these are the expected states of the pins unless you changed the **programming of the FT232RL** . On other hardware if the states of pins does not agree with what is expected, check your wiring. If you see no activity at all, then you may have chose the wrong COM port, or don't have the FTDI driver installed.

ARMexpress Suggested RS232 connection

For a finer image see ARMexpRS.pdf in your install directory (C:\Program Files\Coridium\Schematics).



Pin 21

On most Parallax boards this line is connected to a regulated 5V supply.

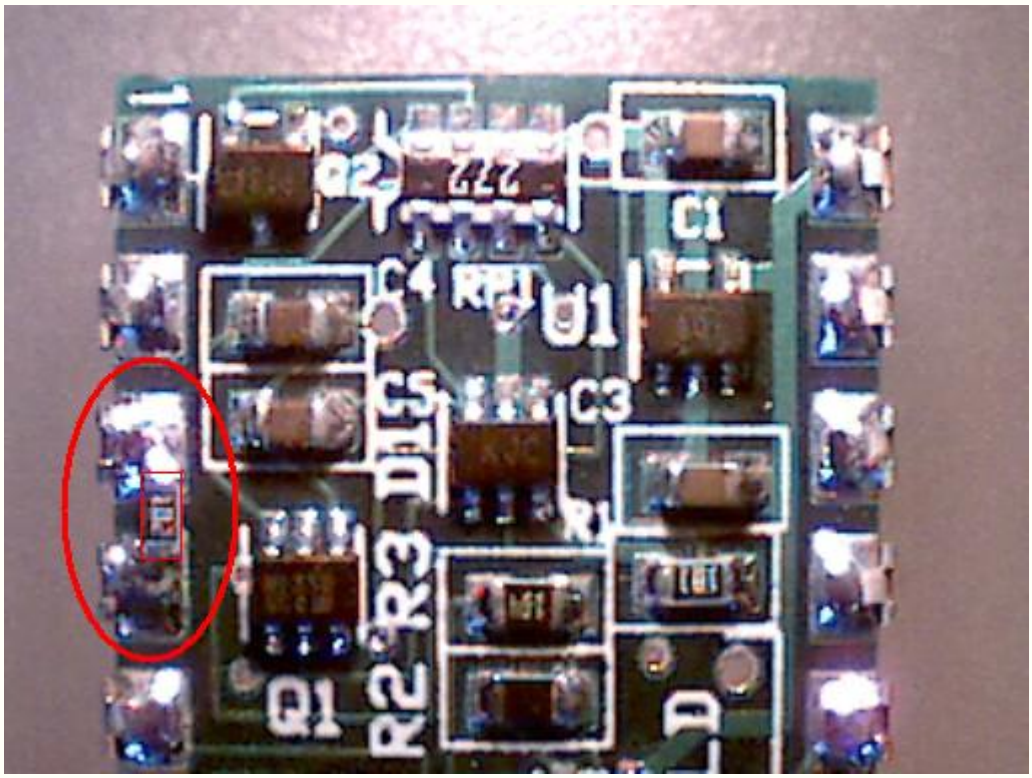
Do not connect a power source greater than 5V directly to pin 21.

When not connected this pin is pulled up to 3.3V by RP1 on the module.

When using MakeItC, this line is pulled low to download a C program, which can be done automatically by connecting to an NPN transistor with the RTS line on the serial port.

Pin 3

On later revision ARMexpress and ARMexpress LITE a 1K pulldown has been added on the module between pins 3 and 4 (as pictured below. If your unit does not have this, then a 1K pulldown resistor is required, when there is no signal on pin 3.


R2
C1
U1
C3
R3
Q1
R2

General Interfacing

Both the ARMexpress and the ARMmite can be directly connected to 5V TTL devices. The output voltage for these ARM devices ranges from 0.4V to 2.9V when driving upto 4mA of current. Most TTL devices will recognize these as valid logic levels (normally defined to be 0.8 and 2.0V)

Inputs

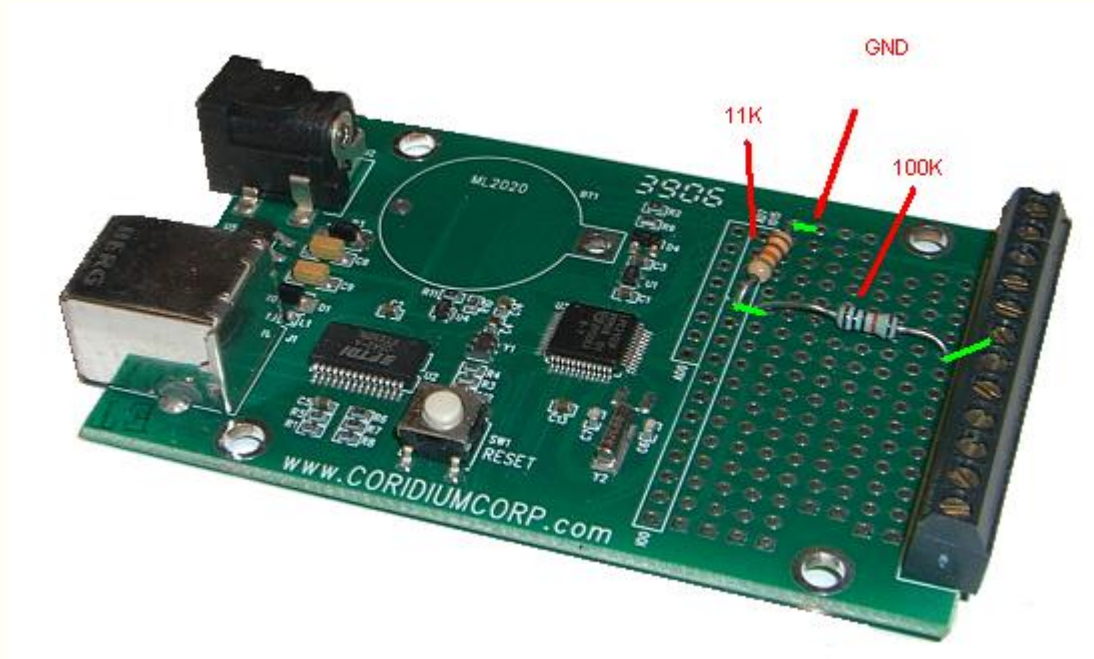
The ARMexpress and ARMmite may also be directly connected to 5V TTL outputs. If they are TTL compatable the voltage levels of the TTL output would normally be (0.4 and 3.4V), though they may go higher. The inputs for these ARM devices are 5V compatable.

Tieing to Supply lines

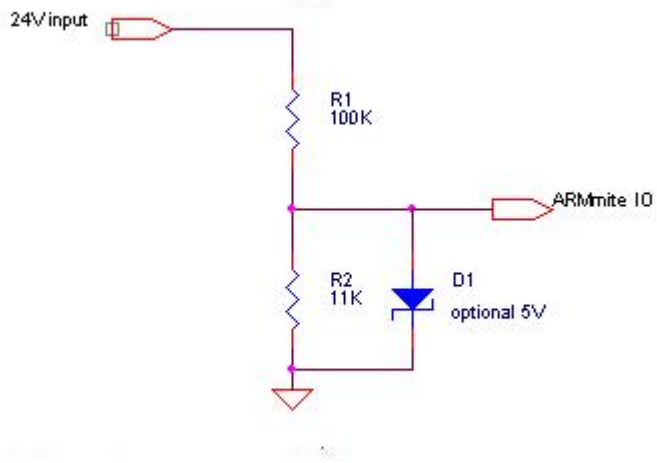
The ARMexpress and ARMmite inputs may be connected directly to a GND pin, but if connecting to a fixed high level, then it may be connected to a 5V supply line with a 1K or greater resistor. This is the same recommendation for any TTL compatable device. The reason being is that the 5V supply may exceed the 5V at times.

Interfacing to higher voltages

A resistor divider may be used to connect the ARMexpress and ARMmite to voltages that go higher than 5V. The picture below shows a connection appropriate for a 24V signal. A 100K resistor is connected from the input to IO(11) and then an 11K resistor connects IO(11) to GND. This will divide that 24V input to vary between 0 and 2.4V.



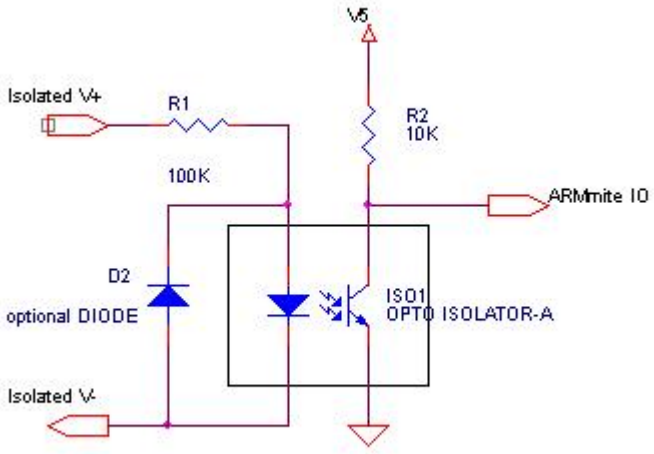
This resistor divider divides the 24V by 10 and also limits the current if that 24V goes higher. The circuit below shows schematically the connection that was made.



The resistors can be varied to handle different voltages. If the voltage to be sensed is susceptible to large spikes a 5V Zener diode can be connected in parallel with R2 to further protect the ARMmite IO.

Opto-Isolator

Another way to sense large voltages and to isolate the ARMmite from those voltages is to use an opto-isolator. These devices consist of an LED and a photo-transistor in a single package. They can provide isolation of 1000s of Volts. Below is a sample circuit. The D2 optional diode should be used if the isolated voltage to be sensed is an AC voltage. The value of R1 should be chosen depending on the Opto-isolator spec, with the current through the opto-isolator diode typically being 10 mA.



Serial Configuration

Though we recommend using TclTerm to talk to Coridium ARM products, here are settings for other terminal programs.

Any program on the PC that can communicate with a serial port can send or receive data to the ARM. This would include MSCOMM and Visual BASIC. Also various C's including GCC. Other options include Perl or Tcl scripts.

However these programs must be able to control the DTR and RTS lines under user control. If they cannot refer to the next **section**. Programs that cannot include Matlab, Hyperterm and Teraterm.

The TclTerm.tcl is the source for a Tcl program that operates as a terminal emulator for the ARMexpress family. You can use it if you have access to any of the GPL Tcl interpreters, or a compiled version is available on the Coridium Support page. The sources are also at the ARMexpress Yahoo Groups Files Section where you will also find a sample C program (writen for MinGW) that will also communicate with the ARMexpress family.

Baudrate

19.2 kbaud, 8 bit, No Parity, 1 stop bit. These settings are controlled by TclTerm, any settings in the Device Manager are IGNORRED.

End of Line

expects a LF (line feed),

CR is currently ignored.

Voltage Levels

/SOUT, /SIN and ATN (pins 1,2,3) will accept either TTL or RS-232 levels. ATN when high resets the ARMexpress, and ATN should not be allowed to float. It should either be connected directly to DTR, or some TTL signal that is LOW or Ground. The /SOUT driver relies on either /SIN or ATN being low to generate the low going voltage. This allows for full-duplex serial operation.

When TclTerm appears to be deaf

There are cases where the USB driver and TclTerm get out of sync. This includes when the board is disconnected from the USB port, and sometimes when the serial configuration is changed. In these cases it may be necessary to exit TclTerm and then restart it.

TclTerm configuration settings

The configuration of TclTerm is saved in a file TclTerm.ini. It is written when either it does not exist (when first installed) or when the configuration is changed by the user. This file is a Tcl source which may be edited by the user. If it becomes corrupt, delete the file and the default configuration will be restored.

USB use with MatLab, Hyperterm, TeraTerm

General Info

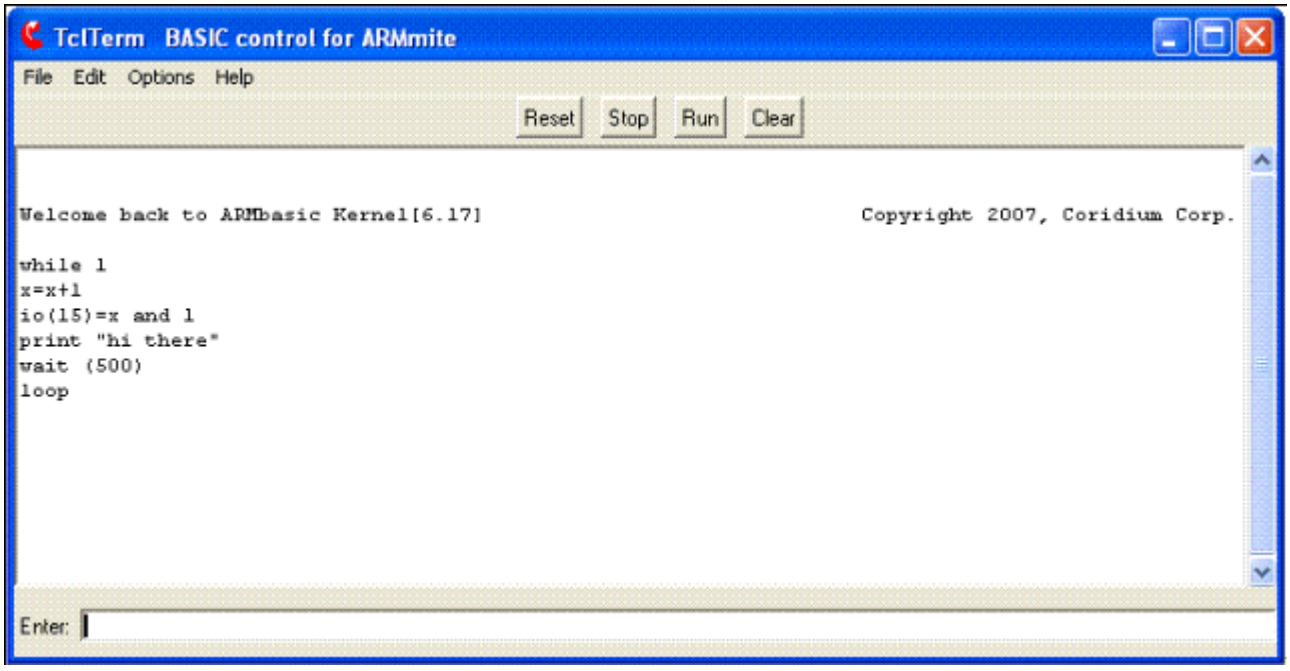
The ARMmite and ARMexpress use the DTR and RTS serial control lines to control programming and reset for the device. The state chosen allows the ARMmite/express to run and be reset by the push button while idle (ie. no serial program running).

PC side programs

Programs on the PC such as Tcl, MSCOMM and GCC allow the control lines to be controlled by the user. But some pre-compiled programs do not allow this control, such as HyperTerminal, TeraTerm, and MatLab. This page describes the steps to allow these programs to operate. For now this process must be done manually.

Useful debugging tool

Before starting its useful to load a program into the ARMmite/express that will flash the LED and also continuously send some data out the serial port. Here is one that works well...



Download the latest TclTerm

In order to be able to communicate with the ARMmite/express after the control lines have been changed, make sure you are running the latest TclTerm. Versions 1.6 and later have this support.

http://www.coridiumcorp.com/files/TclTerm.zip

Next, the driver must be changed for the USB serial device. The FTDI D2XX driver must be used. Download it from the FTDI website.

http://www.ftdichip.com/Drivers/D2XX.htm

Choose the proper version for your operating system, and download and install the driver. The installation executable may be used, and there are instructions in the Installation Guides on that page.

Configuration Utility

Next the settings of the serial control lines need to be changed, this requires the MProg utility from FTDI. Download and install this program.

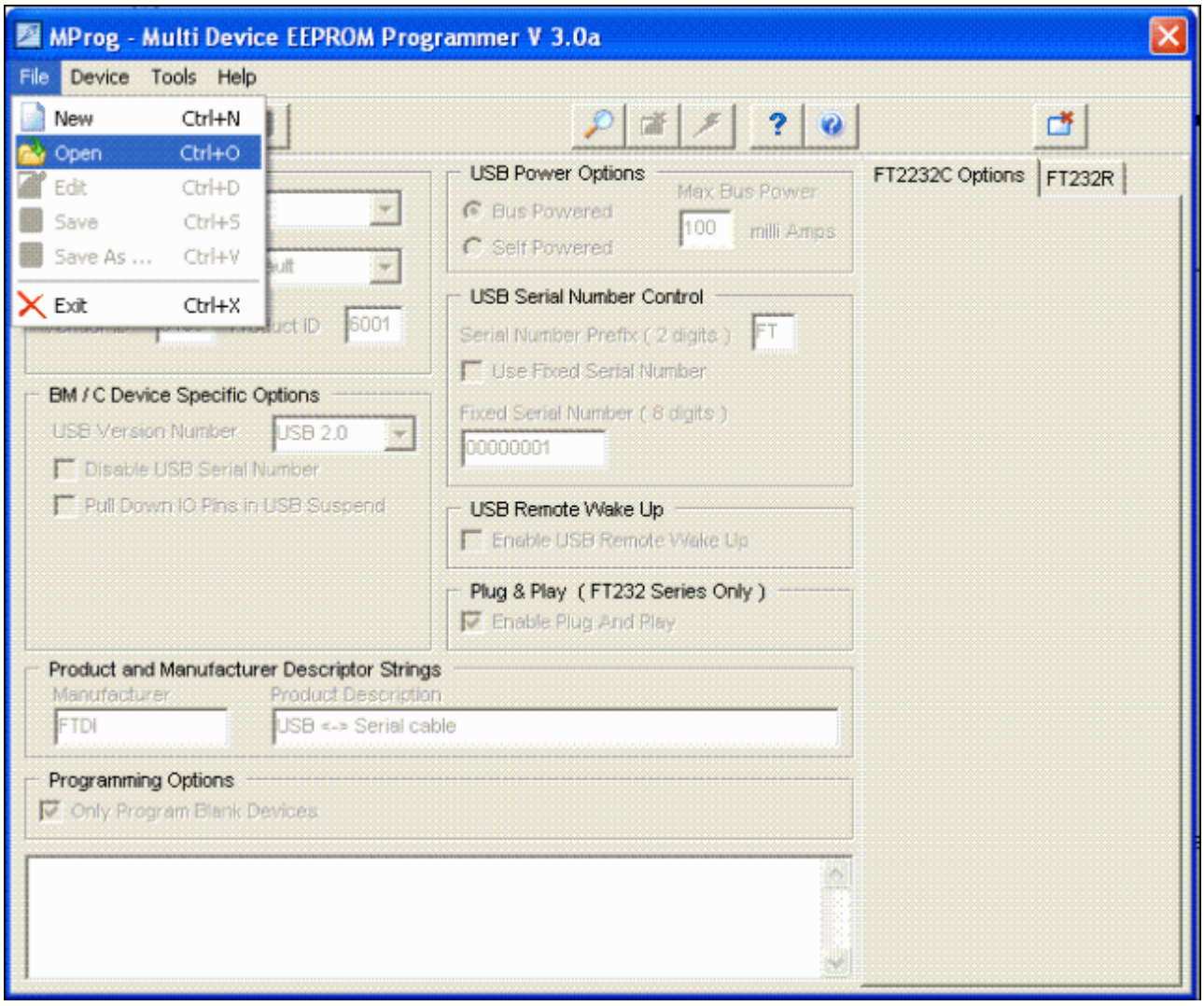
http://www.ftdichip.com/Resources/Utilities/MProg3.0_Setup.exe

Next download the data files for configuration of the ARMmite or ARMexpress eval PCBs. Unzip these files and store in a convenient directory (such as C:/Program Files/MProg 3.0a/Templates)

http://www.coridiumcorp.com/files/USBconfig.zip

Setup ARMmite/ARMexpress for MatLab, HyperTerminal, or TeraTerm

Once a BASIC or C program has been loaded, to configure the board to communicate with these other PC serial programs. Run the MProg utility. Load the serial File version in. And then reprogram the FTDI chip. **ONLY have 1 ARMmite or ARMexpress plugged in at time when you perform this operation.**



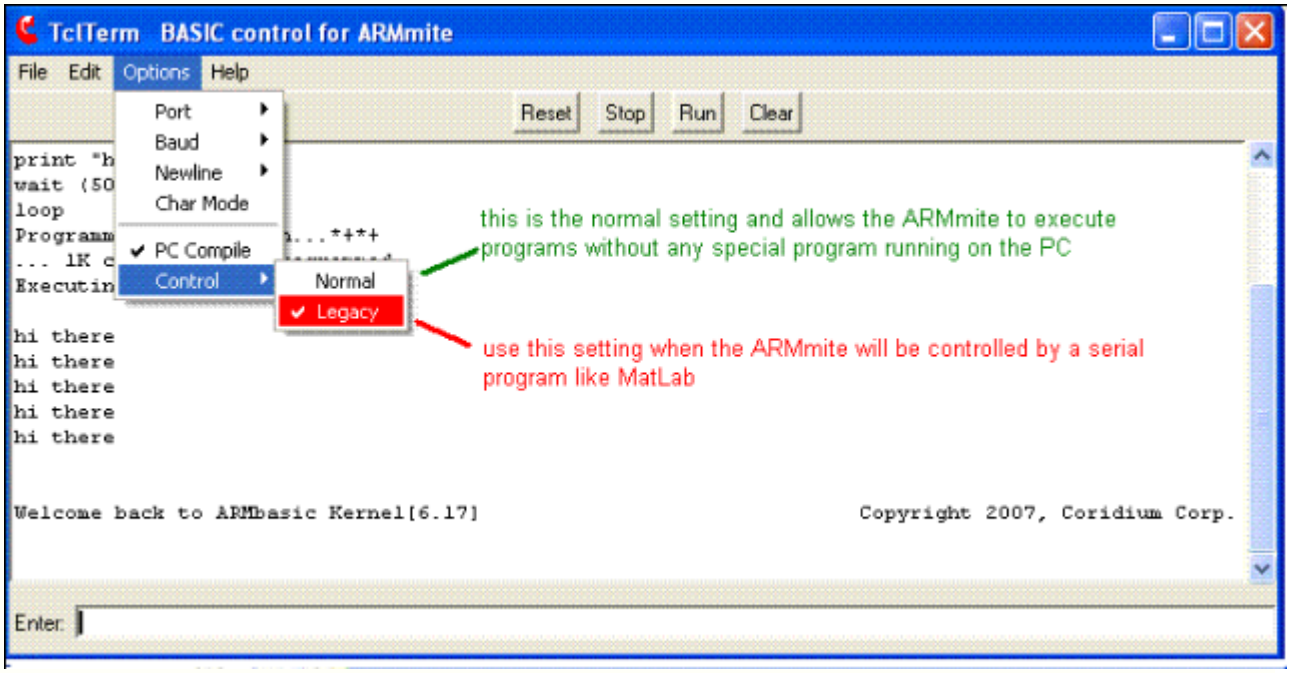
Exit this program and close any serial programs such as TclTerm. For this change to take effect, the ARMmite/express must be disconnected from the PC and reconnected.

Now the ARMmite/express will be idle until the MatLab serial port is open, Hyperterminal, or TeraTerm is run. Then after those programs are run, to start your BASIC or C program press the RESET pushbutton on the ARMmite/express.

Change the TclTerm settings for the reconfigured ARMmite/ARMexpress

In order to be able to change the BASIC program, you will still need to use TclTerm, but it will have to be

configured to use the new control line configuration (DTR and RTS inverted).



To return to the original settings. Run the MProg utility. Load the usbMITEnormal.ept or usbEXPnormal.ept file version in. And then reprogram the FTDI chip. Exit MProg and any serial programs, disconnect and reconnect ARMmite/express. And set the TclTerm Control option back to **Normal**.

Power

USB Power
The USB specification allows for up to 500 mA at 5V to be supplied to external devices. In many cases this is limited to 100 mA by the manufacturer of the PC or hub.

ARMexpress and its eval PCB uses approximately 50 mA when running and 10 mA when idle. So it can be powered from the USB port for programming, without the need for the alternate power supply.

Once the programming is completed, the ARMexpress may be run without a connection to a PC. In this case an alternate power supply has been provided in the evaluation kit. This supply will generate an un-regulated 6-8V which is connected to pin 24. Onboard the ARMexpress this will be regulated to 3.3V and 1.8V for use by the ARM CPU.

Initial Power on conditions

On power up all pins are tri-stated on the ARMexpress.

Restarting the program

If the user has programmed the ARMexpress, that program will be started when the power is applied, or restarted when RESET is asserted either low on the open-collector pin 22, or positive true on the ATN pin.

If the user program ends by getting to the last statement of the program or executing an END instruction, the ARMexpress will power down and await either input on the debug serial port, or a RESET.

Break operation or STOP
If the user code is running, it can be stopped by a RESET condition. This will normally restart the user code, but there is a short window (500 msec) where the ARMexpress will wait to see if there is input on the serial debug port. If the character received on the serial port is ESCAPE (27) or CTL-C (3) then the user program is prevented from running and the ARMexpress is ready to be reprogrammed. Or the user can restart the program by typing RUN or using the RUN button in TclTerm.

Smart Power

The USB evaluation board can be powered from either the USB, an external supply or BOTH. Power from the USB is controlled such that it is turned on by the USB controller. Power to the ARMexpress can also come from the external power supply and these are controlled to allow both USB and the power supply to be connected to the device at the same time.

The power connector is a 2.5mm, which is compatable with the Cui PP-002B part.

Parallax STAMP compatability

The Parallax STAMP products operate from a 5V supply. This can come from an unregulated input on pin 24, or from a regulated 5V supply on pin 21. The ARMexpress is backward compatable with both these connections, but for new designs it is recommended that power be supplied on pin 24. The voltage required is 4.5V or greater on pin 24, or 5V on pin 21. Also for future expansion, pin 21 should not be connected for new designs. The maximum voltage that may be applied to either pin 24 is 16V, but this is not a recommended continuous voltage, as it will cause extra heat to be generated by the ARMexpress onboard voltage regulators. For this reason the recommended maximum is 9V. When using an unregulated supply not supplied by Coridium, care should be excercised, as the current draw of the ARMexpress is low and the voltage will often be much higher than the rated voltage. The user should ensure that this voltage does not exceed the limit of 16V.

Timing

The oscillator
The ARMexpress uses a ceramic resonator for the timing element. It is accurate for 1%. It is used for timing of operations of SERIN, SEROUT, OWIN, OWOUT, PULSEIN, PULSEOUT, and COUNT.

Other operations such as I2CIN, I2COUT, SPIIN, SPIOUT, SHIFTIN, SHIFTOUT, PWM and FREQOUT are "bit-banged" loops that are calibrated to the speed of the CPU.

Interupts

The serial port for control and the timer are the only interupts currently used by the ARMexpress. The service routines for these actions have been minimized so that the user program is only interupted for TBD microseconds.

Operations that require accurate timing will disable the interupts during that critical period. These operations include OWIN, OWOUT, SERIN and SEROUT. Other operations that would be negatively impacted by an interupt also disable the interup for a period of time. Those include PULSIN, PULSOUT, PWM, RCTIME and FREQOUT.

Interupts and User code
When the ARMexpress receives serial input it will interput to copy data into its buffer. This will cause a small delay in the users program. In most cases this is not noticedable, but may be where user is timing with TIMER.

User code can cause the serial port to be deaf when running long operations such as FREQOUT or PWM. In normal operation this should not be a problem.

SPI,Microwire

The **Serial Peripheral Interface Bus** or **SPI** bus is a very loose standard for controlling almost any digital electronics that accepts a clocked serial stream of bits. A nearly identical standard called "**Microwire**" is a restricted subset of SPI.

SPI is cheap, in that it does not take up much space on an **integrated circuit**, and effectively multiplies the pins, the expensive part of the IC. It can also be implemented in software with a few standard IO pins of a microcontroller.

Many real digital systems have peripherals that need to exist, but need not be fast. The advantage of a **serial bus** is that it minimizes the number of conductors, pins, and the size of the package of an integrated circuit. This reduces the cost of making, assembling and testing the electronics.

A serial peripheral bus is the most flexible choice when many different types of serial peripherals must be present, and there is a single controller. It operates in full duplex (sending and receiving at the same time), making it an excellent choice for some data transmission systems.

In operation, there is a **clock**, a "data in", a "data out", and a "chip select" for each integrated circuit that is to be controlled. Almost any serial digital device can be controlled with this combination of signals.

SPI signals are named as follows:

- SCLK - serial clock
- MISO - master input, slave output
- MOSI - master output, slave input
- CS - chip select (optional, usually inverted polarity)

Most often, data goes into an SPI peripheral when the clock goes low, and comes out when the clock goes high. Usually, a peripheral is selected when chip select is low. Most devices have outputs that become high **impedance** (switched-off) when the device is not selected. This arrangement permits several devices to talk to a single input. Clock speeds range from several thousand clocks per second (usually for software-based implementations), to several million per second.

Most SPI implementations clock data out of the device as data is clocked in. Some devices use that trait to implement an efficient, high-speed full-duplex data stream for applications such as digital audio, digital signal processing, or full-duplex telecommunications channels.

On many devices, the "clocked-out" data is the data last used to program the device. Read-back is a helpful built-in-self-test, often used for high-reliability systems such as avionics or medical systems.

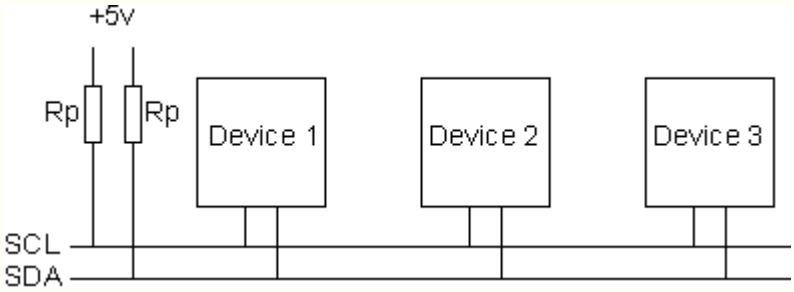
In practice, many devices have exceptions. Some read data as the clock goes up (**leading edge**), others read as it goes down (**falling edge**). Writing is almost always on clock movement that goes the opposite direction of reading. Some devices have two clocks, one to "capture" or "display" data, and another to clock it into the device. In practice, many of these "capture clocks" can be run from the chip select. Chip selects can be either selected high, or selected low. Many devices are designed to be daisy-chained into long chains of identical devices.

SPI looks at first like a non-standard. However, many programmers that develop **embedded systems** have a software module somewhere in their past that drives such a bus from a few general-purpose I/O pins, often with the ability to run different clock polarities, select polarities and clock edges for different devices.

The interface is also easy to implement for bench test equipment. For example, the classic way to implement an SPI interface from a personal computer to custom electronics is via a custom cable to the PC's parallel printer port. The parallel port generates and reads standard **TTL** logic voltages; +5V is high, ground is low. A number of helpful people have developed drivers to give access to this port in the most restrictive operating systems, such as Windows NT (see below), from the least likely environments, such as Visual Basic.

Using the I2C Bus

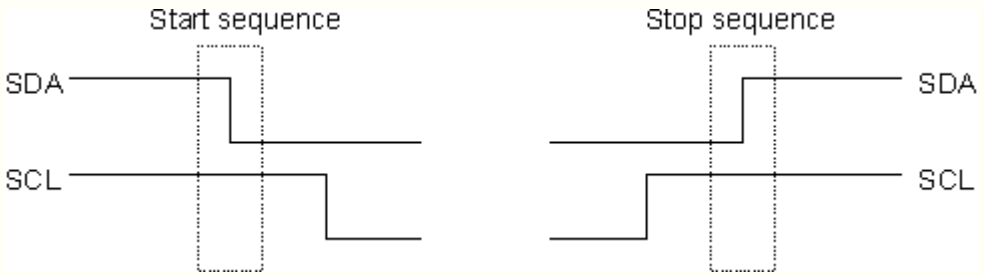
The physical I2C bus
This is just two wires, called SCL and SDA. SCL is the clock line. It is used to synchronize all data transfers over the I2C bus. SDA is the data line. The SCL & SDA lines are connected to all devices on the I2C bus. There needs to be a third wire which is just the ground or 0 volts. There may also be a 5volt wire is power is being distributed to the devices. Both SCL and SDA lines are "open drain" drivers. What this means is that the chip can drive its output low, but it cannot drive it high. For the line to be able to go high you must provide pull-up resistors to the 5v supply. There should be a resistor from the SCL line to the 5v line and another from the SDA line to the 5v line. You only need one set of pull-up resistors for the whole I2C bus, not for each device, as illustrated below:



The value of the resistors should be from 1.8K (1800 ohms) to 4.7k (4700 ohms). It depends on the length of the I2C bus, the longer the bus, the smaller value should be used. If the value is too large, the rise time of the signals will be too slow and the bus may not work properly. If the resistors are missing, the SCL and SDA lines will always be low - nearly 0 volts - and the I2C bus will not work.

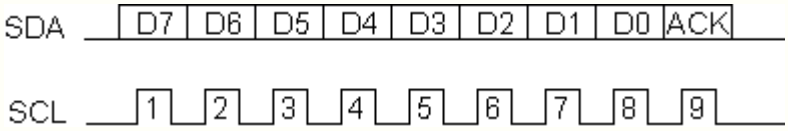
Masters and Slaves
The devices on the I2C bus are either masters or slaves. The ARMexpress as a master is always the device that drives the SCL clock line. The slaves are the devices that respond to the master. A slave cannot initiate a transfer over the I2C bus, only a master can do that. There can be, and usually are, multiple slaves on the I2C bus, however there is normally only one master. ARMexpress does not support multiple masters. Slaves will never initiate a transfer. Both master and slave can transfer data over the I2C bus, but that transfer is always controlled by the master.

The I2C Physical Protocol
When the ARMexpress wishes to talk to a slave it begins by issuing a start sequence on the I2C bus. A start sequence is one of two special sequences defined for the I2C bus, the other being the stop sequence. The start sequence and stop sequence are special in that these are the only places where the SDA (data line) is allowed to change while the SCL (clock line) is high. When data is being transferred, SDA must remain stable and not change whilst SCL is high. The start and stop sequences mark the beginning and end of a transaction with the slave device.



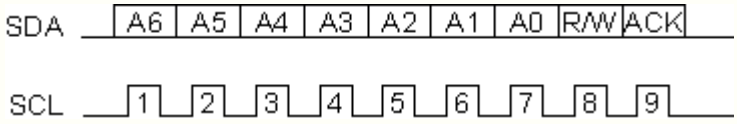
Data is transferred in sequences of 8 bits. The bits are placed on the SDA line starting with the MSB (Most Significant Bit). The SCL line is then pulsed high, then low. Remember that the chip cannot really drive the line high, it simply "lets go" of it and the resistor actually pulls it high. For every 8 bits transferred, the device receiving the data sends back an acknowledge bit, so there are actually 9 SCL clock pulses to transfer each 8 bit byte of data. If the receiving device sends back a low ACK bit, then it has received the data and is ready to accept another byte. If it sends back a high then it is indicating it cannot accept any further data and the

master should terminate the transfer by sending a stop sequence.



How fast?
ARMexpress runs in Fast mode at approximately 380 KHz.

I2C Device Addressing
All I2C addresses are either 7 bits or 10 bits. The use of 10 bit addresses is rare and is not covered here. All of our modules and the common chips you will use will have 7 bit addresses. This means that you can have up to 128 devices on the I2C bus, since a 7bit number can be from 0 to 127. When sending out the 7 bit address, we still always send 8 bits. The extra bit is used to inform the slave if the master is writing to it or reading from it. If the bit is zero are master is writing to the slave. If the bit is 1 the master is reading from the slave. The 7 bit address is placed in the upper 7 bits of the byte and the Read/Write (R/W) bit is in the LSB (Least Significant Bit).



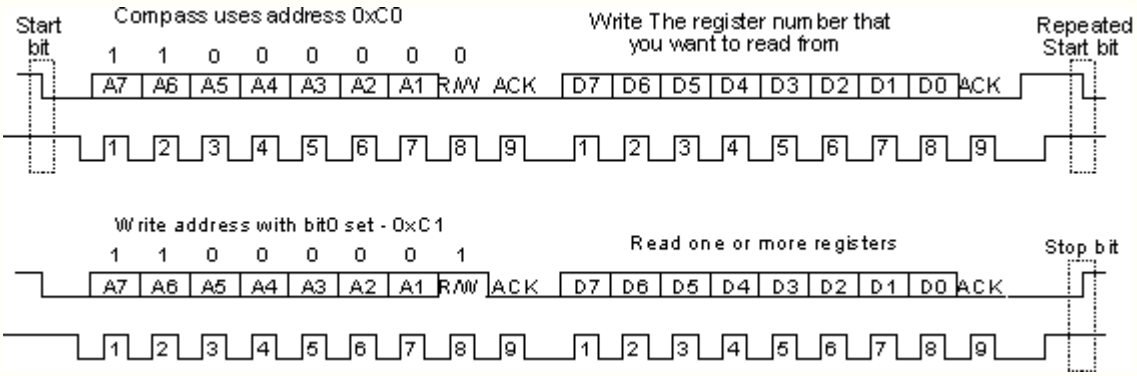
The placement of the 7 bit address in the upper 7 bits of the byte is a source of confusion for the newcomer. It means that to write to address 21, you must actually send out 42 which is 21 moved over by 1 bit. It is probably easier to think of the I2C bus addresses as 8 bit addresses, with even addresses as write only, and the odd addresses as the read address for the same device.

The I2C Software Protocol
The first thing that will happen is that the master will send out a start sequence. This will alert all the slave devices on the bus that a transaction is starting and they should listen in incase it is for them. Next the master will send out the device address. The slave that matches this address will continue with the transaction, any others will ignore the rest of this transaction and wait for the next. Having addressed the slave device the master must now send out the internal location or register number inside the slave that it wishes to write to or read from. This number is obviously dependant on what the slave actually is and how many internal registers it has. Some very simple devices do not have any, but most do. Having sent the I2C address and the internal register address the master can now send the data byte (or bytes, it doesn't have to be just one). The master can continue to send data bytes to the slave and these will normally be placed in the following registers because the slave will automatically increment the internal register address after each byte. When the master has finished writing all data to the slave, it sends a stop sequence which completes the transaction. So to write to a slave device:
1. Send a start sequence
2. Send the I2C address of the slave with the R/W bit low (even address)
3. Send the internal register number you want to write to
4. Send the data byte
5. [Optionally, send any further data bytes]
6. Send the stop sequence.

Reading from the Slave
This is a little more complicated - but not too much more. Before reading data from the slave device, you must tell it which of its internal addresses you want to read. So a read of the slave actually starts off by writing to it. This is the same as when you want to write to it: You send the start sequence, the I2C address of the slave with the R/W bit low (even address) and the internal register number you want to write to. Now you send another start sequence (sometimes called a restart) and the I2C address again - this time with the read bit set. You then read as many data bytes as you wish and terminate the transaction with a stop sequence. So to read the compass bearing as a byte from the CMPS03 module:
1. Send a start sequence
2. Send the I2C address of the slave with the R/W bit low (even address)
3. Send the internal register number you want to read from.
4. Send a start sequence again (repeated start)

2. Send the I2C address of the slave with the R/W bit high (odd address)
6. Read data byte from the slave device. (may be repeated depending on the slave capabilities)
7. Send the stop sequence.

The bit sequence will look like this:



Wait a moment
The ARMexpress does not support slaves that use clock stretching. The result is that erroneous data is read from the slave. Beware! Luckily this function is relatively rare these days.

Example Master Code

```
DIM A$(10)
I2CIN 1,$30,$10, [STR A$ \10]    ' read 10 bytes from slave $30 register $10
connected on pins 1,2

I2CIN 5,$40,$20, [X]             ' read a single byte from slave $40 register
$20 on pins 5,6

FOR I=0 TO 9
  A$(I) = $30 + I
NEXT I

I2COUT 1,$30,$10, [STR A$ \10]   ' send 10 bytes toslave $30 register $10
connected on pins 1,2

X=$55
I2COUT 5,$40,$20,[X]             ' send a single byte to slave $40 register $20
on pins 5,6

I2COUT 5,$50,$20,[$AA]           ' send $AA to slave $50 register $20 on pins
5,6
```

Easy isn't it?

The definitive specs on the I2C bus can be found on the Philips website. Its currently **here** but if its moved you'll find it easily be googleing on "i2c bus specification".

ARM Peripheral Use

The ARM peripheral bus

Timer0 free running micro-second counter (TIMER command)
Timer1 used for ON TIMER interrupt
Timer1, Timer2 and Timer3 used for HWPWM on ARMmite or ARMexpress LITE
Uart0 UART
Uart1 Not Used
PWM used when HWPWM is engaged
I2C Not Used
SPI reserved
RTC used for time-keeping

```
In Idle just the CPU clock stops and any interrupt will wake it.
```

Tables

Tables

ASCII Character Codes

ASCII Character Codes

ARMbasic uses the standard "ASCII extended" character set. The compiler uses the character set values 32 to 126 which corresponds to SPACE through TILDA.

Characters outside this range may have a special meaning and are interpreted by the terminal emulation program that is controlling the ARMexpress. Those would include BACKSPACE, TAB, CR and LF. These characters cause changes in the stream of characters going to or from the ARMexpress module. These characters may be interpreted differently on a PC vs. a Mac.

Two codes XON and XOFF are used for flow control. When a large **ARMbasic** program file is sent to the ARMexpress module, the module may require a delay when writing code into Flash memory. During these writes of code to Flash, an XOFF character will be sent to the PC that indicates that the PC should pause sending data. After the block is written (about 0.4 second) an XON will be sent to resume communication.

However when using SERIN or SEROUT, there is no special interpretation of characters, so all codes 0 to 255 may be sent without any change.

Dec	Hex	Meaning	Dec	Hex	Meaning
000	000	NUL (Null char.)	064	040	@ (AT symbol)
001	001	SOH (Start of Header)	065	041	A
002	002	STX (Start of Text)	066	042	B
003	003	ETX (End of Text)	067	043	C
004	004	EOT (End of Transmission)	068	044	D
005	005	ENQ (Enquiry)	069	045	E
006	006	ACK (Acknowledgment)	070	046	F
007	007	BEL (Bell)	071	047	G
008	008	BS (Backspace)	072	048	H
009	009	HT (Horizontal Tab)	073	049	I
010	00A	LF (Line Feed)	074	04A	J
011	00B	VT (Vertical Tab)	075	04B	K
012	00C	FF (Form Feed)	076	04C	L
013	00D	CR (Carriage Return)	077	04D	M
014	00E	SO (Shift Out)	078	04E	N
015	00F	SI (Shift In)	079	04F	O
016	010	DLE (Data Link Escape)	080	050	P
017	011	DC1 (XON)	081	051	Q
018	012	DC2 (Device Control 2)	082	052	R
019	013	DC3 (XOFF)	083	053	S
020	014	DC4 (Device Control 4)	084	054	T
021	015	NAK (Negative Ack)	085	055	U
022	016	SYN (Synchronous Idle)	086	056	V
023	017	ETB (End of Trans. Block)	087	057	W
024	018	CAN (Cancel)	088	058	X
025	019	EM (End of Medium)	089	059	Y
026	01A	SUB (Substitute)	090	05A	Z
027	01B	ESC (Escape)	091	05B	[(left bracket)
028	01C	FS (File Separator)	092	05C	\ (back slash)
029	01D	GS (Group Separator)	093	05D	] (rightbracket)
030	01E	RS (Request to Send)	094	05E	^ (caret)
031	01F	US (Unit Separator)	095	05F	_ (underscore)
032	020	SP (Space)	096	060	`
033	021	! (exclamation mark)	097	061	a
034	022	" (double quote)	098	062	b
035	023	# (number sign)	099	063	c
036	024	$ (dollar sign)	100	064	d

037	025	%	(percent)	101	065	e	
038	026	&	(ampersand)	102	066	f	
039	027	'	(single quote)	103	067	g	
040	028	(	(left parenthesis)	104	068	h	
041	029	)	(right parenthesis)	105	069	i	
042	02A	*	(asterisk)	106	06A	j	
043	02B	+	(plus)	107	06B	k	
044	02C	,	(comma)	108	06C	l	
045	02D	-	(minus or dash)	109	06D	m	
046	02E	.	(dot)	110	06E	n	
047	02F	/	(forward slash)	111	06F	o	
048	030	0		112	070	p	
049	031	1		113	071	q	
050	032	2		114	072	r	
051	033	3		115	073	s	
052	034	4		116	074	t	
053	035	5		117	075	u	
054	036	6		118	076	v	
055	037	7		119	077	w	
056	038	8		120	078	x	
057	039	9		121	079	y	
058	03A	:	(colon)	122	07A	z	
059	03B	;	(semi-colon)	123	07B	{	(left brace)
060	03C	<	(less than)	124	07C	|	(vertical bar)
061	03D	=	(equal sign)	125	07D	}	(right brace)
062	03E	>	(greater than)	126	07E	~	(tilde)
063	03F	?	(question mark)	127	07F	DEL	(delete)

Support

Support

- How to contact the developers
- How to report a bug
- Contributors

How to contact the developers

The GCC compiler is part of the GNU project.

MakeItC and ARMbasic are products of Coridium Corp.

- **www.coridiumcorp.com**

Tech Support monitors the following groups.

- **groups.yahoo.com/group/ARMexpress**
- **groups.yahoo.com/group/gnuarm**
- **groups.yahoo.com/group/lpc2000**

Coridium has done custom ports of ARMbasic to other platforms.

- **techsupport@coridiumcorp.com**

See also

- **Reporting a bug**

How to report a bug

Before reporting a bug, try to make sure it's a bug in Coridium code and not a bug in your own code. Try to write a small test that reproduces the problem you are encountering. Read any relevant documentation. If you show people that you have tried to solve your own problem, rather than immediately running for help, you will be more likely to find people willing to help you.

Be as specific as you can - "The FREQOUT runtime library function fails when it is called with a value of 1234" is much better than "It crashes".

The first place to go in the case you believe you've encountered a bug is **groups.yahoo.com/group/ARMexpress**

If you have isolated a compiler bug completely, and you have steps to reproduce it and a small piece of sample code, you can also file a bug report with tech support at **support @coridiumcorp.com**.

DO NOT file general "it doesn't work!" bug reports in the **groups.yahoo.com/group/ARMexpress** system. Only isolated, reproducible bugs should be posted there.

Contributors

MakeItC was developed by Coridium to support its own GCC efforts in the development of the ARMbasic compiler. MakeItC will continue to be expanded and refined as needs exist. Coridium does entertain suggestions for feature enhancements to MakeItC.

MakeItC is written in Tcl, a cross-platform language. Coridium will help others port MakeItC to other systems, but it is beyond our capabilities and expertise to port or support platforms other than Windows.

A number of utilities have been used to produce the ARMexpress system.

Freewrap is used to generate TclTerm from a Tcl/Tk script (source available from Coridium website).

Winarm is the basis for the gcc compiler.

A freeware grep from Tim Charron.