The generation of Bnldsp C object code modules and libraries for the LEIR Fesa BnldspXXX device classes.

E. Bracke (AB/RF-cs)

Table of Contents.

Introduction. 3
The Documentation. 4
The Tools. 4
Checking the memory map. 5
The driver simulator library. 6
The generic header file directories 7
The GEN_HEAD directory 7
The DSP / RTT_shared directories. 7
The generic linkable libraries directory. 7
The generic ‘Make’ source directory. 8
The ‘per Fesa Class’ C code directories. 8
The HEADERS subdirectory. 9
The XXXfuns, XXXrts directories. 9
The XXXprpL, XXXprtL directories 9
The XXXsrvs directory. 9
The FDI_INIT directory. 9
The EXCEL directory. 10
Note on the FESA / FEC management integration. 11
The ‘make recipe’. 13

Introduction.

The BNLDSP project subdirectory tree, in which is situated the DOCUMENTATION directory where you found the description you are currently reading, contains other interlinked directories with C code that are together required for the generation of all modules / libraries for making the LEIR RF Fesa devices work.

In all, we distinguish:

- Documentation in directories: DOCUMENTATION and bnldspDR,
- Tools in the directories: bnldspMM and bnldspLI,
- C source code (header files) in the project generic directories: GEN_HEAD, RTT_shared and DSP_shared
- Generic linkable libraries in directory: LIB
- Generic 'Make' source in directory: MAKE_SRC
- Finally the 'per Fesa class C code' directories: BnldspXX, where 'BnldspXX' refers to an approx. same named Fesa device class for which the C code is intended. Not all Fesa classes however have currently C code parts implemented, so, no BnldspXX directory tree for those.

A more detailed discussion of each of these directories (structure and contents / use) will follow in this note's 'The BNLDSP directory tree.' section.

The 'make recipe' section of this note goes into the detail of the order in which compilation should be executed such as to obtain a coherent set of object code modules and libraries, ready for integration in the Bnldsp Fesa classes.

The LEIR RF low level control system consists of a VME chassis with currently 3 (later may be 4) VME hardware modules, initially designed together with Brookhaven National Laboratory in the USA. These modules are described by a separate note ('DSP carrier board manual – board release 1.0' by: M. E. Angoletta, J. DeLong (BNL), F. Pedersen). The VME modules are hardware wise spoken rather similar, but different daughter cards on them cater for specialization and software for the on-board DSP and FPGA's can also be quite different.

This current note describes the implementation of the software that provides the remote control of these VME modules via the CERN standard frontend computer software standard architecture called 'Fesa'. In particular the C code modules that will be integrated as callable functions for the standard Fesa 'device instances'.

The Fesa devices operate on control aspects of the hardware. This means that there is generally no 'one Fesa device' to 'one hardware device' mapping, but a Fesa device can access the same functionality on either of the 3 (or 4) VME modules as required by the main control room operator.

Currently there are some 13 Fesa classes defined for the LEIR RF beamcontrol with each one from 1 (BnldspCTL) to 77 (BnldspTIM) individual devices instantiated.

This note is stored in the BNLDSP/DOCUMENTATION directory which can be found in the CERN CVS file repository, currently at:
http://isscvs.cern.ch/cgi-bin/viewcvs-all.cgi/?root=abrfcs
(and also available on Erik's home page at:
http://bracke.home.cern.ch/bracke/HTML/LEIR_Development/LEIR_Development.htm).

Note:
Also have a look at the readme.txt file in the BNLDSP root. It might have some interesting (last minute) info.

The Documentation.

In the `DOCUMENTATION` directory some extraneous information can be found:

- For the management of frontend hardware configuration via an Oracle database for the VME driver. (file: 'HardwareConfigurationManagement_modelReview.pdf'). This is a draft document made by Alain Gagnaire.
- Another file (file: 'AB front-end.pdf') describes, in the form of a PowerPoint presentation done by Nicholas de Metz-Noblat, the frontend computer software organization.
  Of particular interest is the chapter on the main file systems (page 17 and onwards). There is explained how to create the frontend specific startup file, transfer.ref ('`make transfer.ref`') and how to build a frontend specific data table ('`make new_dtab`'). Also have a look in the 'excerpt_m_n_dtab.txt' file, it shows which code (server / RT) is put where for frontend startup.
- The file: 'Starting User Programs on LynxOS, Linux and HP-UX computers.pdf', written by Alastair Bland, contains info on the syntax of file `transfer.ref` which one must know for properly defining a macro for the database that will generate an entry in `transfer.ref` for the starting-up of a Fesa class at boot time.
- A note describing the usage of the 'closely collaborating' 4 BNLDSP Fesa classes that form the OASIS interface for 'analogue', ocilloscope like, signal display in file: 'oasisQ.pdf'.
- As an adaptation to Alain Gagnaire's note, specifically intended for integrating the BNLDSP Fesa device classes into the LEIR front end computers (for auto-boot), I've written the document: 'fecConf.pdf'.
- Finally, this current note is stored here as file: 'genCCode.pdf'.

The second documentation directory is called: `bnldspDR`. This directory is a (partial) copy of the VME driver source code which can currently be found in its entirety on the paths:

`/ps/src/dsc/drivers/bnldspvme/VME-Bnldsp` (old, original code)
`/ps/src/dsc/drivers/VME-Bnldsp` (current, operational code)

It represents the C source of the BNLDSP VME board driver and is given here for information only. This was notably used for the creation of the `bnldspLI` tool (see chapter 'The driver simulator library.'.) and the driver access library header files in directory `GEN_HEAD` (see chapter 'The generic header file directories.' are also derived from them. Nowhere in the BNLDSP project the C code itself in this directory is used or even compiled.

The Tools.

Development for hardware progresses at the same time as development on the control software. With programmable hardware (DSP and FPGA's) on board, this means that there is quite a lot of software involved which we gather under the heading 'hardware' for the purpose of this note. This 'hardware-software' is written in a language close and (almost) compatible to the C language; the chosen language for the control software.

Therefore as interface between hardware and software development we have chosen to use the memory map (as seen on the VME bus) of the BNLDSP boards. A set of header (include) files describes this memory map as well as many bit patterns and other constants that must be known by both hardware and software developers.

To keep coherence over all parts of this project, we need to make sure that all software sees the same memory map and uses, if so required, the same value of the constants.

The first tool that helps to guarantee a coherent memory map all over the BNLDSP project is the `bnldspMM.exe` programme. This tool also creates the 'Named Region lookup table Initializer' files which are further discussed below.

Currently two Fesa device classes (BnldspTIM and BnldspCTL; discussed in chapter 'The 'per Fesa Class' C code directories.') can be simulated entirely in software, without the need of real BNLDSP hardware, either on the MS Windows desktop (MS Visual C++) or, when compiled with e.g. GNU's gcc, from the commandline on any UNIX(-like) system. The only requirement for this is the availability of a VME driver simulator (either our own driver simulator library linked in, or BNLDSP drivers loaded in 'Simulator' mode; the latter however, only on VME UNIX-like systems).

The second tool provided in the `BNLDSP` tree is thus the driver simulator library, either called: `bnldspLIB.lib` (MS Visual C++) or: `libbnldspLI.a` (UNIX systems).

Finally, remember that the LEIR Fesa devices are intended to run on a VME frontend computer that currently uses a PowerPC as CPU. The tools might therefore probably only run on such a platform; anyway they were only compiled / tested / used on the rflynx platform as the development machine.

Checking the memory map.

The `bnldspMM` directory is the root of the development tree for the `bnldspMM.exe` programme, the tool responsible for checking / printing of map and other log files. Here we find an `RTT_shared` and a `DSP_shared` directory in which will be deposited any 'fresh' set of the memory map definition files coming from the hardware developer.
Currently downloading a 'fresh' version of these files is done from the same named directories at the CERN dfs network, path (all on 1 line):
`\\cern.ch\dfs\Users\a\angolett\www\project-leir-dsp-bc\DSP_code\LEIRLLRF_PROJECT\LEIRfastloop\LEIR2007\Version 208\`
Also note that not all files from these remote directories are used in the software development environment; only download those required for local compilation.

Also to be found in the `bnldspMM` directory tree is a `HEADERS` directory in which is deposited:

- The current definition of the memory map, as used in the C code development, file: `bnldspMM.h`
- The current definition of the named region (see discussion of named regions below) map, as used in the C code development, file: `bnldspNR.h`
- The current definition of the lookup table / constants definition, file: `bnldspLU.h`
- A set of lookup table initializer files all with extension: `*.lui`

A 'named region' is a structure that carries parameter values (offset and blocksize) for the BNLDSP hardware driver that must be passed on by a user of the driver library when hardware access is required. The 'named region' structure also has a member for a short text label which can be used by logging / documenting such accesses. It is the 'name' of the parameter set.
Furthermore the structure carries some other info like the internally used data type and the number of array elements that must be accessed by the driver. Note that an 'array element' here is one of the internally defined 'data types' and the latter could well be declared as some sort of a structure (generally: an 'agglomerate' in C terminology), hence access to the hardware could be done quite elegantly and logged efficiently!

For a programme module the 'named regions' lookup table, declared in `bnldspNR.h`, is just a another, be it rather special, lookup table that requires an initializer file of its own. The lookup table is organized as an array of lookup tables, one element 'per' BNLDSP board. It is one of the tasks of the `bnldspMM.exe` programme to create the initializer files for each of the 'named regions' lookup tables. Currently the tool creates 4 named region lookup table initializer files; we find them in the `bnldspMM` directory as `bnldspX0.nri` files, '`X0`' designating the BNLDSP VME board type.

Initial versions of these files are always required for successful compilation of `bnldspMM.exe`. For that reason we find in the `NRI_INIT` directory a default `bnldpsXx.nri` file that can be installed in the tool's root directory (with the correct names) by running script '`cp_inits.bat`' (MS Windows environment) from that directory. Doing '`gmake init`' at the tool's root does the same for UNIX environments.

This allows successful compilation of the tool. During the first run, without the `–chk` option, the tool will overwrite the default `*.nri` files with correctly defined versions from the current memory map. After that correct another compilation should be done and also the `–chk` option for the tool will then perform as required when the tool is run.

All hardware defined used constants are entered in lookup table initializer files that carry the extension `*.lui`. These files initialize structures (lookup tables) which are type declared in the file `bnldspLU.h`. A programme module needing a particular lookup table defines a variable of the required structure, by including file `bnldspLU.h` for the table's type, and also includes the corresponding `*.lui` file for its initialization upon programme load time.

> **Important remark:**
> It is important to remember that the constants as defined in the hardware developer's files in the `RTT_shared` and `DSP_shared` directories are only used **once** in the software developer's code files, namely either directly in the `bnldspLU.h` file for simple constant (re-)definition or in the `*.lui` initializer files for the lookup table definition.
> All access to the hardware developer's constants in the software developer's code files thus passes via a lookup table access by using an enumerated type or `#defined` label from the `bnldspLU.h` file for getting the constant value definition.
> This strategy of namespace separation allows for maximum isolation (thus: freedom) between the 2 worlds of development and gives the software development code enhanced robustness.

Upon successful run of the `bnldspMM.exe` programme, and subsequent careful checking of the created log and map files, we can instruct via the batch script `cp_MMres.bat` (MS Windows) or, in the UNIX world, by doing '`gmake install`', to copy the set of tested header and initializer files as well as the used definition files from the `RTT_shared` and `DSP_shared` directories into the generic `GEN_HEAD` and `RTT_shared` and `DSP_shared` directories, situated in the project root `BNLDSP`. We thus make all these files available for compilation of all C code modules.
In the UNIX world we can also do '`gmake fecInstall`' and this same copying process will upload the files into the frontend computer software repository currently on path:
`/ps/local/ppc4/BNLDSP`
This way the files are accessible when compilation of the Fesa Classes is performed during the installation of the latter on a particular frontend via '`make new_dtab`' in directory:
`/ps/src/dsc/<accelerator>/<frontendName>`
(See note: 'AB front-end.pdf' and the chapter 'The 'make recipe'.' for more details.)

The driver simulator library.

The driver simulator library is created in the root of its directory tree: `bnldspLI`. It uses the generic header files currently available from the `GEN_HEAD` , `RTT_shared` and `DSP_shared` directories, which thus are required to be checked and 'up loaded' by the `bnldspMM.exe` tool beforehand.

After being created, the library is then available for incorporation in the executable programmes that simulate an environment for the C functions intended for use by Fesa device classes. The library will finally be stored in the generic linkable libraries directory of the BNLDSP directory tree.

The executable programmes will output into a logfile (`bnldspX0.out`) the data that would normally be downloaded into the Bnldsp hardware and, when used in hardware read mode, after correct initialization by first writing some data, will return the aforementioned initialization data when the code performs a hardware read operation. For a more explicit explanation see the 'The 'per Fesa Class' C code directories.' chapter.

The generic header file directories.

The following directories contain header files that are included in all `BnldspXXX` C code functions that will be incorporated into the Fesa BnldspXXX device classes. Being 'generic' means that only one definition of these files exist and therefore the definition of memory map and other constants will be guaranteed the same all over the LEIR BNLDSP project.

Important note:
I'd like to stress that these following 3 directories, later, when compiling the Fesa classes and also when compiling / linking for the FrontEnd Computer executables, are also to be considered as generic for all BNLDSP Fesa classes and FrontEnd Computer executables. Fesa is organized on a 'per device class' basis, while here, for the C code function code we rather have a 'several Fesa classes generic' approach for some of the header (include) files. The importance of this statement as well as the solution that was chosen for the dilemma created by these two different approaches will become clear later in this note.

The `bnldspMM.exe` tool is responsible for populating the `GEN_HEAD`, `RTT_shared` and `DSP_shared` directories with all (checked!) files that are needed for the correct compilation of all C code functions. The hardware driver simulator library is made generically available in the `LIB` directory either by a script '`cp_LIdrv.bat`' (MS Visual C++ environment) or by '`gmake install`' (UNIX environment).

The GEN_HEAD directory.

Are stored here, 'uploaded' by running the batch script `cp_MMres.bat` (MS Windows environment) or by doing '`gmake install`' (UNIX environment) from the `bnldspMM.exe` tool for the first 5 bullets; and: by running the batch script `cp_LIdrv.bat` (MS Windows) or by doing '`gmake install`' (UNIX) from the `bnldspLI` tool for the last bullet:

- The C code functions memory map declaration file: `bnldspMM.h`
- The C code functions named region map declaration file: `bnldspNR.h`
- The C code functions lookup table and constants declaration file: `bnldspLU.h`
- The set of C code functions lookup table initializer files: `*.lui`
- The created Named Region lookup table initializer files: `bnldspX0.nri`
- The BNLDSP driver library (and also simulator driver library!) function prototype and other definition files:
  `bnldsp.h`, `BnldspIoctlAccess.h` and `BnldspUserDefinedAccess.h`

The DSP / RTT_shared directories.

Here we only find the checked 'hardware developers' constants and structures definition files, 'uploaded' in the `bnldspMM.exe` tool root by running the `cp_MMres.bat` script (MS Windows environment) or by doing '`gmake install`' (UNIX environment).

The generic linkable libraries directory.

Running, in the `bnldspLI` tool root, the driver simulator script '`cp_LIdrv.bat`' (MS Windows environment) or '`gmake install`' (UNIX environment) will store in this directory the driver simulator library for incorporation by any of the executable programmes that simulate hardware access via the BnldspXXX C code functions. This is currently the only generic library available in the system.

The generic 'Make' source directory.

In this directory we find include files for the definition of generic header file dependencies and 'make recipes' of C function libraries in the BnldspXXX sub projects. Indeed, in all of these sub projects we can distinguish similar C code functions that we thus can compile under very similar conditions.

Therefore it is useful to define these 'similar make recipes' only once, include that definition in the each of the sub project's specific 'make file' and further customize them with sub project local, specific definitions (macros and targets) before the resulting 'make' is effectively run.

Note that not all sub projects need to include all of the body include files and that all these 'Make sources' are only used in the UNIX environment.

We find here:

- Generic Fesa development environment macro definition files: `*.def`
- Generic header dependency files: `*.dep`
- Generic makefile body files: `*.bod`
    - for the 'main make body' part
    - for the 'printer functions' library
    - for the 'property functions' library
    - for the 'auxiliary functions' library
    - for the 'RT actions functions' library

The 'per Fesa Class' C code directories.

Preliminary note:
As the BNLDSP directory tree will be created from the CERN Central CVS repository (CVS root module: '`abrfcs`') in your own CVS sandbox, we shall be discussing here only in terms of compilation / library and test programme creation for UNIX systems. For testing in an MS Windows environment, a Visual C++ project structure should also be created on a 'per sub project' and even on a 'per sub project's / per functions library' basis. Although the various batch scripts needed in the MS Windows test environment are stored in the CVS repository and would thus be available in your local BNLDSP tree, it would probably be better to ask Erik how to set up such an MS Visual C++ test system before being able to use it correctly...

We can split all BNLDSP 'C code functions' sub projects of the 'per Fesa Class' C code directories in roughly 2 types:

- The relatively 'complex' ones ('type 1'), currently: `BnldspCTL` and `BnldspTIM`
- The relatively 'simple' ones ('type 2'), currently: `BnldspBQ, BnldspDC, BnldspGF, BnldspMB, BnldspMD, BnldspRE, BnldspSC, BnldspSD`

The first type has available a test program that can simulate the LEIR machine cycle with PPM sub cycles. These functions, normally called by the Fesa device for which they are intended, can thus also be called by that main programme and then extensively tested in e.g. the MS Visual C++ environment where the evolution of variables on stack and function calls can be observed, break pointed and single stepped. For the main programme to work, it needs to be linked to the driver simulator library.
The Fesa required C functions are gathered in the 'Real Time functions' and in the 'auxiliary functions' libraries. Are also created for internal use by the C functions of the sub project the libraries with the property C++ interface functions and the sub project specific printer functions.

For the creation of the afore mentioned executable test programme of this first type, we find in here, for the UNIX environment, a `GNUmakefile`, that, together with the available generic `MAKE_SRC` directory resident include files and the local `mk_main.tgs` file specialize this `GNUmakefile`. See for further details the 'The 'make recipe'.' chapter.

The second type does not have a main programme available and only the 'auxiliary functions' and 'RT functions' need to be implemented in their respective libraries. Being fairly simple and straight forward, the fact that no simulation can be done in a test environment is not really a big problem here.

Both types however, are able to create very 'verbose' logfiles and can be 'programmed' at start-up via 'command line arguments', either from the commandline (test) or, if so instructed, read from a pure ASCII text file (run time; in operation); file: `gEnvInit.cla` generally located in the startup (work-) directory. This latter facility is implemented in a special RT function (doCLibGlobEnvInit) that is only called at startup time by the BNLDSP Fesa device class concerned via the Fesa specificInit() facility.

The following subdirectories can be distinguished in each of the 'C code functions' sub projects:

The HEADERS subdirectory.

Here is initially stored the sub projects' specific (main) header file: `bnldspXX.h`. This directory is 'generic' within the sub project, which means that it might be searched for header files when creating the other sub project's function libraries. During the making of the project there will also be installed the C code functions libraries header files such that a possible test programme can access the functions of these libraries.

The XXXfuns, XXXrts directories.

Here we find the C code functions that will finally be used by the Fesa device classes. Notably these are intended for use by the RT part of the Fesa concept. With 'XXX' we mean the sub project's usual initials, e.g. '`CTL`' for the `BnldspCTL` sub project, or '`BQ`' for `BnldspBQ` (required for Fesa device class BnldspBIQUAD!).

We also find here a `HEADERS` directory that contains specific header files for the creation of the library concerned, either the auxiliary functions library or the RT functions library. Inside this latter directory we might find header files that were stored while creating the other XXX libraries. Indeed, the order in which the libraries must be created is imposed. See for further details the 'The 'make recipe'.' chapter.

The C code functions of these directories can be gathered into libraries. For that reason there exists here, for the UNIX environment, a `GNUmakefile`, that, together with the available generic `MAKE_SRC` include files and the local, either `mk_funs.tgs` or `mk_rts.tgs` file, specialize this `GNUmakefile`. See for further details the 'The 'make recipe'.' chapter.

The XXXprpL, XXXprtL directories.

These directories, which only contain files in the 'type 1' group of sub projects, are similarly organized like the `XXXfuns` and `XXXrts` directories. In here we can create the current sub project internally used libraries with functions that interface the Fesa properties (`XXXprpL`) and sub project specific (log file) printer functions (`XXXprtL`).
Note that in the whole sub project we only find in the `XXXprpL` directory a Cplusplus code file: `XXXprpL.cpp` which is a source file that conditionally adds some C++ code (compilation flag `FESACODE` defined) but, when compiled for use in the test environment, it is a 'pure' C source file containing 'pure' C code, no C++. However, as the possibility now exists that some C++ code is added, compilation should be done with a C++ compiler under all conditions.

The XXXsrvs directory.

Currently this directory is not (yet) used, it is empty for the moment. It is intended to follow the general scheme of library organization and creation; in this case it would be filled with C code for a functions library to be used by the server part in the Fesa paradigm.

The FDI_INIT directory.

The 'type 1' sub projects use a concept of an array of internally declared 'Fesa devices'. This array needs to be initialized at the moment of programme load. For that a 'Fesa Devices Initializer' file is used which can be found in the sub programme's root directory; normally named something like `fesaXdev.fdi`. Also in the other functions libraries such a file could be used, generally they will be named like: `<libraryDirectoryName>.fdi`.

Occasionally these files need to be modified, e.g. when the layout of the internally used ‘Fesa device’ structure changes. This is a rather delicate operation currently done ‘by hand’ and, in order to recover from erratic behavior (compilation) after modification, this directory carries a ‘working’ backup of these files. Overwriting the existing `*.fdi` files goes with: ‘`gmake fdiInit`’

In the future it could be foreseen that the test programme provides an option to create these files from currently initialized memory during its run time, but at the moment this does not yet exist.

The EXCEL directory.

This is currently also only existing in ‘type 1’ sub projects. It contains some MS Excel spreadsheets that are used for documentation and also for the calculation of the value for some `#define` constants that are used in the C code header files of the current sub project.

Note on the FESA / FEC management integration.

The 'per Fesa Class C code functions' were initially gathered in C libraries and integrated in the Fesa environment during the linking phase of the Fesa device classes. This way of integration of C code is an available feature of the Fesa development world and it was used during manual testing. When tried to implement this approach for the automatic boot configuration of a Front End Computer (FEC), it appeared to be not that easy. Therefore a slightly different approach had to be adapted.

By integrating the compilation of the C code modules during the compilation of the Fesa C++ class code rather than using a library later, the C code modules will be added into the Fesa module of the 'Common' pack which is actually a library in its own right. During the linkage phase of the Fesa Real Time and Server executables (and the 'Test' single process) the C code functions will then be put in place. When 'delivering' (`Fesa Deliver <FesaClass> <version> <CPU>`) to the FEC management, all code for a functioning Fesa class will thus automatically be present in that environment for each of the Fesa class executables.

Of course, such an approach requires that the paths to the various header (include) files and C source code directories be known to the Fesa and FrontEnd Computer compilation environments. It was explained that the developer of Fesa source code is allowed to create directory structures of his own inside the standard compilation directories of the Fesa environment. And that is exactly the way that was adopted. (Thanks for your explanation that lead to this, Michel!). It required only a small adaptation to the '`make.specific`' file present in the Fesa development `COMMON` directory (and in the future possibly also in the `RT` and `SERVER` directories, if that would be needed) and a subsequent adaptation in the resp. '`makefile`' templates. (Adding a `SPECIFIC_CFLAGS` macro + its treatment for C compilation and making '`make clean`' properly cleaning the C code `*.o` files.)

Another problem appeared in the FEC management when creating a new loadable data table for a front end machine ('`make new_dtab`'). At this stage the standard installation system asks whether one desires to 'compile', question that should be answered at least once with 'yes'. So, also the FEC management system must also have some knowledge of the C code files directories. Note in this respect that in the `makefile` for compilation over there the already declared macro (but not defined) `FESA_EXTRA_LIBS` had to be defined for the linkage to the operational driver access libraries of the executable.

This problem could be solved by creating the 'generic' header directories in the common (FESA and FEC worlds) accessible source repository via: `/ps/local/<CPU>/include`.

All together the following steps were necessary to make compilation of all code and FEC autoboot work:

- Create in the standard 'delivered source' repository at: `/ps/local/ppc4/` the partial `BNLDSP` tree:
  `BNLDSP/GEN_HEAD`
  `BNLDSP/RTT_shared`
  `BNLDSP/DSP_shared`
  and populate these directories with the required files. It can be done by the `bnldspMM.exe` tool's '`gmake fecInstall`'. See also details in the chapter The 'make recipe'.

- Create a symbolic link to this partial `BNLDSP` tree in the standard 'include' directory at: `/ps/local/ppc4/include` which makes C code modules compilation work for both FESA and for the FEC worlds.

- Create in the standard CVS Fesa development repository, for each of the Bnldsp Fesa device classes, in their directory `COMMON`, the partial `BNLDSP` tree:
  `BNLDSP/<BnldspXX>/HEADERS`
  `BNLDSP/<BnldspXX>/<XXXfuns>`
  `BNLDSP/<BnldspXX>/<XXXfuns>/HEADERS`
  `BNLDSP/<BnldspXX>/<XXXrts>`
  `BNLDSP/<BnldspXX>/<XXXrts>/HEADERS`
  `BNLDSP/<BnldspXX>/<XXXprpL>`
  `BNLDSP/<BnldspXX>/<XXXprpL>/HEADERS`
  `BNLDSP/<BnldspXX>/<XXXprtL>`
  `BNLDSP/<BnldspXX>/<XXXprtL>/HEADERS`
  Note that for 'type 2' C code some of these directories will be left empty (for the moment at least). To

populate this tree with required source files we use in the BnldspXX directory: 'gmake fesaInstall'. See also details in the chapter The 'make recipe'.

The <XXXprpL> and <XXXrts> directory resident C code functions are exclusively used in the Fesa RT code and could therefore be implemented in this code's RT directory only. However, compilation of all the C code in this BNLDSP sub tree requires the hierarchically, 'one-directory-up' positioned directory BNLDSP/<BnldspXX>/HEADERS available. Moreover, incorporating the sub-tree in COMMON also better reflects the situation of the C code development environment and therefore helps the programmer's view of the whole (advantage for maintenance over time).

The nice effect of this scheme is that the usual 'Fesa Commit' or 'Fesa Deliver' would now also incorporate the CVS upload of this directory structure and all C code in there that is used for the Fesa class.

Finally it is pointed out that for correct startup of each Fesa class, the RT executable requires a text file (gEnvInit.cla) that contains the 'Command Line Arguments', read by the C code RT function doCLibGlobEnvInit(), function that executes at class (re-)startup only. Also, for logging purposes, a place for storing the logfile should be available.

The traditional FEC place to retrieve / store executable programme data in run-time is:
/dsc/local/data
which corresponds to the directory path on cs-ccr-dev1:
/acc/dsc/<accelerator>/<FEC>/data
and where is: accelerator: 'tst' for test systems; 'lei' for LEIR operational systems.

- We created in there the partial BNLDSP tree:
  BNLDSP/<FESA class name>
  where 'FESA class name' is used as the work directory from where the corresponding Fesa class code is started and in which its output log file will be created.

It should be remembered, that the Fesa executable code will always read or write ('hard coded' patch in the executable during compilation) the <FESA class name>DeviceData.xml and <FESA class name>PersistentData.xml documents from the following directories:

- for operational code: /acc/dsc/<accelerator>/<FEC>/data
  (on the FEC accessed as:) /dsc/local/data
- for test environment: executable startup directory: ./
  (which is usually:)
  <FESA test env.>/<FESA Class>/v<x>/TEST/<FEC>

To make use of this infra structure, a special set of BNLDSP_X, BNLDSP_X_DLEITST and BNLDSP_X_NOSTART macros have been created for use at the FEC management database. Here, 'X' can be either 'M', 'R' or 'S', designating the executables for 'Mixed' (single process: RT and Server), 'Real Time' or 'Server' respectively. These macros serve for the definition of a line in the startup sequence as can be found in the transfer.ref file. This note will not go into further detail of these issues.

For 'make transfer.ref' and 'make new_dtab' see documents: 'AB front-end.pdf', 'fecConf.pdf', and 'Starting User Programs on LynxOS, Linux and HP-UX computers.pdf'.

The 'make recipe'.

A complete cycle of updating and regeneration of the C code functions is a 3 step process:

- Firstly, in the software development environment, we adapt and run the tools for getting a new set of base `bndspMM.h`, `bnldspLU.h` and `bnldspNR.h` header files and lookup table initializer (`*.lui` and `*.nri`) files.
- Secondly we adapt all the C code functions of all sub projects, compile for the software development environment and test there the 'type 1' class of sub projects and after success: compile the 'type 2' sub projects.
- Thirdly install all modified files in the operational Fesa and FrontEnd Computer environments, compile for those environments and final test.

Remember that all 'make' files offer target explanation with: `gmake<CR>` in their resp. directories. The same, all tools (`bnldspMM.exe`, `BnldspCTL.exe`, `BnldspTIM.exe`) offer commandline argument help by doing on the commandline: <`executableName`>`.exe -?`

First step.
Adapting and running the tools goes via:

- Updating the `DSP_shared` and `RTT_shared` directories in the `bnldspMM.exe` tool's root directory `bnldspMM` with a fresh set of 'hardware developers' source header files from the LEIR projects depository (see chapter: 'Checking the memory map.').
- Adapting the Excel spreadsheets in the EXCEL directory if so required and recalculate new `#define` constants for the base C header files.
- Adapting the 'all C functions' base header files: `bndspMM.h`, `bnldspLU.h` and `bnldspNR.h` in the `bnldspMM.exe` tool's include file directory `HEADERS` if so required.
- Adapting various lookup table initializer (`*.lui`) files in the `bnldspMM.exe` tool's include file directory `HEADERS`; possibly creating new ones for newly declared lookup tables in the base header files.
- Adapting the various printer / checking parts in the C code of the `bnldspMM.exe` tool (file: `bnldspMM.c`). Particularly involved subroutines are likely to be (**non** exhaustive list!): `fPrintMemMap()`, `initLookup()`.
- Re-initializing the `bnldspMM.exe` tool's `*.nri` default named region tables initializer files. This is done by (at the tool's root directory): `gmake init`
- Compilation of the `bnldspMM.exe` tool (and, most likely, solving some compilation problems here…).

  Compilation is done by (at the tool's root directory): `gmake clean all`
- Running the `bnldspMM.exe` tool without the `–chk` commandline option, thus creating a new set of `*.nri` files derived from the current memory map.
  Do it like: `bnldspMM.exe –db –fl –ml –nl –ol –tl -al`
- Checking the by the `bnldspMM.exe` tool created log / map files to see that the memory map is exactly what we hoped it would be.
- Re-compilation of the `bnldspMM.exe` tool with the now correct `*.nri` files. (Note that we do no '`clean`' here as to preserve `bnldspMM.exe` first run's log output already in the logfile.)
  We do: `gmake all`
- Re-run the `bnldspMM.exe` tool, now with the `–chk` commandline option such as to check the newly created named region lookup tables against the current memory map.
  Do it like: `bnldspMM.exe –db –fl –ml –nl –ol –tl –al -chk`
- If all this seems fine, install the complete set of header / lookup table initializer / named region initializer / DSP/RTT shared files from the `bnldspMM.exe` tool root directories in the `BNLDSP` root generic

GEN_HEAD and RTT/DSP_shared directories.
Done by: gmake install

- Generate a new version of the bnldspLI hardware driver simulator library with the new generic header files.
  Done in this tool's root directory (bnldspLI) by: gmake clean all
- If all this seems fine, install the complete set of driver header files in the BNLDSP root generic GEN_HEAD directory.
  Done by: gmake install

Second step.

We now describe the adaptation and compilation of the 'type 1' sub project BnldspCTL. The other 'type 1' sub project is done in a similar way. The remaining, 'type 2', sub projects are also similar, be it that they have no C code for XXXprpL and XXXprtL functions / libraries and they have also no executable test programme available. These latter sub projects thus only follow a partial set of the following list.

Assuming we now have the sub project's root directory as work directory, it then goes like this:

- Adapting the 'all BnldspCTL C functions' base header file: bndspCT.h in the BnldspCTL base include file directory HEADERS if so required.
  Notably the properties and device instances enumerated types might be involved with the changes. These changes could certainly have a 'ripple through' effect all over the C code, everywhere!
- If property and / or device instance enumerated types in file bndspCT.h have changed, backup the currently used (working) fesaCdev.fdi file in the directory FDI_INIT and adapt this file in the sub project root to reflect the changes.

For the now following adaptation / compilation phase of the sub project's libraries, it should be noted that each of these libraries form a sub project on its own. It is thus possible to individually compile each of them with gmake <target> in each of their root directories; see the possible targets with the help of: gmake<CR>.

All library 'makes' require the macros CPU=ppc4; FESA_VER=v2 (or v0, v1 etc.), FESACODE=TRUE and REALDRV=TRUE (or FALSE, the default value for both) defined on the gmake commandline.
Macro CPU is required for FESA / FEC world; FESA_VER defines the Fesa class version directory to be used, FESACODE enables (or disables) inclusion of specific C++ code when compiled for inclusion in the operational Fesa class and REALDRV=TRUE enables linkage to the real BNLDSP driver whilst REALDRV=FALSE enables linkage to the local simulator driver library of the test system (only having a meaning for the 'type 1' C functions).
Note that it is in principle also possible, for test purposes, to use the driver simulator library with operational Fesa code; it requires however that the make.specific file in the Fesa development environment's TEST directory is instructed to use the correct path for finding it during link time of the executable. See instructions over there.

- Possibly adapt the 'printer functions' library C source file, CTLprtL.c, and try to re-compile it with (we assume macro CPU defined in the user's environment):
  gmake clean all install FESA_VER=v1 FESACODE=TRUE REALDRV=TRUE
  If successful, next step.
- Possibly adapt the 'property functions' library C source file, CTLprpL.c, and try to re-compile it with:
  gmake clean all install FESA_VER=v1 FESACODE=TRUE REALDRV=TRUE
  If successful, next step.
- Possibly adapt the 'auxiliary functions' library C source file, CTLfuns.c, and try to re-compile it with:
  gmake clean all install FESA_VER=v1 FESACODE=TRUE REALDRV=TRUE
  If successful, next step.
- Possibly adapt the 'RT functions' library C source files, gEnvInit.c, doPrepDL.c, doAcq.c, doDL.c, doRstDSP.c and doUpdDev.c, and try to re-compile them with:
  gmake clean all install FESA_VER=v1 FESACODE=TRUE REALDRV=TRUE
  If successful, next step.

Note that there is a batch script available, `make_all_libs.bat`, that does it for all 4 libraries of a sub project for you (Attention! The `FESAINSTALL` macro definition and the Fesa class version directory name are mandatory, the others optional; default definition is: '`FALSE`'.):

```
make_all_libs.bat FESAINSTALL=FALSE v1 FESACODE=TRUE REALDRV=TRUE
```

There also exists a batch script in the BNLDSP root (one directory 'up'), `make_all_libs.bat`, that does the compilation for all sub projects, all libraries. This script has all currently used Fesa classes version directories already coded 'in'! Not required to specify any on its commandline.:

```
make_all_libs.bat FESAINSTALL=FALSE FESACODE=TRUE REALDRV=TRUE
```

Third step.

We again describe the C code installation of the 'type 1' sub project BnldspCTL only. Reasoning for the other 'type 1' and 'type 2' sub projects is the same as for the second step described above. Of concern is here only the way of how to port the various header and other C code sources, after the former tests successfully performed, into the Fesa development and into the FrontEnd Computer installation / management (FEC auto boot of Fesa device classes) environments. Is here assumed that in the FEC management environment and in the Fesa BNLDSP CVS sandbox (in the Fesa class sub directory `COMMON` and possibly also those in the `RT` and / or `SERVER` directories) the correct BNLDSP directory trees (partial structures from the C code development BNLDSP tree) are already existing. This is how 'uploading' to these environments then goes:

- Updating the generic `BNLDSP` header file directories (`BNLDSP/GEN_HEAD`, `/RTT_shared`, `/DSP_shared`) is done with the `bnldspMM.exe` tool's 'make'. We can check where these generic directories are located by inspecting the tool's GNUmakefile.
  We do in directory `BNLDSP/bnldspMM`: `gmake fecInstall`

- Updating the Fesa class `BnldspCTL/<version x>/COMMON/BNLDSP` C source code tree is done with the development's BnldspCTL tool's 'make'. We can check where the class specific destination directories are located by inspecting the tool's GNU`makefile` and its 'make' include files `mk_main.bod` and `mk_main.tgs`.
  We do in directory `BNLDSP/BnldspCT`: `gmake fesaInstall`

The batch script of the second step section above is also capable of installing the C source code in the Fesa environment. Either for one sub project (in the sub project's root, `BnldspCT`, here):

```
make_all_libs.bat FESAINSTALL=TRUE <Fesa class version x dir name>
```

or, in the `BNLDSP` root for all sub projects in one go:

```
make_all_libs.bat FESAINSTALL=TRUE
```

- Generation of the Fesa executable code in the usual way should now be possible. Testing correct executable code on test FEC `dleitst3` should confirm this; the modifications that started off this development iteration be checked. For making the test packages we do in the Fesa environment's `TEST` directory:
  `gmake all` (for the `BnldspCTL_M` (Mixed, RT and Server) single process executable)
  `gmake rt` (for the `BnldspCTL_R` (RT) executable)
  `gmake server` (for the `BnldspCTL_S` (Server) executable)
  `gmake shared` (for the creation of a FesaShared_S server executable; not used for `BnldspCTL`, just for information, not further discussed here for the moment. See the Fesa documentation: 'Fesa Equipment Links' and the document 'The Bnldsp Fesa class quartet BnldspSIGNAL, BnldspTRIG, BnldspMUX, BnldspSCOPE forming LEIR's RF beamcontrol signal interface to the OASIS data presentation layer' where a FesaShared_S server is actually built.)

- When all is OK; we should deliver the modified code to the operational environment. This is done on the service machine `cs-ccr-dev1` in the development Fesa CVS sandbox for the BnldspCTL device class by the usual:
  `Fesa Deliver BnldspCTL 0 ppc4`
  Note that the `Fesa Deliver` script is doing an inherent `gmake clean all` which means that previously created RT and Server executables have been destroyed. Re-make them as described above!

- Check that the `FESA_EXTRA_LIBS` macro in the makefile of the FEC environment at path:
  `/acc/dsc/src/tst/dleitst3` is correctly defined for access to the Bnldsp driver library as:

```
FESA_EXTRA_LIBS = -L/ps/local/ppc4/drvrutil/lib/Bnldsp \
                  -lBnldspIoctlAccess -lBnldspUserDefinedAccess
```

If this Fesa class still needed to be incorporated in a frontend computer's startup sequence, we need to make a new startup `transfer.ref` file. For that, first enter the class in the FEC management database (CO-DM's Controls configuration portal via: http://ab-div-co-dm.web.cern.ch/ab-div-co-dm/. Of interest is the link 'AB Controls Configuration Applications') and add a line in the FEC's startup sequence over there. I do not go into the detail of this issue here.

- A new `transfer.ref` is then generated e.g. for test FEC `dleitst3`, on the service machine `cs-ccr-dev1` in the directory: `/acc/dsc/src/tst/dleitst3` with:
  `gmake transfer.ref`
- Finally we have to re-generate the executable code for the operational FEC management environment. That is currently also done, e.g. for test FEC `dleitst3`, on the service machine `cs-ccr-dev1` in the directory: `/acc/dsc/src/tst/dleitst3`. We do:
  `gmake new_dtab`
  and answer the questions whether compilation and subsequent installation must be done both with 'yes'.

  Important remark on this last point:
  When we have defined in the deployment document of a particular frontend computer that this Fesa class should be 'manually' started, `gmake new_dtab` does **not** compile the (Fesa Delivered) code and does **not** install the binary in the AB-CO repository at:
  `/acc/dsc/<accelerator>/<FEC>/bin`
  and also does **not** extract from the database and install the Fesa `<class>DeviceData.xml` file in the usual directory at:
  `/acc/dsc/<accelerator>/<FEC>/data`

  Indeed, in the Fesa deployment tool the term 'manual' means that the user is supposed to create a binary and run this code by himself **and** in his own file space, where he has also manually put the Fesa database extracted device data file (`<class>DeviceData.xml`). It is thus considered not to be required for the FEC software installation system to handle class installation in any way.

  Compilation and subsequent installation for a frontend computer is **only** done when a Fesa class deployment is declared as 'automatic' startable!

  Hint:
  If this is not what a user wants, then he must declare the deployment of a FESA class in a particular FEC as 'automatic' startup and define in the FEC startup database the 'Inhibit' flag. The startup file (`transfer.ref`), when recreated with `gmake transfer.ref`, will then have the startup of the FESA class 'commented out' and thus effectively not started.
  For BNLDSP usage we have defined the special FEC database macros `BNLDSP_M_NOSTART`, `BNLDSP_R_NOSTART`, `BNLDSP_S_NOSTART`, which use the binary file copy mechanism of `transfer.ref` but the latter does not launch the executable(s) at FEC startup. The user finds however in the regular BNLDSP Fesa class startup directory the executable(s) that must be started 'manually'!

More information on FEC management installation / compilation etc. issues can be found in the documents: 'fecConf.pdf' and: 'AB front-end.pdf', to be found in the `DOCUMENTS` directory.

___