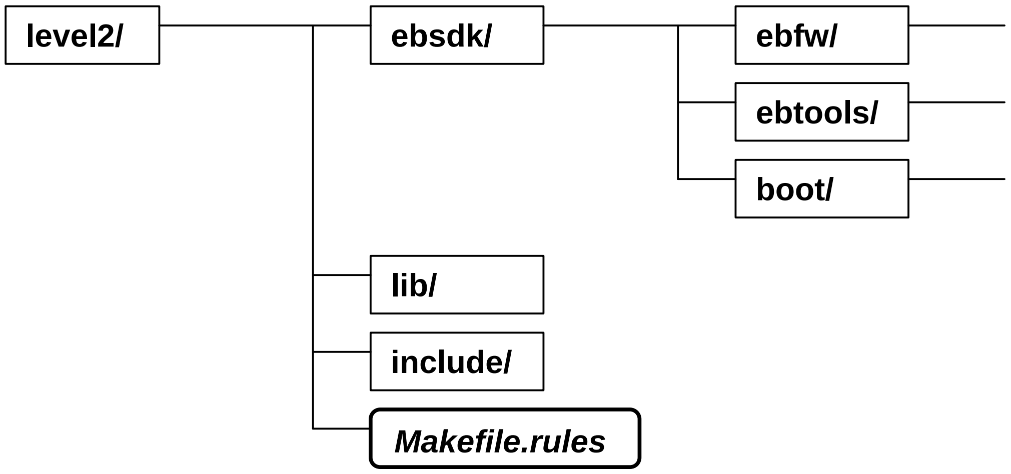
Compiling Programs for PC164 Board

System Setup

Directory Structure

The system administrator needs to install the Digital PC164 Software Development Kit (SDK) within the directory structure shown below:



The *ebsdk* directory contains the SDK files as unpacked from the tar file which the distribution comes in. The *lib* and *include* directories contain library and header files respectively. These must be named as shown unless the *Makefile.rules* file is edited. The choice of the root directory, level2 in the figure above, is completely arbitrary and must be inserted into the rules file as described below.

Adapting the Rules File

The rules file contains the default rules for building a downloadable application. It specifies the location of the SDK as well as the various directories to search for libraries, header files etc. The variables which will need to be edited for a given system configuration are:

- **TOPDIR** = root directory for the whole system directory structure
- **TFTPDIR** = directory to copy the executables to for TFTP transfer to the board
- **BOARDNAME** **=** IP name of the board (or it's IP address)

These are all found at the top of the rules file and must be altered to the correct locations. Users who are going to download programs to the board must also be given permission to write to the TFTP directory. Other variables might need tweaking for different compiler versions, executable locations etc.

User Setup

Standards

The new makefiles enforce several standards for compiling programs. The most noteworthy are the directory structure for each application, which is outlined below, and the use of the C++ compiler to compile all C program code as well as C++.

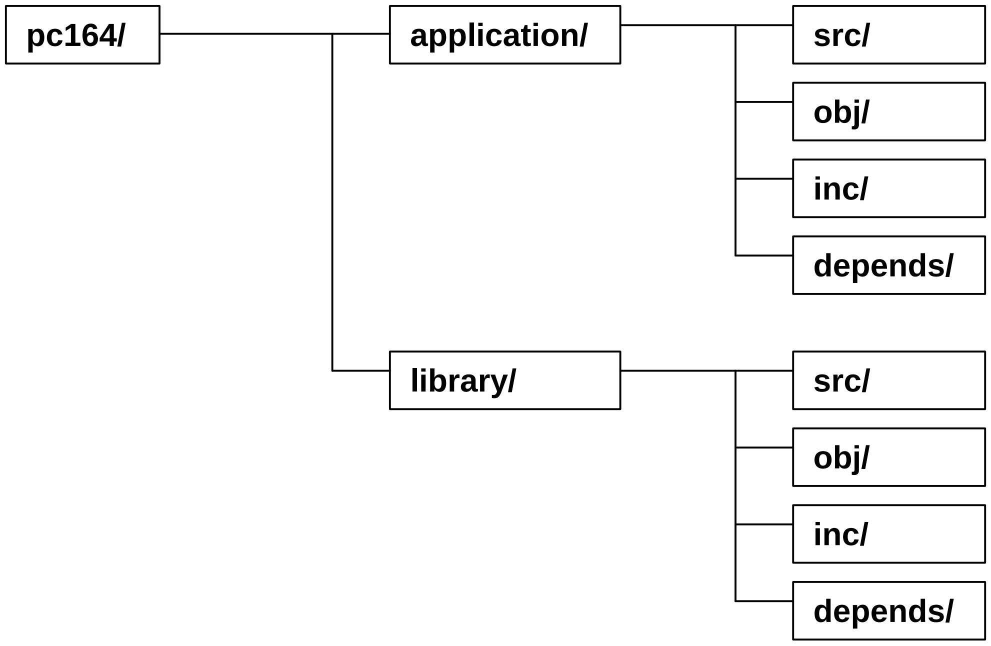
The choice to use the C++ compiler for C code is based on several factors:

- C++ compiler enforces strong prototyping and typecasting on programmers which allows errors to be detected by the compiler. The standard DEC C compiler will just assume functions return integers if no prototype is given which leads to run time errors if header files are omitted.
- When compiling code without compiler optimizations the C++ compiler produces significantly (approximately 30%) faster code that the C compiler.
- The standard D0 offline environment will be C++. By ensuring our C code is C++ compatible it will make integration into the offline environment for simulations much more straightforward.

There are now two types of applications which you can compile: a library or an executable. The library application option is meant for standard routines which will be linked into several executables. With this option the makefile will archive all the object files into a single library file. If you choose the executable option then the makefile will link the object files produced into an executable and then copy this to the correct system location for TFTP download to the board.

Directory Structure

To compile programs for downloading to the PC164 board you need to create the following directory structure:



The names of the *pc164*, *application* and *library* directories are chosen by the user, however the *src*, *obj*, *inc* and *depends* directories must be given those precise names. Only one *library* or *application* directory is needed per application or library the user wishes to compile.

The contents of the four directories for each application/library are the following:

src	- source code
obj	- compiled object code
inc	- header files
depends	- automatically generated file dependencies

In addition each application directory contains a "Makefile". This makefile must define the application directory, the type of the application (executable or library) and any user defined flags required to compile the program correctly.

Customizing the Makefile

The makefile for compiling the application contains several variables to allow the user to define the application's directory, name and type as well as providing a means to add extra flags to the various compilation stages. The variables the user *must* define are:

- **APP** = name of the application
- **APPTYPE**= type of the application. This must be either 'lib' or 'exe'
- **APPDIR** = base directory of the application containing the src, obj, etc. directories
- **SRCS** = list of source files separated by spaces e.g. "code1.c code2.c code3.c…"

As well as these required variables there are several optional ones which give extra compile flags to the various compiler stages.

- **APPLIBDIR** = extra library directories to search
- **APPLIBS** = extra libraries to link with
- **APPASFLAGS** = flags for the assembler
- **APPCXXFLAGS**= flags for the C++ compiler (also used to compile C code)
- **APPCPPFLAGS** = flags for the C pre-processor
- **APPDEBUG** = flags for the ladebug debugger

These optional flags are passed to the relevant commands without any formatting being done. Consequently they must contain valid compiler flags i.e. The APPLIBDIR variable must contain '-L' in front of the directory name e.g. –L/user/lib.

As well as defining the variables above the user must also ensure that their makefile includes the Makefile.rules file contained in the system directory for the PC164 board. This is achieved by adding the command:

```
include /level2/Makefile.rules
```

at the end of the application's makefile. The '*/level2*' part of the filename must be altered to the directory containing the *Makefile.rules* file on the system where the application is to be compiled.

Using the Makefile

The makefile implements several different rules to make compiling, debugging and tidying up easier. The primary use of the makefile is to compile only the parts of your application that have been altered and then to link these to produce a new executable which is then placed in the correct directory for TFTP transfer to the board. This is achieved with the command:

```
> gmake
```

As well as this several other commands are implemented. To help remove unwanted files created by the *emacs* editor there is the

```
> gmake purge
```

command which deletes all files ending with a tilde (~) in the application's root, *src* and *inc* directories. These files are backup files generated by the *emacs* editor. There are also the commands

```
> gmake clean          and          > gmake spotless
```

which both delete the application's object and executable files. The command "`gmake spotless`" also deletes the dependency files created in the *depends* directory. The final command implemented is

```
> gmake debug
```

which starts the debugger and attempts to connect it to the board. For this command to work the board should already have the code downloaded to it and be waiting for a connection from a *ladebug* client.

Example Makefile

Included below is an example of a makefile to compile the C++ test framework on *alfa-m.pa.msu.edu* at Michigan State.

```
#
# Makefile for C++ Framework for Level 2 Global Trigger
#
#                                           RWM 13/4/98
#

# Applications name
APP     = cpptest
APPTYPE = exe

# Application directory
APPDIR = .

# Source code files
SRCS = Tool.C ElectronTool.C frame.C \
       InputBuffer.C Electron.C Buffer.C

# Application specific flags/libraries :
#  APPLIBDIR   : library directories to search eg. -L/user/fred/lib
#  APPLIBS     : Libraries to include eg. -lfred
#  APPASFLAGS  : Flags for compiling assembly code
#  APPCXXFLAGS : Flags for compiling C/C++ code eg. -g, or -O4
#  APPCPPFLAGS : Flags for the C pre-processor eg. -I/user/fred/include
#  APPDEBUG    : Flags for the Ladebug debugger program
APPLIBDIR   =
APPLIBS     =
APPASFLAGS  =
APPCXXFLAGS = -O2
APPCPPFLAGS =
APPDEBUG    =

default:
	gmake $(APP)

include ../Makefile.rules
```