DRAFT 0.3

Beaglebone Black
based
ALE Modem/Controller
Development Plan

By N2CKH

This is a living development plan document presenting my vision for an Embedded ARM host Linux operating system based, Software Defined ALE Modem/Controller solution based on the Beaglebone Black (BBB) Embedded Linux developers board as the host.

I would obviously prefer to make use of an inexpensive hardware host that has been specifically designed and packaged as an Open Software Defined DSP based HF modem/controller for which a Software Developer Kit (SDK) has been published. However as no such device exists, the BBB has been chosen as the next best approach.

The goal with BBB-ALE is to provide the MARS user and the average Amateur Radio user an affordable hardware based software defined ALE modem/controller solution to both meet the ALE standard requirements and the end users budget. BBB-ALE in my vision is seen as a low cost build it yourself ALE Modem/Controller approach using an embedded Linux development board.

The BBB is currently the lowest cost ARM based developers board on the market that is readily available world wide on a Commercial-Off-The-Shelf (COTS) basis, which has the mix of processing power and feature sets that I believe will support a DSP software defined ALE approach.

In the x86 world, a new relatively low cost option that can be viewed as competition to the BBB and a host of other ARM development boards are making their debut. However most are twice the cost or more of the BBB at $45USD. The x86 based Intel Atom E3825 based board, running Debian, at $99USD in single core is twice the cost of a BBB and $30 more in dual core, the later of which is the model that interests me most. The Minnow Board (http://www.minnowboard.org/meet-minnowboard-max/) which is coming out mid 2014, I will be watching closely. I am currently working with the more expensive Atom N2800 family in a COTS Embedded Industrial Computer approach with both Linux and Windows Embedded Standard 7 (WES7) in support of MARS and plan to do the same on that platform with the coming Atom E3845 processor.

What is an Embedded Operating System?

An embedded operating system is an operating system for embedded computer systems. These operating systems are designed to be compact, efficient at resource usage, and reliable, forsaking many functions that non-embedded computer operating systems provide, and which may not be used by the specialized applications they run.

What is Embedded Linux?

A Linux distribution that targets one or more embedded processor devices.

The advantages of embedded Linux over proprietary embedded operating systems include multiple suppliers for software, development and support; no royalties or licensing fees; a stable kernel; the ability to read, modify and redistribute the source code, ease and restriction free Live boot media distributions. In addition Linux can be completely separated in operation from is GUI interface resulting in less CPU and memory overhead.

The technical disadvantages include a comparatively large memory footprint (kernel and root file system); complexities of user mode and kernel mode memory access, and a complex device drivers framework and the lack of specific tools to address embedded scenarios and hardware specific OS versions which may only permit software floating point optimization.

What are the driving factors for Embedded Operating System Development?

The cost of hardware solutions and achieving proper performance in adhering to the MIL-STD and STANAG requirements and consistent results, period.

New COTS ALE radios have come down, one manufacturer offers an HF transceiver with ALE option when all configured for about $1,500USD that actually works properly to FED-STD-1045A but where it is only marketed in the U.S. for Government sales. New MIL-STD modems however are more easily purchased, however only modems in the MS-110A/S4285 class come in under $2,500USD presently, MS110B/S4539 are more expensive. The used equipment market is costly as well and if one does get a deal on the hardware, the cost of the required software to program or operate and the documentation adds up rapidly.

ALE hardware alternative

The low priced ARM based BBB development board is the obvious choice as a target host platform based on price point alone. The BBB, an Audio Codec along with UART to RS-232 and a suitable case when connected the USB client cable to a PC running PuTTy for Man Machine Interface (MMI) represents a minimum BBB-ALE configuration and total estimated cost of approximately $175USD. The addition of operational configuration options presented later herein (e.g. LCD display module) by the end user do not enter into the initial cost equation.

In addition to ALE Modem/Controller support, the BBB may be able to support a full ALE Modem/Controller /w MIL-STD Data Modem capability predicated on its ability to support the CPU loading of the data modems Digital Signal Processing requirements implemented in software.

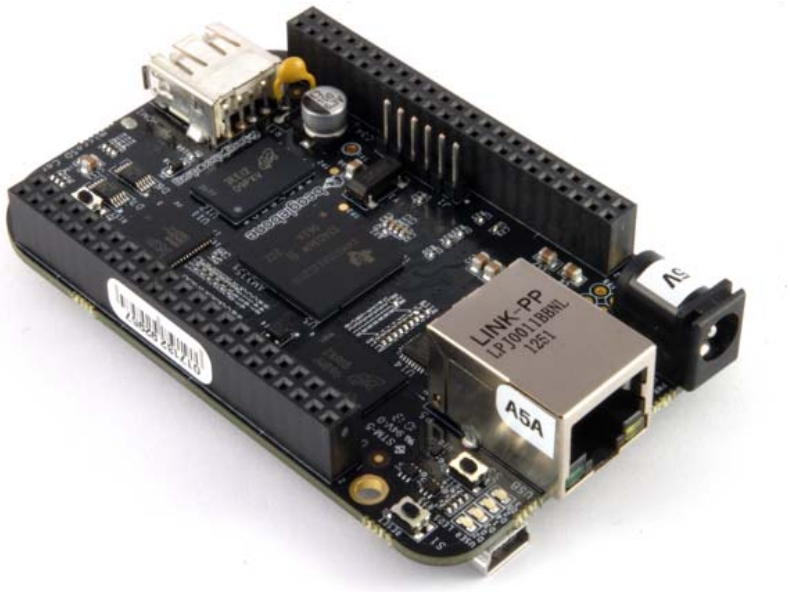
Preliminary Efforts

I have performed a great deal of study regarding both embedded Linux and Windows Embedded Compact 7 (WEC7) development. The ability of embedded Linux on an ARM processor to support a software defined ALE modem and its DSP processing requirements seems to be a given, whereas the ability to do so for a MIL-STD Data Modem is questionable. However the ability of an ARM/WEC7 combination on the BBB in providing the required floating point DSP processing required of in a software defined implementation without presenting too much CPU or RAM use for MIL-STD Data Modem use is very doubtful. There is however the potential of doing so as an ALE modem/controller, which may be explored at some point.

I have been configuring cross development platforms and tool chains and building image distribution libraries for ARM and Angstrom Linux targeting the Beaglebone Black (BBB) for evaluation. I have now started doing so for the recently released official Debian image for the BBB as I have now installing it for development and testing in my current STANAG data modem evaluation efforts. I also have the WEC7 BSP for the ARM on the BBB and will investigate that as well via an SD boot or a second BBB.

Why the BBB?

There are many ARM single board computers on the market under $100USD and more coming along all the time. The important thing as from the technical aspects in my view is that the selected host(s) be backed by a large manufacturer where they have worldwide COTS distribution for easy to purchase and were the host is well documented and has decent user community support networks.



Keeping that in mind, the most widely available, to now include walking into your local Radio Shack (where it is value added packaged with extras at additional markup cost) is

the very popular Beaglebone Black (BBB) TI ARM Cortex-A8 1Ghz CPU based board for $45USD running embedded Linux.

http://www.youtube.com/watch?v=FcqQvH41OR4

Designed by TI engineers and having a large and growing user community support model there is a ton of material and many embedded operating systems choices to includes WEC7 for the BBB.

The features of the BBB (http://beagleboard.org/Products/BeagleBone+Black) are rather impressive for its cost per unit and size.

- Processor: TI Sitara AM3359AZCZ100, 32-Bit RISC Microprocessor, 2000 MIPS@1GHz (MIPS/MHz = 600/300, 1200/600, 1600/800)
- Graphics Engine: SGX530 3D, 20M Polygons/S
- SDRAM Memory: 512MB DDR3L 606MHZ
- Onboard Flash: 2GB, 8bit Embedded MMC
- PMIC: TPS65217C PMIC regulator and one additional LDO.
- Debug Support: Optional Onboard 20-pin CTI JTAG, Serial Header
- Power Source: miniUSB USB or DC Jack, 5v D.C. External Via Expansion Header
- PCB 3.4" x 2.1" 6 layers
- Indicators 1-Power, 2-Ethernet, 4-User Controllable LEDs
- HiSpeed USB 2.0 Client Port: Access to USB0, Client mode via miniUSB
- HiSpeed USB 2.0 Host Port Access to USB1, Type A Socket, provides 5v up to 500mA LS/FS/HS
- Serial Port UART0 access via 6 pin 3.3v TTL Header. Header is populated
- Ethernet 10/100, RJ-45 connector
- SD/MMC Connector microSD , 3.3v
- User Manual Input: Reset Button, Boot Button, Power Button
- Video Out: 16b HDMI, 1280x1024 (MAX), 1024x768, 1280x720, 1440x900 w/EDID Support
- Audio Via HDMI Interface, Stereo
- Expansion Connectors:
    - Power 5v, 3.3v, VDD_ADC(1.8v)
    - 3.3v I/O on all signals
    - McASP0, SPI1, I2C, GPIO(65), LCD, GPMC, MMC1, MMC2, 7 AIN(1.8V MAX), 4 Timers, 3 Serial Ports, CAN0, EHRPWM(0,2),XDMA Interrupt, Power button, Expansion Board ID (Up to 4 can be stacked)
- Weight 1.4 oz (39.80 grams)

Although the BBB seems to be up to the task of an ALE Modem/Controller, as noted previously, there are challenges to be dealt with as to it being a suitable host for MIL-STD serial tone modem use which are currently being investigated. If these challenges

can be over come, it would permit adding serial tone modem support for ALE follow-on within the BBB-ALE package.

It is my opinion that the BBB hits the ARM price point spot on target. However if it also provided on board High Definition Audio on-board and a dual core ARM and twice the Flash storage and RAM for another $50 total cost then we would really have the ARM board we need in support of BBB-ALE. However as that is not the case, we can only dream of future developments and see what can be achieved using the current BBB.

BBB Evaluation

Efforts to date have been performed using a BBB under Angstrom Linux along with a standard BBB Audio Cape provided for hardware modem development evaluation. Based on my testing to date, the unit holds promise of being up to hosting a Military serial tone modem. The use of Linux however appears to be the only suitable OS solution for the BBB and MIL-STD Data Modem support in a traditional non-GUI hardware modem only configuration.

However for ALE use with no integral serial tone data modem support, perhaps the use of Windows Embedded Compact 7 could be made. Then the BBB with a small 4.3" 480x272 dot 16 bit LCD graphics display (with touch screen option) via RS-232, I²C-BUS OR SPI-BUS could provide a stand alone portable ALE GUI solution with a stripped down GUI feature set similar to that of desktop MARS-ALE could be complemented. Such a port would not be as strait forward as with WES7, as WEC7 has a lineage of Windows CE vs. desktop Windows 7 as does WES7.

A complete BBB based basic ALE and or MIL-STD modem solution with cables may come in under or about $200USD without options. For ALE modem/controller use of a display and user interface is more of a requirement but can also be an option when remote controlled only. In a basic configuration ALE operation shall be under the control of dumb terminal or other software designed for ALE use, otherwise a front panel LCD control interface is required.

What is the difference between a Modem and a Modem/Controller?

A Modem/Controller differs from a Modem in that it also provides the ability either via its firmware or as a conduit under external application software to not only function as a radio modem but to also control other radio equipments, e.g. HF SSB transceiver.

There are many examples of HF modems and modem/controllers, aside from an ALE modem/controller, CCIR 493 SELCAL modem/controllers and models of SCS PACTOR modem/controllers are two common examples.

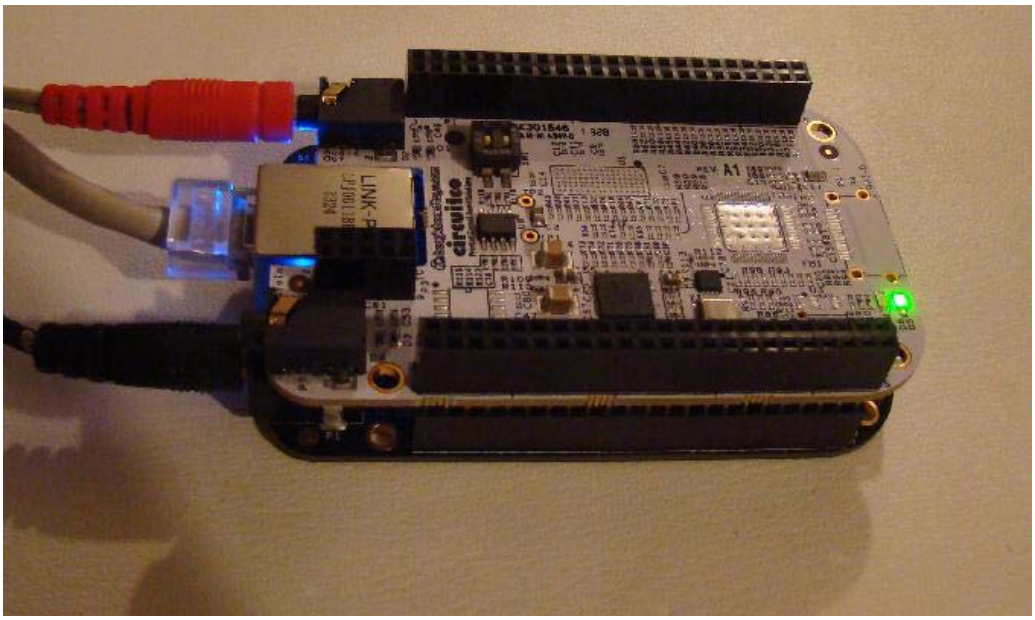
In addition, an HF modem that offers auxiliary UART, RS-232 or other I/O ports that support TX and RX could if an SDK was provided, be pressed into use as a modem/controller when reprogrammed for the role. I long ago was interested in the possibility of such reuse of the Kantronics KAM-XL as a potential ALE modem/controller.

Most of the information here would apply equally well to the BBB as a MIL-STD Data Modem only, or ALE modem/controller only or to an ALE modem/controller with MIL-STD Data Modem. The BBB as a host for ALE is pretty much a sure things, its ability to host Military serial tone modems is still to be determined.

BBB Evaluation to date

To date work on the BBB host has been in Linux where a STANAG 4285 modem test core (using the audio cape as seen below) which when tested on an x86 i5 core Linux laptop results in 5% or less CPU loading whereas on the BBB the same code built with a host of ARM compiler optimizations results in greater than 50% loading (down from 98% initially). 50% should be within the green zone or close for serial tone modem use as long as the CPU speed governor is always locked to the maximum speed, which it was not at first.

The next step was to be a Linux OS change to Debian Wheezy Hard that supports hardware floating point math to gauge improved loading over the shipped Angstrom Linux distribution which only supports software floating point operations. However an official Debian image based on Wheezy for the BBB has been released which supports Hard Floating-point and is now being evaluated.

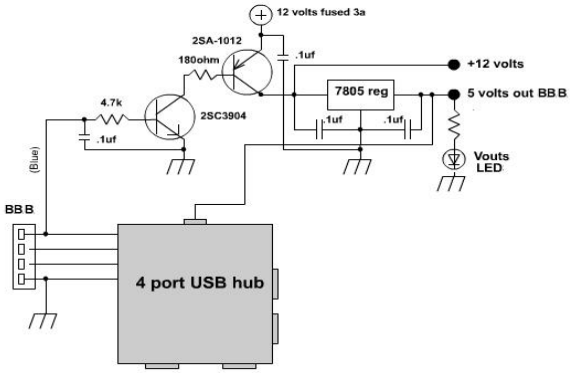


There is likely no question that the BBB is suitable for the basis of an ALE modem/controller as CPU loading will be much less for ALE. However it does however represent the minimum class of host hardware for the MARS member on a very small budget and one that likes the approach of roll your own packaging as to case etc. Should the BBB be up to the task of a software defined MIL-STD serial tone modem, then an integrated package with ALE and Data Modem for follow-on configuration can be developed for MARS use.

The standard BBB Audio Cape (the first generation of which I have, a second generation design has been released recently) has already been ruled out for use with the MIL-STD Data Modem effort, however it may be suitable for use with ALE as audio requirements are not as stringent for ALE. For BBB-ALE the end user may also be able to make use of USB audio devices such as the USB Signalink, either "AS IS" or repackaged as desired into the BBB-ALE case by the end user.

A third party Codec board will likely be required for MIL-STD Data Modem application and a potential one has already been identified but has yet to be sampled. The manufacturer may also enhance the codec board with a TXCO and it is estimated to cost about the same price as the standard BBB audio cape. Among other benefits of the 3rd party codec board is that it will free up the HDMI port for potential uses for video and perhaps audible user alarms for someone so configured as HDMI support must be disabled when the audio cape is used.

It has also been determined that the BBB requires a well regulated 5.1v D.C. source satisfying the overall current load and thus can easily be configured for 12v D.C. operation and long term battery powered operation. Any use of powered USB hubs should either be those that run off 12v or less as well, most are also 5.1v as is the BBB.



Make sure any use of a Hub is made by the Hub being plugged into the USB Host connector and powered on and that keyboard and mouse as well as USB audio device if being used are plugged in before powering on the BBB.

BBB-ALE for ARS

BBB-ALE will be made available to the Amateur Radio Service users as an ALE solution based on embedded Linux only. It will support an LCD/keypad interface and remote via

Ethernet Cross Cable or USB with external power or USB (miniUSB) client only if the computer provides enough current when any supported display is used.

The standard Audio Cape may be suitable for ALE, however its use would rule out the availability of the HDMI video port, thus an outboard USB Codec device of the I2C bus TXCO based codec mentioned earlier would be the choice for those users desiring HDMI support.

Basis of BBB-ALE on Linux

Linux based BBB-ALE as an ALE modem/controller shall be based on the existing 386EX/ADSP hosted DOS "C" language code project that was the basis for the MS-Windows PC-ALE. In addition to it being ported to Linux with its existing feature set, the project will be enhanced with a number of existing features from the current PC-ALE and MARS-ALE projects such the "Smart Scanning Calling", "2G Listen-Before-Transmit" and "AMD Retries" as well as developing features such as "Link Protection" (in the box GPS receiver option will be required) and "LQA In Call Individual Linking Call" being developed for MARS-ALE at the request of Army MARS.

BBB-ALE HF SSB Radio device(s) control will be handled by the user selection of a radio or other class of device driver. The model for the radio driver will be published and drivers for specific recommended make/models of radio devices will be provided as examples of each radio device class model. The ALE controller will be user selectable to either directly control radio devices or allow for remote control. The alternative remote radio device means shall be by device control event messages sent out the selected remote control port and a user provided terminal software performing the actual CAT control of the radio and optionally PTT as well any other radio device such as AATU. This support will be integrated into the Man Machine Interface (MMI) event message scripting language.

Development Progress

Where will my embedded efforts be at in 3 or 6 months or a year from now? I really don't know. I am working as hard and as fast as possible to learn all that is required, test configurations and further develop and refine my embedded OS vision and document it for the benefit of all MARS.

Although WES7 allows for a major amount of code reuse, I will need to develop source code for running the tools under WES7 on the GP PC hosts and especially on the embedded targeted Atom based hosts for both the stand alone terminal configuration and full embedded remote controlled focus with or without manual LCD/keypad user interface installed.

At the same time I am continuing to develop code for S4285 test project on the ARM host under Linux, enhance and develop more in the way of performance and features for desktop tools for general PC/OS use. However at some point down the road all that desktop development effort as to tools for MARS use will likely stop as running the code under those conditions provide less than optimal results.

Embedded Code Development Effort

Below is a breakdown by target host of the initial actual source development effort being taken and planned.

For details as they take place any MARS member can join the Cross Platform Embedded Development forum http://groups.yahoo.com/group/MARS_CPED/ where I post information on a fairly regular basis on my progress. For BBB-ALE and open to all Radio Amateurs, join: https://groups.yahoo.com/neo/groups/BeagleBoneBlack-ALE

Project Code Development

The BBB or any selected ARM based host as an ALE modem/controller shall be based on the existing 386EX/ADSP hosted DOS "C" language code project that was the basis for PC-ALE. In addition to being ported to Linux, the project will be enhanced with existing features from the current PC-ALE and MARS-ALE projects such the "Smart Scanning Calling", "2G Listen-Before-Transmit" and "AMD Retries" as well as developing features such as "Link Protection" (in the box GPS receiver option will be required) and "LQA In Call Individual Linking Call" being developed for MARS-ALE at the request of Army MARS.

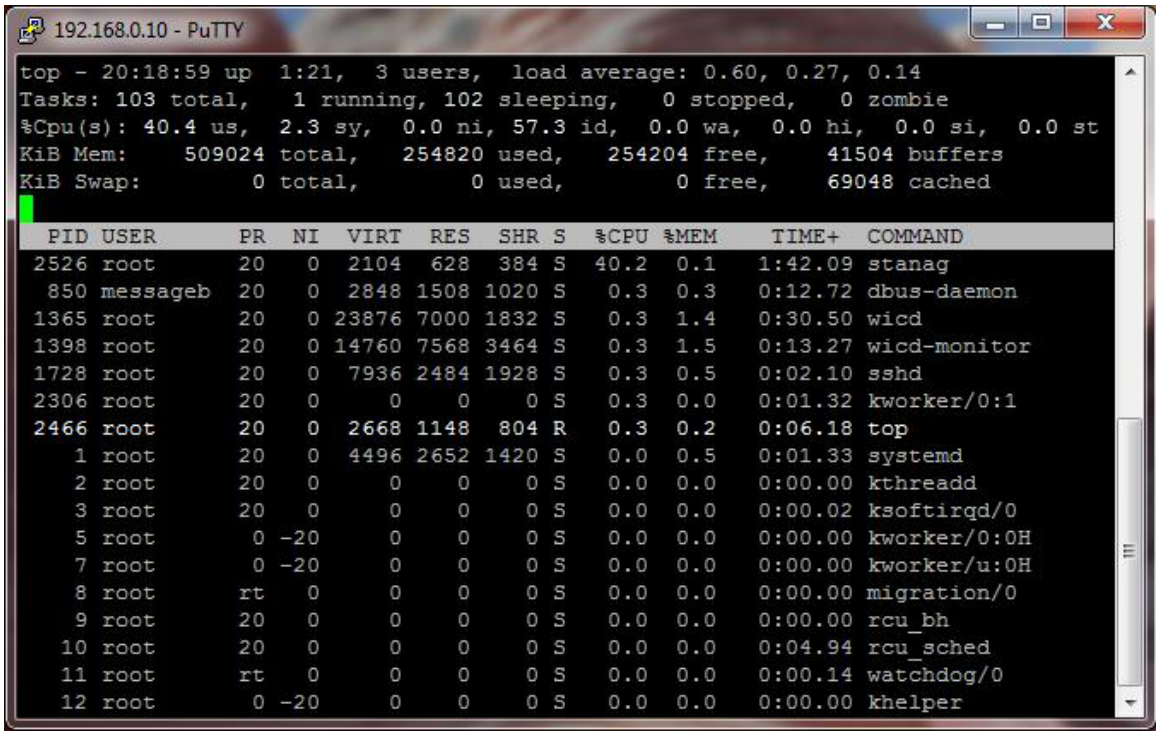
The initial BBB focus has been using the existing S4285 unproto modem core as ported from the x86 based Linux C code S4285/AX.25 terminal project baseline as modified where reducing the CPU loading been the goal. The initial effort was under Angstrom where running modem receiver and observing CPU loading from the DSP processor and working to mitigate that loading on the ARM processor has been the main effort, which have resulted in a reduction from 98% at first to using less than 60% with ARM specific compiler floating point math optimizations enabled and in the proper order as seen below:

gcc –Wall -O3 -mcpu=cortex-a8 -mfpu=neon -ftree-vectorize -mfloat-abi=softfp -ffast-math -fsingle-precision-constant control.c interleaver.c convolutional.c transmit.c general.c tables.c receive.c eqnew.c kalman.c demodulate.c io_queue.c psuedo.c frame.c crc.c -ostanag -lm

The following video is of decoding an S4285 ASYNC 300bps LONG transmission under Angstrom:

www.n2ckh.com/MARS_ALE_FORUM/BBB_S4285_300LONG_T2.wmv

The new Debian OS release that replaced Angstrom and its Soft Float, supports Hard float. The use oif hardware floating-point unit (FPU) support using the NEON SIMD architecture extensions with the proper compiler optimizations settings has so far resulted in a reduction of CPU loading to just over 40%, this was from 98%, then to 67% at first. Running the CPU with the Governor locked at a 1000Mhz clock and the use of all ARM hard float optimizations features that yield gains will be utilized to achieve the best performance. Below is a screen cap of the "top" utility displaying the "stanag" executable while hunting for S4285 preamble and the resulting CPU loading and memory usage.



Once a final determination regarding CPU loading using Hard vs. Soft float is made with the S4285 unproto modem, then a decision will made regarding porting the MS110A modem core written in C++ to the ARM and developing a project for testing it.

BBB-ALE Project Code Roots

The reuse of the existing x86 DOS based 386EX/ADSP project will accelerate the initial BBB-ALE development.

The reuse of code from the existing MARS-ALE and PC-ALE baselines for enhanced features shall also be made to meet design goals.

A number of existing features of the 386EX/ADSP project will require modification or elimination to take into account the selected ARM host, Debian OS and other hardware interfacing of the embedded environment.

The 386EX/ADSP project footprint is rather small, consisting of less than 60 computer software units (CSU's) of C language code and the supporting header files. A number of files will no longer be required, such as the interface between the ADSP and the Host processor.

The 386EX/ADSP project is a complete ALE modem/controller model which provides the structures to support the minimum FED-STD-1045A and MIL-STD-188-141A requirements and a number of optional features as well. For example, in addition to the mandatory Advanced Message Display (AMD) protocol for messaging, the optional Data Terminal (DTM) and Data Binary Message (DBM) protocols for both ASCII and

BINARY messaging are supported and both Broadcast (BRD) and Automatic Request Query (ARQ) modes for DTM and DBM are provided.

The project provides ALE call types:

- Individual Calls
- AllCalls
- Selective AllCalls
- AnyCalls
- Selective AnyCalls

The ALE Application Interface (API) CSU is the highest level code of the system which processes all ALE events. The API module of the existing 386EX/ADSP project processes all ALE API events received by the ALE Kernel to call the appropriate ALE function in the ALE kernel where the existing API events are:

ALE_TIMER
ALE_CALL
ALE_CLEAR
ALE_GLOBAL_ALLCALL
ALE_SELECTIVE_ALLCALL
ALE_GLOBAL_ANYCALL
ALE_SELECTIVE_ANYCALL
ALE_DBM_MSG
NOTE: in conjunction with API_ASCII_DATA or API_BINARY_DATA via user selection for transmit and automatic detection on receive
ALE_DTM_MSG
NOTE: in conjunction with API_ASCII_DATA automatically
API_ASCII_DATA
API_BINARY_DATA
ALE_AMD_MSG

The ALE Timer support provides for:

DWELL_TIMER
TWT_TIMER
KEYDOWN_TIMER
WRTT_TIMER
WRT_TIMER
SWT_TIMER
TWRN_TIMER
TWAN_TIMER
TWA_TIMER
INITIATE_FRAME
INITIATE_CLEAR
INITIATE_POLL
INACTIVITY_TIMEOUT

STATE MACHINE POLL_TIMER
RADIO_TIMEOUT
LQA_AGE_TIMER
TRANSPARENT_ESCAPE_TIMER

The existing CSU interface between ALE Controller and Radio equipment however is very bare where the current model covers the basics with:

- Key the transmitter
- Un-Key the transmitter
- Read the status of the PTT line
- Mute the receiver
- Un-Mute the receiver
- Put the ATU in circuit
- Put the ATU out of circuit

In addition there is only support for one each particular model KENWOOD and ICOM radio with hard code baud rate parameters. There is support of CAT Frequency and Mode control using USB as the mode and no CAT PTT. Also there isn't actually any implementation of ATU support for direct control coded or for actually reading the status of the PTT line.

Basically this project was designed for a narrow HF SSB transceiver focus where at 4800 baud most any KENWOOD radio of the era would work as that was the only baud rate the manufacturer supported. However as to ICOM as only 19,200 baud coded the radios supported at that time were limited to the latest models. In addition there is no consideration given to the issue of PA Spectral Purity Filter Relays and ALE scanning.

The Ranked State Machine CSU handles handshakes and calls and calculates swt and twrn period and ranks stations by LQA, it is the Ranked State Machine that is the heart of the ALE controller wherein lies the Linked State Machine that is the heartbeat. Here is a high level view of this CSU:

```
/* Globals located in PPARSER.C */
extern      Call_type     incoming_call_type;
extern      int           lqa_response_required;

/* Globals located in RADIO.C */
extern      int  scan_in_progress;

/* Locals to the link state machine module */
Public Link_state link_state;
Public       int single_multi;
Private      int channels[MAX_CHANNELS];
Private      int channels_in_use;
Private      int rank_channel_offset;
```

```
Private      int size_of_group;/* Size of the outgoing group */
Private      int twan_flag;    /* Has TWAN matured */
Private      int frame_in_progress; /* Is a frame being received */
Private      Call_type outgoing_call_type; /* Type of Outgoing Call */
Private      char poll_address[ADDRESS_STRING_LENGTH];/* Station being polled*/

/********* Routines for keeping record of connected stations **************/

Private const Other_address huge *other_linked_address[MAX_NET_SIZE];
Private int check_linked_address(const char *address)
Private void add_linked_address(const char *address)
Private void delete_linked_address(const char *address)
Private void reset_linked_addresses(void)
Private int count_linked_addresses(void)
/* This routine is used when formatting an acknowledgement, it ensures   */
/* the acknowlegement is only sent to those addresses who have responded */
Private void load_linked_addresses(const char *address[], int *number)
Public void display_linked_addresses(void)
/** Actions on outgoing addresses **/
/** Outgoing addresses are those addresses we wish to link to **/
Private const char *out_address[MAX_NET_SIZE];
Private void reset_out_addresses(void)
Private void add_out_address(const char *address)
Private void add_out_addresses(const char *address[], int number)
/* * Delete an address from the list of outgoing addresses.  */
Private void delete_out_address(const char *address)
/** The following stations have not responded to the call. **/
/** Update their LQA to reflect this.  **/
Private void update_lqa_for_non_respondies(void)
Private void prepare_ack_addresses(void)
/** MOTD Processing **/
Private void process_any_motd(void)
/** Functions to calculate timer parameters **/
Private long calculate_twrt_period(void)
Private long calculate_twr_period(void)
Private long calculate_twan_period(void)
/** Setup the next channel for a call **/
Private int prepare_channel_for_sending(void)
Private int prepare_net_for_ranking(const Net_address *net, const char *member[])
/** Is a frame currently being received **/
Private void is_frame_in_progress(const Events event)
Private void stop_all_timers(void)
{
 /* Disable all possible timers */
 stop_timer(TWT_TIMER);
 stop_timer(KEYDOWN_TIMER);
```

```
 stop_timer(SWT_TIMER);
 stop_timer(TWAN_TIMER);
 stop_timer(TWRN_TIMER);
 stop_timer(WRT_TIMER);
 stop_timer(WRTT_TIMER);
 stop_timer(DWELL_TIMER);
}
/** Go back to scanning **/
Private void resume_scanning(void)
/* * Actions to take when re-entering linked state. * */
Private void enter_linked_state(void)
```

/** Link State Machine **/

The Link State Machine is the heartbeat of the ALE controller. Here is a pseudo code representation of the Linked State Machine flow:

```
switch(link_state)
{
  /* Scanning disabled */
  case SCANNING_STOPPED:
    switch(event)
    {
      case START_SCAN:
        single_multi = MULTI_CHANNEL_OPERATION;
        enable_scanning();
        resume_scanning();
        link_state   = SCANNING;
        break;
      case INITIATE_FRAME:
        /* Start of a call                    */
        reset_linked_addresses();
        /* Start the listen before transmit timer    */
        /* Should be different for different modes   */
        start_timer(TWT_TIMER,calculate_twt_period());
        /* Go to state where seeing if other traffic */
        link_state   = LISTEN_FOR_ACTIVITY_BEFORE_CALL;
        break;
      case STOP_SCAN:
        /* This will cause an update to the display */
        break;
      default:
        display_bar  = CLEAR;
        break;
    }
    break;
  /* Scanning through the group of channels looking for ALE activity    */
  case SCANNING:
   switch(event)
   {
     case DWELL_TIMER:
       scan_radio();
       start_timer(DWELL_TIMER,get_dwell_period());
       reset_parser();
       driver_word_resync();
```

```
  single_multi = MULTI_CHANNEL_OPERATION;
  break;
 case DCD_DETECTED:
  /* Stop scanning and listen for long enough to detect  */
  /* a digital DCD. Only do this if no scan command is   */
  /* in progress otherwise it will stop on the wrong     */
  /* channel.                                            */
  if(scan_in_progress != SET)
  {
   link_state = PAUSE_SCAN_TO_LISTEN_FOR_CALL;
   start_timer(DWELL_TIMER,DCD_PAUSE_PERIOD);
  }
  break;
 case RX_FRAME_BEGIN:
  /* Start of a new frame must listen to see for me */
  stop_timer(DWELL_TIMER);
  link_state = PAUSE_SCAN_TO_LISTEN_FOR_CALL;
  break;
 case INITIATE_FRAME:
  /* Start of a call */
  reset_linked_addresses();
  /* Start the listen before transmit timer    */
  /* Should be different for different modes   */
  start_timer(TWT_TIMER,calculate_twt_period());
  /* Go to state where seeing if other traffic */
  link_state = LISTEN_FOR_ACTIVITY_BEFORE_CALL;
  break;
 case INITIATE_ALLCALL:
   outgoing_call_type = ALL_CALL;
   reset_linked_addresses();
   /* Start the listen before transmit timer    */
   /* Should be different for different modes   */
   start_timer(TWT_TIMER,calculate_twt_period());
   /* Go to state where seeing if other traffic */
   link_state = LISTEN_FOR_ACTIVITY_BEFORE_CALL;
   break;
 case INITIATE_ANYCALL:
   outgoing_call_type = ANY_CALL;
   reset_linked_addresses();
   /* Start the listen before transmit timer    */
   /* Should be different for different modes   */
   start_timer(TWT_TIMER,calculate_twt_period());
   /* Go to state where seeing if other traffic */
   link_state = LISTEN_FOR_ACTIVITY_BEFORE_CALL;
   break;
 case INITIATE_POLL:
   outgoing_call_type = POLL_CALL;
   reset_linked_addresses();
   /* Start the listen before transmit timer    */
   /* Should be different for different modes   */
   start_timer(TWT_TIMER,calculate_twt_period());
   /* Go to state where seeing if other traffic */
   link_state = LISTEN_FOR_ACTIVITY_BEFORE_CALL;
   break;
 case INITIATE_THIS_WAS_SOUND:
   /* Check for traffic before sounding */
```

```
      start_timer(TWT_TIMER,calculate_twt_period());
      link_state = LISTEN_BEFORE_THIS_WAS_SOUND;
      break;
    case INITIATE_THIS_IS_SOUND:
      /* Check for traffic before sounding */
      start_timer(TWT_TIMER,calculate_twt_period());
      link_state = LISTEN_BEFORE_THIS_IS_SOUND;
      break;
    case STOP_SCAN:
      db_disable_scanning();
      single_multi = SINGLE_CHANNEL_OPERATION;
      link_state   = SCANNING_STOPPED;
      break;
    case START_SCAN:
      db_enable_scanning();
      resume_scanning();
      break;
    default:
      display_bar = CLEAR;
          break;
 }
 break;
/* While scanning a DCD was heard. We have stopped on the channel to see */
/* If the call is destined for us.                                       */
case PAUSE_SCAN_TO_LISTEN_FOR_CALL:
  switch(event)
  {
    case DWELL_TIMER:
      /* Go back to scanning */
      scan_radio();
          link_state = SCANNING;
          start_timer(DWELL_TIMER,get_dwell_period());
      reset_parser();
      driver_word_resync();
          break;
    case RX_FRAME_BEGIN:
          /* Start of a new frame must listen to see for me */
          stop_timer(DWELL_TIMER);
      display_bar = CLEAR;
          break;
    case RX_FRAME_END:
          /* The frame can't have been for us */
      resume_scanning();
          break;
    case FRAME_FOR_ME:
          /* We are being called set up addresses */
          update_terminating_address(current_own_address->address);
      lead[0]=current_other_address->address;
          update_leading_addresses(lead,SINGLE_ADDRESS);
          set_termination_this_is();
          /* Stop timeouts */
      stop_all_timers();
          /* Goto transmit if individual call */
          if(incoming_call_type == INDIVIDUAL_CALL)
          {
            /* send response  */
```

```
        process_any_motd();
        goto_transmit();
            start_timer(KEYDOWN_TIMER,get_pttd_period());
            link_state = IND_WAIT_COMPLETION_OF_RESPONSE;
     }
          else
          {
        /* Net or group call */
            start_timer(SWT_TIMER,swt_get_swt_period());
            start_timer(TWAN_TIMER,calculate_twan_period());
        link_state = NET_WAIT_COMPLETION_OF_RESPONSE;
     }
          break;
   case FRAME_ALLCALL:
    /* Go into the linked state do not respond */
    add_linked_address(current_other_address->address);
    display_linked_to(current_other_address->address);
    equip_set_linked();
    enter_linked_state();
    break;
   case FRAME_ANYCALL:
    /* Update own address to respond with          */
    update_terminating_address(current_own_address->address);
    /* Update other address to respond to          */
    lead[0]=current_other_address->address;
    update_leading_addresses(lead,SINGLE_ADDRESS);
    incoming_call_type = ANY_CALL;
    /* Respond as a group or net call using PRN Slot */
    start_timer(SWT_TIMER,swt_get_swt_period());
    start_timer(TWAN_TIMER,calculate_twan_period());
     /* Stay linked after response */
     set_termination_this_is();
     /* The response will be sent after SWT matures   */
     link_state = NET_WAIT_COMPLETION_OF_RESPONSE;
     break;
   case FRAME_NOT_FOR_ME:
    /* Not being called go back to scanning */
    resume_scanning();
    break;
   case STOP_SCAN:
     db_disable_scanning();
     single_multi = SINGLE_CHANNEL_OPERATION;
     link_state = SCANNING_STOPPED;
     break;
    case START_SCAN:
      enable_scanning();
      resume_scanning();
      break;
     default:
      display_bar = CLEAR;
      break;
   }
  break;
 case LISTEN_FOR_ACTIVITY_BEFORE_CALL:
    switch(event)
    {
```

```
case DCD_DETECTED:
  break;
case LBT_DETECTED:
  /* Only consider detection if threshold reached */
case RX_FRAME_BEGIN:
  /* Someone else is using the channel try another */
  stop_timer(TWT_TIMER);
  if(prepare_channel_for_sending()==ROK)
  {
    /* We have not exhausted the pool yet       */
    /* Start the listening timer                */
    /* Should be different for different modes   */
    start_timer(TWT_TIMER,calculate_twt_period());
    /* Go to state where seeing if other traffic */
    link_state = LISTEN_FOR_ACTIVITY_BEFORE_CALL;
  }
  else
  {
    /* Pool is empty alert operator and scan */
    display_call_failed();
    resume_scanning();
  }
  break;
case TWT_TIMER:
  /* Channel clear goto transmit */
  set_termination_this_is();
  goto_transmit();
  start_timer(KEYDOWN_TIMER,get_pttd_period());
  switch(outgoing_call_type)
  {
    case POLL_CALL:
    case INDIVIDUAL_CALL:
      link_state = IND_WAIT_COMPLETION_OF_CALL;
      break;
    case GROUP_CALL:
    case NETWORK_CALL:
    case ANY_CALL:
    case ALL_CALL:
      link_state = NET_WAIT_COMPLETION_OF_CALL;
      break;
    default:
      break;
  }
  break;
default:
  display_bar = CLEAR;
  break;
        }
        break;
      case IND_WAIT_COMPLETION_OF_CALL:
        switch(event)
        {
      case TRANSMISSION_COMPLETE:
      /* Initial frame has been sent */
      goto_receive();
      /* Start the response and tune timer */
```

```
        start_timer(WRTT_TIMER,calculate_twrt_period());
        link_state = IND_WAIT_RESPONSE;
        break;
    case KEYDOWN_TIMER:
     /* Transmitter keyed start sending calling frame */
     if(single_multi == MULTI_CHANNEL_OPERATION)
     {
       /* Send a scanning section */
       send_call_frame();
     }
     else
     {
       /* read length of scanning section from database */
       send_sclc_call_frame();
     }
     display_bar = CLEAR;
     break;
    default:
      display_bar = CLEAR;
      break;
  }
   break;
case IND_WAIT_COMPLETION_OF_RESPONSE:
   switch(event)
   {
       case TRANSMISSION_COMPLETE:
         /* Response frame has been sent listen for ack */
         goto_receive();
         /* Start the response timer, twrt used instead of twr */
         start_timer(WRT_TIMER,calculate_twrt_period());
         link_state = IND_WAIT_ACKNOWLEDGEMENT;
         break;
       case KEYDOWN_TIMER:
         /* Send the response frame */
         send_resp_frame();
         display_bar = CLEAR;
         break;
       default:
         display_bar = CLEAR;
         break;
    }
    break;
case IND_WAIT_COMPLETION_OF_ACKNOWLEDGEMENT:
  switch(event)
  {
      case KEYDOWN_TIMER:
          /* Transmitter ramped to full power sent ack */
         send_ack_frame();
         display_bar = CLEAR;
         break;
       case TRANSMISSION_COMPLETE:
         /* We are now linked */
         goto_receive();
         equip_set_linked();
         enter_linked_state();
         next_frame_to_send();
```

```
    break;
  default:
     display_bar = CLEAR;
     break;
  }
  break;
case IND_WAIT_ACKNOWLEDGEMENT:
   switch(event)
   {
      case DCD_DETECTED:
        /* We maybe about to hear the acknowledgement         */
        /* Start the WRT  timer with a value of 784 ms 2 Trw   */
        /* We must hear the start of the response frame before */
        /* this matures.                                      */
        start_timer(WRT_TIMER,784);
        break;
     case FRAME_FOR_ME:
        /* We are now connected */
        add_linked_address(current_other_address->address);
        display_linked_to(current_other_address->address);
        equip_set_linked();
        enter_linked_state();
        next_frame_to_send();
        break;
     case WRT_TIMER:
        /* The incomming call has failed */
        resume_scanning();
       break;
    case FRAME_NOT_FOR_ME:
    case RX_FRAME_END:
       /* The frame can't have been for us */
       resume_scanning();
       break;
    default:
       display_bar = CLEAR;
       break;
   }
   break;
case IND_WAIT_RESPONSE:
   switch(event)
   {
     case DCD_DETECTED:
        /* We maybe about to hear the response                */
        /* Start the WRTT timer with a value of 784 ms 2 Trw   */
        /* We must hear the start of the response frame before */
        /* this matures.                                      */
          start_timer(WRTT_TIMER,calculate_twrt_period());
          break;
    case RX_FRAME_BEGIN:
      /* Frame has started stop the timer */
      stop_timer(WRTT_TIMER);
      break;
    case FRAME_FOR_ME:
      /* Response has been heard send acknowledgement */
      if(outgoing_call_type == POLL_CALL)
      {
```

```
                /* We are polling so send a this was */
                /* Send last LQA command with no ack */
                finish_poll();
                set_termination_this_was();
                goto_transmit();
                display_polled_with(current_other_address->address);
                start_timer(KEYDOWN_TIMER,get_pttd_period());
                link_state = LINKED_BRINGING_DOWN_LINK;
            }
            else
            {
              /* Remove it from the pending list */
              delete_out_address(current_other_address->address);
              /* Add it to the received list */
              add_linked_address(current_other_address->address);
              goto_transmit();
              start_timer(KEYDOWN_TIMER,get_pttd_period());
              display_linked_to(current_other_address->address);
              link_state = IND_WAIT_COMPLETION_OF_ACKNOWLEDGEMENT;
            }
            break;
          case FRAME_NOT_FOR_ME:
            /* Not for us so try again */
          case WRTT_TIMER:
            /* Call has failed try again on another channel */
            /* If we are only on a single channel give up   */
            update_lqa_for_non_respondies();

if((prepare_channel_for_sending()==ROK)&&(single_multi==MULTI_CHANNEL_OPERATION))
            {
               /* We have not exhausted the pool yet      */
               /* Start the listening timer               */
               /* Should be different for different modes */
               start_timer(TWT_TIMER,calculate_twt_period());
               /* Go to state where seeing if other traffic */
               link_state = LISTEN_FOR_ACTIVITY_BEFORE_CALL;
            }
            else
            {
               /* Pool is empty alert operator and scan */
              display_call_failed();
               resume_scanning();
             }
            break;
          default:
            display_bar = CLEAR;
              break;
          }
              break;
          case LINKED_IDLE:
              switch(event)
              {
                  case FRAME_FOR_ME:
                     /* Start of a 3-way handshake */
                   switch(incoming_call_type)
                   {
```

```
                case INDIVIDUAL_CALL:
                   goto_transmit();
                   start_timer(KEYDOWN_TIMER,get_pttd_period());
                   link_state = IND_LINKED_WAIT_COMPLETION_OF_RESPONSE;
                   break;
                case NETWORK_CALL:
                case GROUP_CALL:
                    start_timer(SWT_TIMER,swt_get_swt_period());
                    start_timer(TWAN_TIMER,calculate_twan_period());
                    link_state = NET_LINKED_WAIT_COMPLETION_OF_RESPONSE;
                    break;
                default:
                    display_bar = CLEAR;
                break;
               }
               break;
             case FRAME_FOR_ME_CLEAR:
                delete_linked_address(current_other_address->address);
                display_cleared_with(current_other_address->address);
                if(count_linked_addresses() == 0 )
                {
                  /* Only resume scan if no linked stations left */
                  equip_clear_linked();
                  resume_scanning();
                }
                break;
                     case INITIATE_FRAME:
                       /* Start a new 3-way handshake */
       if(frame_in_progress == CLEAR)
       {
        /* Only if the channel is free otherwise back off */
                        set_termination_this_is();
                        goto_transmit();
                        start_timer(KEYDOWN_TIMER,get_pttd_period());
                        switch(outgoing_call_type)
        {
          case ALL_CALL:
          case INDIVIDUAL_CALL:
                                  link_state                                    =
IND_LINKED_WAIT_COMPLETION_OF_INITIAL_FRAME;
                           break;
                          case NETWORK_CALL:
          case GROUP_CALL:
          case ANY_CALL:
                                  link_state                                    =
NET_LINKED_WAIT_COMPLETION_OF_INITIAL_FRAME;
                           break;
          default:
            dprintf("Unknown call type while linked\n\r");
            display_bar = CLEAR;
          break;
        }
       }
       else
       {
        start_timer(INITIATE_FRAME,1000);
```

```
 }
                  break;
                case INITIATE_CLEAR:
 /* Send a This was frame */
 if(frame_in_progress == CLEAR)
 {
  /* Only if the channel is free otherwise back off */
                   /* Goto transmit */
                   set_termination_this_was();
                   goto_transmit();
                   start_timer(KEYDOWN_TIMER,get_pttd_period());
                   link_state = LINKED_BRINGING_DOWN_LINK;
 }
 else
 {
  start_timer(INITIATE_CLEAR,1000);
 }
                  break;
case TWA_TIMER:
 /* Timed out due to inactivity */
 reset_linked_addresses();
 equip_clear_linked();
 resume_scanning();
 break;
case SM_POLL_TIMER:
 /* See if anything is waiting */
 next_frame_to_send();
 start_timer(SM_POLL_TIMER,1000);
 break;
default:
 display_bar = CLEAR;
                  break;
       }
       break;
      case IND_LINKED_WAIT_COMPLETION_OF_INITIAL_FRAME:
       switch(event)
       {
                case KEYDOWN_TIMER:
                 /* Transmitter now ready */
                 /* Send call          */
                 send_init_frame();
 display_bar = CLEAR;
                  break;
                case TRANSMISSION_COMPLETE:
                 /* Initial frame has been sent */
                 goto_receive();
                 switch(outgoing_call_type)
 {
  case ALL_CALL:
   /* No response expected */
                    enter_linked_state();
   /* See if we have anything pending transmission */
   next_frame_to_send();
   break;
  case INDIVIDUAL_CALL:
   /* Wait for the response */
```

```
                    start_timer(WRT_TIMER,calculate_twr_period());
                    link_state = IND_LINKED_WAIT_RESPONSE;
                    break;
  default:
    display_bar = CLEAR;
    break;
}
                  break;
                default:
display_bar = CLEAR;
                  break;
       }
       break;
     case IND_LINKED_WAIT_COMPLETION_OF_ACKNOWLEDGEMENT:
       switch(event)
       {
                case KEYDOWN_TIMER:
                  /* Now ready to transmit */
                  /* Send acknowledgement */
                  send_ack_frame();
display_bar = CLEAR;
                  break;
                case TRANSMISSION_COMPLETE:
                  /* Acknowledgement sent go back to idle */
                  goto_receive();
equip_set_linked();
                  enter_linked_state();
                  break;
                default:
display_bar = CLEAR;
                  break;
       }
       break;
     case IND_LINKED_WAIT_COMPLETION_OF_RESPONSE:
       switch(event)
       {
                case KEYDOWN_TIMER:
/* Now transmitting send response frame */
                  send_resp_frame();
display_bar = CLEAR;
                  break;
                case TRANSMISSION_COMPLETE:
                  /* Response has been sent wait for acknowlegement */
                  goto_receive();
                  start_timer(WRT_TIMER,calculate_twr_period());
                  link_state = IND_LINKED_WAIT_ACKNOWLEDGEMENT;
                  break;
                default:
display_bar = CLEAR;
                  break;
       }
       break;
     case IND_LINKED_WAIT_RESPONSE:
       switch(event)
       {
                case DCD_DETECTED:
```

```
                    /* We maybe about to hear the response              */
                    /* Start the WRT timer with a value of 784 ms 2 Trw    */
                    /* We must hear the start of the response frame before */
                    /* this matures.                                    */
        stop_timer(WRT_TIMER);
                    start_timer(WRT_TIMER,784);
                    break;
                   case RX_FRAME_BEGIN:
                    /* Frame has started stop the timer */
                    stop_timer(WRT_TIMER);
                    break;
                   case FRAME_FOR_ME:
                    /* We have heard the response now send acknowledgement */
                    goto_transmit();
                    start_timer(KEYDOWN_TIMER,get_pttd_period());
                    link_state                                                    =
IND_LINKED_WAIT_COMPLETION_OF_ACKNOWLEDGEMENT;
                    break;
                   case FRAME_NOT_FOR_ME:
                    /* Handshake has failed go back to idle */
        display_handshake_failed();
                    enter_linked_state();
                    break;
                   case WRT_TIMER:
                    /* Hanshake has failed go back to idle */
        display_handshake_failed();
                    enter_linked_state();
                    break;
                  default:
             display_bar = CLEAR;
        break;
            }
            break;
           case IND_LINKED_WAIT_ACKNOWLEDGEMENT:
            switch(event)
            {
                   case DCD_DETECTED:
                    /* We maybe about to hear the response              */
                    /* Start the WRT timer with a value of 784 ms 2 Trw    */
                    /* We must hear the start of the response frame before */
                    /* this matures.                                    */
                    start_timer(WRT_TIMER,784);
                    break;
                   case RX_FRAME_BEGIN:
                    /* Frame has started stop the timer */
                    stop_timer(WRT_TIMER);
                    break;
                   case FRAME_FOR_ME:
                    /* Handshake has succeded */
                    enter_linked_state();
                    break;
                   case FRAME_NOT_FOR_ME:
                    /* Handshake has failed */
                    enter_linked_state();
                    break;
                   case WRT_TIMER:
```

```
                    /* Handshake has failed */
                    enter_linked_state();
                    break;
                  default:
        display_bar = CLEAR;
                    break;
          }
          break;
         case LINKED_BRINGING_DOWN_LINK:
          switch(event)
          {
                  case KEYDOWN_TIMER:
                    /* Send this was frame */
                    send_init_frame();
        display_bar = CLEAR;
                    break;
                  case TRANSMISSION_COMPLETE:
                    /* TWAS sent go back to scanning */
                    goto_receive();
#ifdef __TRACING__
                    dprintf(" ADDS left %d \n",count_linked_addresses());
#endif
                    if(count_linked_addresses() == 0)
                    {
          /* All addresses have now cleared resume scan */
          equip_clear_linked();
          resume_scanning();
                    }
                    else
                    {
          /* Still some linked addresses */
          enter_linked_state();
                    }
                    break;
                  default:
        display_bar = CLEAR;
                    break;
          }
          break;
         case NET_WAIT_COMPLETION_OF_CALL:
          switch(event)
          {
                  case KEYDOWN_TIMER:
                    /* We are now transmitting, send the calling frame */
        if(single_multi == MULTI_CHANNEL_OPERATION)
        {
          /* Send a scanning section */
                      send_call_frame();
        }
        else
        {
          /* read length of scanning section from database */
          send_sclc_call_frame();
        }
        display_bar = CLEAR;
                    break;
```

```
                case TRANSMISSION_COMPLETE:
                  /* The initial frame has been sent */
                  goto_receive();
  switch(outgoing_call_type)
  {
   case ALL_CALL:
    /* No one will respond so go to linked */
                    display_linked_to("ALLCALL ?");
    equip_set_linked();
    enter_linked_state();
    break;
                  case NETWORK_CALL:
   case ANY_CALL:
                    start_timer(TWRN_TIMER,twrn_get_twrn_period());
                    link_state = NET_WAIT_RESPONSE;
                    break;
   default:
    break;
  }
                 break;
                default:
  display_bar = CLEAR;
                 break;
       }
       break;
      case NET_WAIT_COMPLETION_OF_RESPONSE:
       switch(event)
       {
 case SWT_TIMER:
                 /* Call must be a group or net */
                 goto_transmit();
                 start_timer(KEYDOWN_TIMER,get_pttd_period());
                 break;
                case KEYDOWN_TIMER:
                 /* Send the slotted response */
                 send_resp_frame();
  display_bar = CLEAR;
                 break;
                case TRANSMISSION_COMPLETE:
                 /* Response has completed */
                 goto_receive();
                 /* Acknowledgement timer is already running */
                 link_state = NET_WAIT_ACKNOWLEDGEMENT;
                 break;
                case TWAN_TIMER:
                 /* 3 way handshake has not completed */
  resume_scanning();
                 link_state = SCANNING;
                 break;
                default:
  display_bar = CLEAR;
                 break;
      }
break;
      case NET_WAIT_COMPLETION_OF_ACKNOWLEDGEMENT:
       switch(event)
```

```
      {
                case KEYDOWN_TIMER:
                  /* now transmitting send the frame */
                  send_ack_frame();
 display_bar = CLEAR;
                  break;
                case TRANSMISSION_COMPLETE:
                  /* The hanshake has completed */
                  goto_receive();
 equip_set_linked();
                  enter_linked_state();
                  break;
 display_bar = CLEAR;
                default:
                  break;
      }
      break;
     case NET_WAIT_ACKNOWLEDGEMENT:
      switch(event)
      {
                case TWAN_TIMER:
 if(frame_in_progress == SET)
 {
                    /* What until the end of the frame to decide if ACK heard */
  twan_flag = SET;
 }
 else
 {
  /* No ACK heard */
  resume_scanning();
 }
                  break;
case FRAME_NOT_FOR_ME:
case RX_FRAME_END:
 if(twan_flag == SET)
 {
  /* TWAN has matured and the frame was not the ACK */
                   resume_scanning();
 }
 /* Otherwise its a response from someone else */
 break;
                case FRAME_FOR_ME:
                  /* We are now linked to the NET or GROUP */
                  add_linked_address(current_other_address->address);
                  display_linked_to(current_other_address->address);
 equip_set_linked();
 stop_all_timers();
                  enter_linked_state();
                  break;
                default:
 display_bar = CLEAR;
                  break;
      }
      break;
     case NET_WAIT_RESPONSE:
      switch(event)
```

```
        {
                case FRAME_FOR_ME:
                  /* A call has come in */
      /* Remove it from pending receipt store */
      delete_out_address(current_other_address->address);
      /* Add it to the linked store */
                  add_linked_address(current_other_address->address);
                  display_linked_to(current_other_address->address);
                  break;
                case TWRN_TIMER:
                  /* Time is up send acknowledgement if required */
                  if(count_linked_addresses() != 0)
                  {
       update_lqa_for_non_respondies();
       /* Only acknowledge those addresses that responded */
       prepare_ack_addresses();
                        goto_transmit();
                        start_timer(KEYDOWN_TIMER,get_pttd_period());
                        link_state                                                    =
NET_WAIT_COMPLETION_OF_ACKNOWLEDGEMENT;
                  }
                  else
                  {
       if(outgoing_call_type != ANY_CALL)
       {
         /* No responses heard try again */
                        if(prepare_channel_for_sending()==ROK)
                        {
                          /* We have not exhausted the pool yet */
                          /* Start the listening timer */
                          /* Should be different for different modes */
                          start_timer(TWT_TIMER,calculate_twt_period());
                          /* Go to state where seeing if other traffic */
                          link_state = LISTEN_FOR_ACTIVITY_BEFORE_CALL;
        }
                        else
                        {
                          /* Pool is empty alert operator and scan */
         update_lqa_for_non_respondies();
                          display_call_failed();
         resume_scanning();
        }
       }
       else
       {
         /* No RESP to ANYCALL, only try on 1 channel so give up */
                        display_call_failed();
        resume_scanning();
       }
                  }
                  break;
                default:
      display_bar = CLEAR;
                  break;
        }
        break;
```

```
      case NET_LINKED_WAIT_COMPLETION_OF_INITIAL_FRAME:
       switch(event)
       {
                case KEYDOWN_TIMER:
  /* Now trasmitting so send initial frame */
                 send_init_frame();
  display_bar = CLEAR;
                 break;
                case TRANSMISSION_COMPLETE:
                 goto_receive();
                 start_timer(TWRN_TIMER,twrn_get_twrn_period());
  link_state = NET_LINKED_WAIT_RESPONSE;
                 break;
                default:
  display_bar = CLEAR;
                 break;
       }
       break;
      case NET_LINKED_WAIT_COMPLETION_OF_ACKNOWLEDGEMENT:
       switch(event)
       {
                case KEYDOWN_TIMER:
 /* Now transmitting so send ACK frame */
                 send_ack_frame();
  display_bar = CLEAR;
                 break;
                case TRANSMISSION_COMPLETE:
                 goto_receive();
                 enter_linked_state();
                 break;
                default:
  display_bar = CLEAR;
                 break;
       }
       break;
      case NET_LINKED_WAIT_COMPLETION_OF_RESPONSE:
       switch(event)
       {
                case SWT_TIMER:
                 /* prepare to send the response */
  goto_transmit();
                 start_timer(KEYDOWN_TIMER,get_pttd_period());
                 break;
                case KEYDOWN_TIMER:
 /* Now transmitting so send response frame */
                 send_resp_frame();
  display_bar = CLEAR;
                 break;
                case TRANSMISSION_COMPLETE:
                 goto_receive();
                 link_state=NET_LINKED_WAIT_ACKNOWLEDGEMENT;
                 break;
                default:
  display_bar = CLEAR;
                 break;
       }
```

```
            break;
           case NET_LINKED_WAIT_RESPONSE:
            switch(event)
            {
                    case TWRN_TIMER:
                     /* Responses should have come in by now */
                     goto_transmit();
                     start_timer(KEYDOWN_TIMER,get_pttd_period());
                     link_state                                                    =
NET_LINKED_WAIT_COMPLETION_OF_ACKNOWLEDGEMENT;
                     break;
                    case FRAME_FOR_ME:
                     /* Response has been heard */
                     display_response_from(current_other_address->address);
                     break;
                    default:
                     break;
            }
            break;
           case NET_LINKED_WAIT_ACKNOWLEDGEMENT:
            switch(event)
            {
                    case TWAN_TIMER:
       if(frame_in_progress == SET)
       {
                      /* What until the end of the frame to decide if ACK heard */
        twan_flag = SET;
       }
       else
       {
        /* No ACK heard */
        display_handshake_failed();
                       enter_linked_state();
       }
                    break;
      case FRAME_NOT_FOR_ME:
      case RX_FRAME_END:
       if(twan_flag == SET)
       {
        /* No ACK heard and TWAN has matured */
        display_handshake_failed();
                       enter_linked_state();
       }
       /* Otherwise its a response from someone else, so wait for ACK */
       break;
      case FRAME_FOR_ME:
                       /* ACK  has come in, go back to LINKED IDLE */
       stop_all_timers();
                    enter_linked_state();
                    break;
      default:
       display_bar = CLEAR;
                    break;
           }
           break;
    case LISTEN_BEFORE_THIS_WAS_SOUND:
```

```
 switch(event)
 {
  case TWT_TIMER:
   /* Commence the sound */
   goto_transmit();
   start_timer(KEYDOWN_TIMER,get_pttd_period());
   link_state = THIS_WAS_SOUND_IN_PROGRESS;
   display_bar = CLEAR;
   break;
                case RX_FRAME_BEGIN:
   /* Abandon the Sound as other activity on the channel */
   resume_scanning();
   break;
  default:
   display_bar = CLEAR;
   break;
 }
 break;
case LISTEN_BEFORE_THIS_IS_SOUND:
 switch(event)
 {
  case TWT_TIMER:
   /* Commence the sound */
   goto_transmit();
                        start_timer(KEYDOWN_TIMER,get_pttd_period());
   link_state = THIS_IS_SOUND_IN_PROGRESS;
   display_bar = CLEAR;
   break;
  case RX_FRAME_BEGIN:
   /* Abandon the Sound as other activity on the channel */
   resume_scanning();
   break;
  default:
   display_bar = CLEAR;
   break;
 }
 break;
case THIS_WAS_SOUND_IN_PROGRESS:
 switch(event)
 {
  case KEYDOWN_TIMER:
   /* Send the sound */
   send_this_was_sound();
   display_bar = CLEAR;
   break;
  case TRANSMISSION_COMPLETE:
   /* Return to scanning */
   resume_scanning();
   break;
  default:
   display_bar = CLEAR;
   break;
 }
 break;
case THIS_IS_SOUND_IN_PROGRESS:
 switch(event)
```

```
    {
      case KEYDOWN_TIMER:
        /* Send the sound */
        send_this_is_sound();
        display_bar = CLEAR;
        break;
      case TRANSMISSION_COMPLETE:
                        /* Listen for 2 seconds before continuing to scan */
                        start_timer(DWELL_TIMER,2000);
                        link_state = PAUSE_SCAN_TO_LISTEN_FOR_CALL;
        break;
      default:
        display_bar = CLEAR;
        break;
   }
   break;
            default:
             break;
 }
 /* Update the status bar only when necessary */
 if(display_bar == SET)
 {
   display_status_bar();
 }
}

/** Call Proceedures **/
/*
 * Description: Initiate a new outgoing call.
 *
 * Parameters: others[] list of pointers of addresses to be called.
 *             nr_others number of addresses in the list.
 *             *own own address to be used when making the call.
 */
Public void make_link(const char *others[], int nr_others)
/** Clear Proceedures **/
/*
 * Description: Initiate a clear.
 *
 * Paramaters: *others[] list of pointers of addresses to be cleared.
 *             nr_others number of addresses in the list.
 */
Public void clear_link(const char *others[], int nr_others)
/** Linked Exchanges */
/*
 * Description: Queue an AMD message to be sent on next frame.
 * Parameters: *amd pointer to AMD message to be sent.
 */
Public void queue_amd(const char *amd,void (*func)(void *))
/*
 * Description: Initiate the sending of and AMD message.
```

```
 *          If no link present, link first.
 * Parameters:  char *others[], stations to whom the AMD is to be sent.
 *          int nr_others, number of stations in the group.
 *          char *own,     own address to be used in the frame.
 *          char *amd,     AMD message to be sent.
 */
Public void initiate_amd_exchange(const char *others[], int nr_others, const char *amd)
/** Request the station to poll the remote address. **/
/** This is used to fill in missing entries in the LQA database. **/
Public void initiate_poll(const char *address, int channel)
Private void finish_poll(void)
/** Emergency kill and return to scanning **/
/** Description: Kill the current link immediately. **/
Public void emergency_kill(void)
```

MAN MACHINE INTERFACE

The Man Machine Interface (MMI) CSU is a command interpreter that processes structured commands received over one of the selected remote control interfaces. MMI commands are simple ASCII character based and are not case sensitive.

The MMI command set shall have a limit of 500 commands where the initial MMI command words and their purpose in creating an MMI command string where one or a combination of two or more commands with or without arguments per line that currently exist are:

Command	**Purpose**
ADD	**= Meta command used with minor commands CHANNEL, GROUP, NET Address, OWN Address, OTHER Address**
AL0	**= Link Protection AL0 mode, the default**
ALLCALL	**= Global and Selective minor command for ALE AllCall call**
AMD	**= Advanced Message Display Minor command**

ANYCALL = Global and Selective minor command for ALE AnyCall call
ARQ = Minor command used with DBM and DTM
AUTO = Minor command used with DISABLE and ENABLE
BELLS = Minor command used with DISABLE and ENABLE
BROADCAST = Minor command used with DBM and DTM
CALL = Meta command used with <Other Address>
CHANNEL = Minor command used with Meta commands
CLEAR = Clear ALE inlink state.
CONFIG = Minor command used with SAVE and LOAD Meta commands.
DATABASE = Minor command used with SAVE and LOAD Meta commands.
DATE = Minor command used with SET
DBM = Data Binary Message binary mode direct command and Meta command when used with RETRIES.
DELETE = Meta command used with CHANNEL, GROUP, NET Address, OWN Address, OTHER Address
DISABLE = Meta command used with ALLCALL, ANYCALL, AUTO, BELLS, ECHO, GPS, LBT, LP, LQA, MOTD, POLL, SCAN, SOUND, THIS_IS, TRACE
DISPLAY = Meta command used with AMD, CHANNEL, HELP, LINKED, KEY, SELF, MYADDRESS, NET_ADDRESS, OPTIONS, OTHER_ADDRESS, OWN_ADDRESS
DTM = Data Terminal Message ASCII mode direct command and Meta command when used with RETRIES.
DWELL = Minor command used with SET
ECHO = Minor command used with DISABLE and ENABLE to enable or disable Telnet echo of MMI commands
ENABLE = Meta command used with ALLCALL, ANYCALL, AUTO, BELLS, ECHO, GPS, LBT, LP, LQA, MOTD, POLL, SCAN, SOUND, THIS_IS, TRACE
END = Direct execute command
EXIT = Direct execute command
FERRORS = Minor command used with SET
FVOTES = Minor command used with SET
GLOBAL = Meta command used with ALLCALL and ANYCALL
GPS = Minor command used with DISABLE and ENABLE
GROUP = Minor command used with ADD and DELETE
HELP = Meta command used alone and with other MMI commands to display help on the command that follows on the same line. Also a Direct execute command when used without arguments.
INACTIVITY = Minor command used with SET
KEY = Minor command used with SET. Also a direct execute command when used alone.
KILL = Direct execute command
LBT = Listen Before Transmit, minor command used with DISABLE and ENABLE
LINKED = Direct execute command
LIST = Direct execute command
LOAD = Meta command used with CONFIG and DATABASE
LQA = Link Quality Analysis
MANUAL = Direct execute command
MODE = Minor command used with ARQ and BROADCAST in their use with DBM and DTM
MOTD = Message of the Day
MYADDRESS = Minor command used with SET
NET_ADDRESS = Minor command used with ADD
OPTIONS = Minor command used with RESET

OTHER_ADDRESS = Minor command used with ADD and RESET
OWN_ADDRESS = Minor command used with ADD and RESET
PTTD = Minor command used with SET
QUIT = Direct execute command
RADIO = Minor command used with SET, RESET
RESET = Meta command used with AMD, CHANNEL, DISPLAY, LQA, OPTIONS, OWN_ADDRESS, OTHER_ADDRESS, RADIO
RESOUND = Minor command used with SET
RETRIES = Minor command used with DBM and DTM
RXFREQ = Minor command used with SET
RXMODE = Minor command used with SET
SAVE = Meta command used with DATABASE
SCAN = Minor command used with SET
SCL = Minor command used with SET
SELF = Minor command used with SET
SELECTIVE = Meta command used with ALLCALL and ANYCALL
SERRORS = Minor command used with SET
SET = Meta command used with CHANNEL, DATE, DBM, DMT, FERRORS, FVOTES, GROUP, INACTIVITY, KEY, MYADDRESS, MOTD, PTTD, RADIO, RESOUND, RXMODE, RXFREQ, SCAN, SCL, SELF, SERRORS, SOUND, SVOTES, TIME, TXMODE, TXFREQ,
SOUND = Minor command used with SET
STOP = Direct execute command
START = Direct execute command
STATUS = Direct execute command
SVOTES = Minor command used with SET
THIS_IS = Minor command used with ENABLE and DISABLE
TIMEOUT = Minor command used with LQA
TRACE = Direct execute command
TXFREQ = Minor command used with SET
TXMODE = Minor command used with SET
UPLOAD = Direct execute command

MARS SOFTWARE VERSION RESERVED MMI COMMANDS:

AL1,AL2,AL3,AL4,LP,LQACALL,LQAPOLL

The Meta and Minor command status can best be explained by showcasing the "DELETE" command which is a Meta command which requires the use of Minor commands and arguments as follows:

DELETE CHANNEL [n] = Delete channel n where n is any existing Channel Number.
DELETE NET [address] = Delete NET address where address is any existing NET Address.
DELETE OTHER [address] = Delete OTHER address where address is any existing OTHER Address.
DELETE OWN [address] = Delete OWN address where address is any existing Self Address Deleting an existing. OWN Address does not shift the ordering of the remaining addresses in the database.
DELETE SELF [address] = Delete SELF address where address is any existing Self Address.

The maximum MMI command string length shall be programmed at 1000 characters, well in excess of what will ever be required.

The maximum number of words in an MMI command string shall be 20, also more than will likely ever be required.

In some cases, MMI commands can accept one or more arguments. For example the MMI AMD command:

AMD [address <address> " message text within DOUBLE quotes with spaces before 1st & after last word message "] > Send station @ address(es) 1..n an AMD message that is provided.

A BBB-ALE MMI user guide shall list the Command syntax and arguments structure for each command with some examples as to the use of each command. However for the most part, due to the power of the commands and all that can be done with them, its pretty much up to the end user as to their use stand alone and when combined in most cases when creating groups of commands.

For example, when MACRO supported is ported to BBB-ALE from MARS-ALE the user will be able to emulate hardware radio AMD store to memory message recall to send. An AMD message memory capability one can configured using the MACRO support and MMI commands by creating say AMDMSG1.MAC and AMDMSG2.MAC files in the MACRO sub directory using the MMI AMD command:

So to setup for sending a resupply with water and food message to BASE1 as the tactical callsign the MACRO the syntax would be:

AMD BASE1 " RESUPPLY WITH FOOD AND WATER ASAP BT AR "

This can currently be tested by a MARS-ALE or PC-ALE user where it has been ported to, by using the DataBar with the MMI box checked. At any time you want, sitting single channel or scanning just have the MMI box checked and in the DataBar enter: MACRO AMDMSG1 and hit Enter in MARS-ALE or in PC-ALE as Charles coded it when he cloned my MMI from the DataBar idea, click the SENDMSG button and you will link with the station and send the AMD. In MARS-ALE have AMD Retries setup as well for a better AMD pseudo ARQ experience.

In my plans for BBB-ALE this type of MMM/MACRO based AMD message storage for reuse will come in quite handy with the LCD/Keyboard interface and even a USB keyboard.

BBB-ALE Directory Structure

The key directories on the BBB that are familiar to any Linux user which BBB-ALE shall access during install, configuration or live operation:

- **bin** Contains the binary executables used by all of the users

- **boot** Contains the device tree binaries for the BBB
- **dev** Contains the device nodes (device drivers)
- **etc** Contains configuration files for the local system
- **home** The user home directories (e.g. */home/root* is the root user home)
- **lib** Contains all of the standard system libraries
- **media/mnt** Here we can mount a micro sd card by default
- **proc** Related to processes running on the BBB that come and go, for example if you go into */proc* and type cat iomem you can see the mapping addresses
- **run** Contains information about the running system since the last boot
- **sbin** Contains the binary executables for the root user (superuser)
- **sys** Contains a virtual file system that describes the system
- **tmp** Contains temporary files location
- **usr** Contains application programs for all of the users
- **var** Contains variable files such as system logs

BBB-ALE shall likely make use of the following key directories for file I/O during its operation:

- **dev**
- **lib** *NOTE: Shall contain all shared "Dynamically Linked Library" files used by BBB-ALE*
- **media/mnt**
- **proc**
- **tmp**
- **usr**
- **var**

To this list of key directories on this level we shall add the following BBB-ALE specific directory structures:

- **bbbale** shall contain the main BBB-ALE binary executable and any child process binary executable, binary library, data (configuration, startup, linked, unlinked, shutdown) and main ALE database file.
    - **classes** Where files pertaining to supported device classes will be maintained using "Dynamically Linked Library" files with .so suffix specific to BBB-ALE. *NOTE: Under desktop Windows the format is .DLL*
        - aatu
        - antsw
        - gps
        - lpa
        - modem
        - radio
        - wxsta
    - **dalog** Where the *DALOG* files ending in .da will be saved

        - **dict** Where DICTIONARY files are maintained.
        - **download** Where the files downloaded via DBM FTP
        - **help** Where the *HELP* files ending in .HLP will be maintained. HELP files detail MMI commands via any port which supports the execution of an MMI command.
        - **macro** Where the *MACRO* files ending in .MAC will be maintained. MACRO files are user created using MMI commands and arguments.
        - **ops** Where the *OPTIONS* files ending in .OPS will be maintained. Options files contain all operational parameters of the program expressed in MMI commands and arguments that are not used in .QRG files.
        - **owlog** Where the OW*LOG* files ending in .ow will be saved
        - **qrg** Where the *QRG* files ending in .qrg will be saved. QRG files contain Scan Group, NET and OTHER station data expressed in MMI commands and arguments.
        - **scripts** Where all BBB-ALE related Shell Script files will be maintained to include the installation, configuration and update scripts and the operational scripts such as *linked* and *unlinked*. All of these areas require a lot of analysis and testing, some items are given as to both the hardware configuration, e.g. the Governor (and locking it to 1Ghz operation) and ALE operation, e.g. *linked* and *unlinked*.
        - tbd...
- tbd...

It must be noted that some files can represent a lot of data writes if enabled, in particular the *dalog (.da)* and *owlog (.ow)* files. Thus a mechanism is planned for these files to be saved to microSD card storage and transferred to other storage locations.

BBB-ALE User Interfaces

I have no specific intentions of providing a Graphical User Interface (GUI) within the Embedded Linux model via the BBB HDMI interface or any other at this time.

The main means of BBB-ALE configuration will be its ASCII configuration files where the use of “nano” or any ASCII editor will be required for manual editing. In addition scripting files will be provided to configure BBB-ALE for known, recommended configurations, such as to work with a specific audio device configuration or LCD/Keypad that will be part of the standard BBB-ALE distribution.

BBB-ALE will also support setup via its front panel LCD/Keypad interface to some extent if so fitted. BBB-ALE will also support remote user entered MMI commands for configuration.

The local and remote interfaces with BBB-ALE will offer two levels of command and control:

1. Man Machine Interface (MMI) ASCII commands via “eth0” and a cross over cable or the USB Client port or “usb0” and a bridged cable or perhaps the serial debug port. PC hosted software applications (regardless of OS) for command and control, e.g. Telnet software such as PuTTy as a minimum will be required.

   NOTE: Third party developers can create GUI front ends of their own design targeting their OS of choice on a PC. In addition, developers may target the BBB for a co-hosted GUI written to communicate with BBB-ALE via the “lo” port at the 127.0.0.1 local loopback address.

2. Hands on front panel or hand held USB cable tethered LCD display with keypad capability or LCD display with USB keyboard or perhaps a wireless keyboard shall be supported to provide received message display and message entry capability as well as configuration and also command and control.

General Purpose I/O for external radio equipment interfacing and Debugging

ADC Mapping

The BBB offers 7 Analog Inputs (ADC) labeled AIN0..AIN6 which can be used in various ways within the BBB-ALE interfacing model. For example a directional coupler for VSWR/PWR could be interfaced to for antenna system monitoring and automatic shutdown should no other means of such data be available from an ATU if installed or from the HF radio.

Before they can be used however, the ADC ports must be enabled and then it must be determined where they are located in the device mapping for use.

The ADC properties are:

- 12 bits (Output values in the range 0-4095)
- 125ns sample time
- 0-1.8V range (as required real world levels must scaled as 1.8v must not be exceeded)
- 2 uA current flows into the ADC pin in this range
- If using a voltage divider, the lower leg that is connected to ground must be <= 1k Ohm
- Since we are measuring millivolts, resistors with 0.1% error tolerance should be used in a voltage divider.
- There is a 1.8V reference voltage VDD_ADC at Port 9 Pin 32.

- The GNDA_ADC that should be grounded to is on Port9 Pin 34.

The pin out listed in pin number ordering is provided in the table below.

Name	Pin #
VDD_ADC	32
AIN4	33
GNDA_ADC	34
AIN6	35
AIN5	36
AIN2	37
AIN3	38
AIN0	39
AIN1	40

The pin out is also depicted the header display below.

GPIO Mapping

The BBB offers a lot of General Purpose Input Output (GPIO) capability. Here is the BBB I/O header display of all the GPIO lines.

65 possible digital I/Os

P9

DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETn
GPIO_30	11	12	GPIO_60
GPIO_31	13	14	GPIO_40
GPIO_48	15	16	GPIO_51
GPIO_4	17	18	GPIO_5
I2C2_SCL	19	20	I2C2_SDA
GPIO_3	21	22	GPIO_2
GPIO_49	23	24	GPIO_15
GPIO_117	25	26	GPIO_14
GPIO_125	27	28	GPIO_123
GPIO_121	29	30	GPIO_122
GPIO_120	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	GPIO_7
DGND	43	44	DGND
DGND	45	46	DGND

P8

DGND	1	2	DGND
GPIO_38	3	4	GPIO_39
GPIO_34	5	6	GPIO_35
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
GPIO_23	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
GPIO_22	19	20	GPIO_63
GPIO_62	21	22	GPIO_37
GPIO_36	23	24	GPIO_33
GPIO_32	25	26	GPIO_61
GPIO_86	27	28	GPIO_88
GPIO_87	29	30	GPIO_89
GPIO_10	31	32	GPIO_11
GPIO_9	33	34	GPIO_81
GPIO_8	35	36	GPIO_80
GPIO_78	37	38	GPIO_79
GPIO_76	39	40	GPIO_77
GPIO_74	41	42	GPIO_75
GPIO_72	43	44	GPIO_73
GPIO_70	45	46	GPIO_71

There are 65 General Purpose I/O (GPIO) pins that can be made available for use, however some of the GPIO pins are not so general.

For example 24 GPIO pins used by the HDMI Video (20 pins) and Audio (4 pins) support alone and will have to be steered clear from as HDMI support may be desired by the end user, even if BBB-ALE does not directly support HDMI ALE interfacing at first or at all. The HDMI port on the BBB is implemented as a virtual cape. Other reasons for disabling HDMI are to make UART 5 and the flow control for UART 3 and 4 available.

Another 10 pins are used by the EMMC support but they can be freed up for use. 8 GPIO pins (GPIO1_0 TO gpio1_7) on the 20 pin header are used by the eMMC chip and pins GPIO1_30 and GPIO1_31. This means that if you are using the eMMC, those GPIO pins are not available. If you want to uses these GPIO pins, you can erase the contents of the eMMC chip and boot from the micro SD card. However running the SD card the OS loads slower and file I/O is slower than from the eMMC. The advantage of the eMMC over the SD card is that it allows you to boot directly from the board and because it is a fast interface, the BBB boots very quickly. It is a little limited at 2GB, but you still can mount a microSD card as a drive for additional storage.

The use of an SD card does have advantages in that all flash memory wears out from writing to the same cells x number of times, the larger the flash memory the long it takes to wear out. The BBB has a 2GB 8-bit eMMC (embedded Multi-Media Card), by Micron Semi. It has a density of 2GB on a 153-pin Ball Grid Array in a WFBGA package (WF means 0.8mm thick) . It is a NAND flash-based device but all internal transitions are handled by the device itself. NAND devices are currently more reliable and less prone to bit errors.

The flash memory controller spreads these writes over all the cells over time and once a cell wears out its dropped from use but that starts excellerating the wear on the remaining cells. The on board flash is small, so it will wear out fairly quickly compared to a 4GB or 8GB or 16GB SD card etc.

An excellent write up on a bootable SD card can be found at:

http://eewiki.net/display/linuxonarm/BeagleBone+Black#BeagleBoneBlack-SetupmicroSD/SDcard

SPI, I2C and the Timers use 4 pins each for another 12 total.

CLKOUT2, EHRPWM1A, EHRPWM1A, EHRPWM2A, EHRPWM2A each use 1 pin for another 5 total.

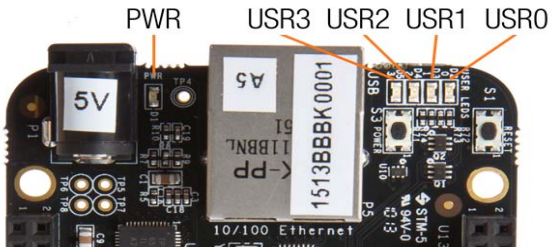
In addition a number of GPIO pins are used in the boot sequence and as I understand it, should be avoided as well as follows:

Pin	**Used for**
GPIO2_6	SYS_BOOT0
GPIO2_7	SYS_BOOT1
GPIO2_8	SYS_BOOT2
GPIO2_9	SYS_BOOT3
GPIO2_10	SYS_BOOT4
GPIO2_11	SYS_BOOT5
GPIO2_12	SYS_BOOT6
GPIO2_13	SYS_BOOT7
UART5_TXD	SYS_BOOT8
UART5_RXD	SYS_BOOT9
UART3_CTSN	SYS_BOOT10
UART3_RTSN	SYS_BOOT11
UART4_CTSN	SYS_BOOT12
UART4_RTSN	SYS_BOOT13
UART5_CTSN	SYS_BOOT14
UART5_RTSN	SYS_BOOT15

Then the following GPIO pins are mapped to the four User LEDS on the PCB which should likely be avoided:

USER LED	**GPIO PIN**
User LED 0	GPIO1_21
User LED 1	GPIO1_22

User LED 2	GPIO1_23
User LED 3	GPIO1_24



So just how many GPIO lines are free and clear for BBB-ALE use at all times it would own the OS is to be determined. Then a range of GPIO pins can be allowed for user selection for application use among those known to be free and clear and those that the user can make clear on their hardware by not using HDMI audio and video for example. For a user that chooses not to free up GPIO pins for use but yet needs I/O for signaling, other means such as external USB port based DLP Design devices can be used.

All of the GPIO pins are 3.3v max and can be configured as inputs or outputs. Any GPIO can be used as an interrupt and is limited to two interrupts per GPIO Bank for a maximum of eight pins as interrupts. The max output current varies by pin. some pins have 4mA and for some 6mA limit. A GPIO port needs to enabled for use and can be set to provide events where an event can trigger on a signal rising edge, falling edge or both.

All of the GPIO lines that are found to be available for our application will be mapped to potential uses such as Radio PTT, Radio Muting, ALE Alarms, external TNC and Modem PTT monitoring, Antenna Switch control, ATU control, PA Bypass etc.

TBD...

UART Mapping

The BBB has six on-board serial ports UART0..UART5 however only UART0, the /dev/ttyO0 is enabled by default, and it is coupled to the serial debug console and does not provide CTS/RTS support.

The other serial ports, which must be enabled before they can be used. UART3 is TX only with CTS and RTS lines, the others have RX, TX, CTS and RTS lines.

The UART map to pins and devices look like this:

Port	RX	TX	CTS	RTS	Device	Remark
UART0	J1_4	J1_5	n/a	n/a	/dev/ttyO0	BBB debug console
UART1	P9_26	P9_24	P9_20	P9_19	/dev/ttyO1	Full use
UART2	P9_22	P9_21	P8_37	P8_38	/dev/ttyO2	Full use
UART3	n/a	P9_42	P8_36	P8_34	/dev/ttyO3	TX only, see note below.

UART4	P9_11	P9_13	P8_35	P8_33	/dev/ttyO4	Full use, see note below.
UART5	P8_38	P8_37	P8_31	P8_32	/dev/ttyO5	Full use, see note below.

NOTE: To make UART 5 and the flow control for UART 3 and 4 available the HDMI port must be disabled.

UART0 is dedicated to the Serial Debug port console.

UART1, 2, 4 and 5 support TX/RX two-way use.

Why UART3 (/dev/ttyO3) is TX only must be attributed to some technical issue that I have not come across in my research. However this limits the usefulness of UART3 in our application to devices that do not require two-way data transfer. If it were RX only then I would dedicate to GPS receiver use and leave at that. However it will be recommended for external smart Antenna Switch device use and other device interfacing where two-way data exchanged is not a requirement. This can work for radio control with caveats as no data returned from the radio will be support, thus no polling support, however for actual operation that is not required of most HF SSB transceivers.

All ports are 5v TTL thus a TTL-UART to serial adapter circuit will be required for interfacing to any RS-232 device, such as an HF SSB radio. If used for a interfacing to an LCD/Keypad display then direct connection is permitted.

USB Port Serial Adapter

To achieve a full TxD, RxD, RTS, CTS, DSR, DTR, DCD, RI, and GND DB-9 capability the use of a standard FTDI chip based USB RS-232 adapter will likely work what with the out-of-the-box FTDI driver support. However I have not tested this as of yet as could not find any here to test with!

However I tried and was happy to find that the Prolific PL2303 based unit of which I have a number of here, these being the ones that LDG provided with the AT-200PC ATU, are apparently working as serial port ttyUSB1 seen below from an edited dmesg listing:

[9990.054137] usb 1-1.2: new full-speed USB device number 7 using musb-hdrc
[9990.134625] usb 1-1.2: default language 0x0409
[9990.134900] usb 1-1.2: udev 7, busnum 1, minor = 6
[9990.134915] usb 1-1.2: New USB device found, idVendor=067b, idProduct=2303
[9990.134927] usb 1-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber =0
[9990.134939] usb 1-1.2: Product: USB-Serial Controller
[9990.134950] usb 1-1.2: Manufacturer: Prolific Technology Inc.
[9990.135492] usb 1-1.2: usb_probe_device
[9990.135509] usb 1-1.2: configuration #1 chosen from 1 choice
[9990.135638] usb 1-1.2: adding 1-1.2:1.0 (config #1, interface 0)
[9990.214507] usbcore: registered new interface driver pl2303
[9990.217014] usbserial: USB Serial support registered for pl2303
[9990.222681] pl2303 1-1.2:1.0: usb_probe_interface

[9990.222709] pl2303 1-1.2:1.0: usb_probe_interface - got id
[9990.222743] pl2303 1-1.2:1.0: pl2303 converter detected
[9990.232113] usb 1-1.2: pl2303 converter now attached to ttyUSB1
root@beaglebone://#

Then for the heck of it I tried my old favorite Belkin FU109 adapter, which uses a microcontroller as its brain and although Belkin does not support past XP, drivers from another vendor that supports Windows 7 work great. Well, to my surprise Debian on the BBB likes the bugger, here is a listing from using dmseg were its now ttyUSB2:

[11255.718171] usb 1-1.4: new full-speed USB device number 8 using musb-hdrc
[11255.798195] usb 1-1.4: ep0 maxpacket = 16
[11255.801711] usb 1-1.4: default language 0x0409
[11255.811500] usb 1-1.4: udev 8, busnum 1, minor = 7
[11255.811519] usb 1-1.4: New USB device found, idVendor=050d, idProduct=0109
[11255.811570] usb 1-1.4: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[11255.811583] usb 1-1.4: Manufacturer: Belkin USB PDA Adapter
[11255.811593] usb 1-1.4: SerialNumber: 605277
[11255.812199] usb 1-1.4: usb_probe_device
[11255.812218] usb 1-1.4: configuration #1 chosen from 1 choice
[11255.812740] usb 1-1.4: adding 1-1.4:1.0 (config #1, interface 0)
[11255.894639] usbcore: registered new interface driver mct_u232
[11255.896857] usbserial: USB Serial support registered for MCT U232
[11255.899912] mct_u232 1-1.4:1.0: usb_probe_interface
[11255.899941] mct_u232 1-1.4:1.0: usb_probe_interface - got id
[11255.899975] mct_u232 1-1.4:1.0: MCT U232 converter detected
[11255.906167] usb 1-1.4: MCT U232 converter now attached to ttyUSB2

Thus a hub hung off the USB host port will yield full 9 pin RS-232 via the use of USB to RS232 adapters it now appears, with Prolific and the MCT U232 recognized off the get go then so is FTDI based units as perhaps other USB chipset based adapters.

Serial Debug Port

During development the Serial Debug Port Header will be utilized for debugging. As seen below the 6 pin debug header is located to the inside of the expansion header.

Just what if any use this port may be in the actual BBB-ALE implementation aside from debugging is to be determined. Its a direct connection to an FTDI and is always active however.

The debug serial port UART0(/dev/ttyO0) settings are:

115,200 Baud, 8 Data Bits, No Parity, 1 Stop Bits, No Handshaking

One of the best to the point written "How To" the debug port use can be found at:

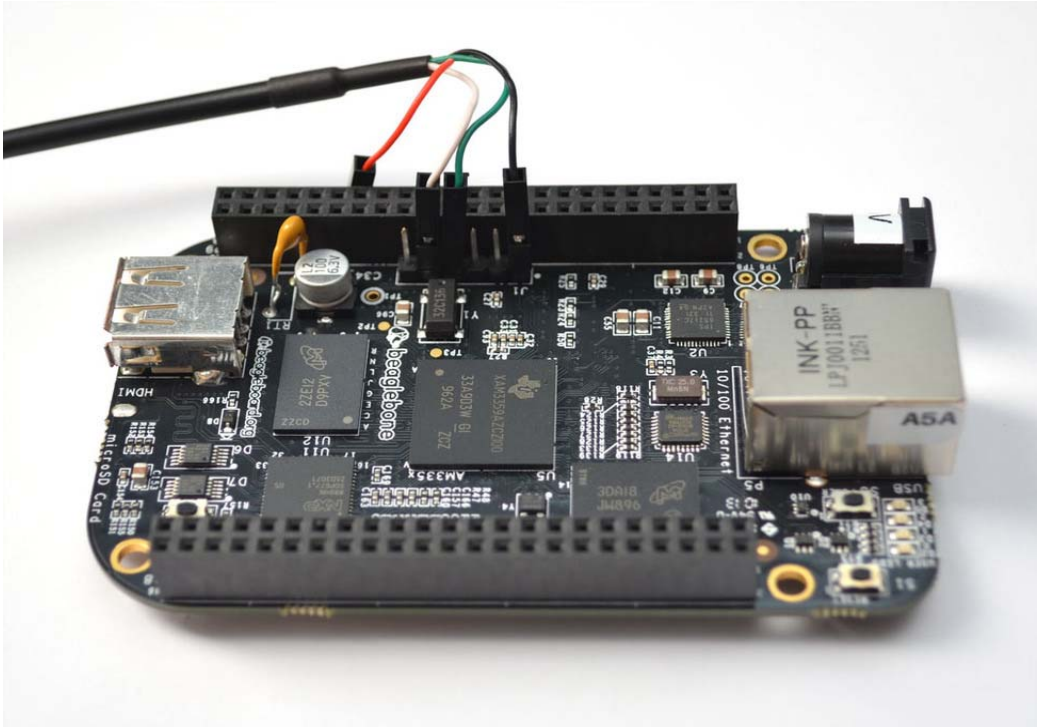
http://codechief.wordpress.com/2013/11/11/beaglebone-black-serial-debug-connection/

The debug port header accepts the $20USD FTDI to TTL cable, where the 3.3v version

is required (http://www.ftdichip.com/Support/Documents/DataSheets/Cables/DS_TTL-232R_CABLES.pdf). Pin 1 on the cable is the black wire and connects to pin 1 on the debug header on the BBB. If there is a use for the debug port in BBB-ALE all the time with the PC then this cable would likely be the best route to take where it would be fished out the case.

However, I bought the Adafruit 3 foot USB cable, P/N PL2303 with individual pin connector terminations for more versatility in other uses. It is a Prolific device based cable available for $10USD as seen below.

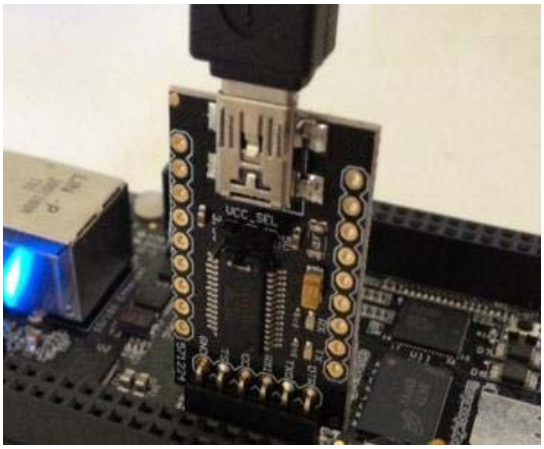
There are four wires with the Adafruit cable: Red power, Black ground, White RX into USB port, and Green TX out of the USB port. The power pin provides the 5V @ 500mA direct from the USB port and the RX/TX pins are 3.3v signal level vs. 5v TTL. Where the Red wire D.C. power connection from the USB port of the PC is not required as seen below.



Either of the two above cable approaches would permit always have the cable connected, even with the case attached.

Another apparently popular approach used by many seems to be direct USB to Serial adapter PCB for around $20USB where as with the FTDI cable, you can plug it directly

into the debug header on the BBB. These units are jumper selectable between 3.3v and 5v uses, many vendors carry it, details can be found at http://www.dfrobot.com/wiki/index.php/FTDI_Basic_Breakout_3.3/5V_(SKU:_DFR0065)



This approach is obviously limited to lab bench debug applications only vs. more field like conditions, forget about have having a cable always connected and the case in place.

The debug port starts displaying the U-boot sequence. U-boot uses a board configuration file called a device tree (also called a device tree binary) that contains board-specific information that the kernel requires to boot the board. This file contains all of the information needed to describe the memory size, clock speeds, on-board devices etc.This device tree binary or dtb (binary) is created from a dts (source) file using a compiler called a device tree compiler (dtc). We are able to develop our own additions and modifications to the description using Device Tree Overlays (DTOs) which can be installed in the */lib/firmware* directory of our Linux distributions.

Here is part of the capture starting a U-boot running Debian on my BBB being used for development:

U-Boot SPL 2014.04-rc2-00015-g99288ca (Mar 12 2014 - 09:49:41)
reading args
spl_load_image_fat_os: error reading image args, err - -1
reading u-boot.img
reading u-boot.img

U-Boot 2014.04-rc2-00015-g99288ca (Mar 12 2014 - 09:49:41)

I2C: ready
DRAM: 512 MiB
NAND: 0 MiB
MMC: OMAP SD/MMC: 0, OMAP SD/MMC: 1
***** Warning - readenv() failed, using default environment**

Net: <ethaddr> not set. Validating first E-fuse MAC
cpsw, usb_ether
Hit any key to stop autoboot: 0
gpio: pin 53 (gpio 53) value is 1
Card did not respond to voltage select!
mmc0(part 0) is current device
Card did not respond to voltage select!
gpio: pin 56 (gpio 56) value is 0
gpio: pin 55 (gpio 55) value is 0
gpio: pin 54 (gpio 54) value is 0
mmc1(part 0) is current device
gpio: pin 54 (gpio 54) value is 1
SD/MMC found on device 1
reading uEnv.txt
1417 bytes read in 6 ms (230.5 KiB/s)
gpio: pin 55 (gpio 55) value is 1
Loaded environment from uEnv.txt
Importing environment from mmc ...
Checking if uenvcmd is set ...
gpio: pin 56 (gpio 56) value is 1
Running uenvcmd ...
reading zImage
3717224 bytes read in 207 ms (17.1 MiB/s)
reading initrd.img
2870434 bytes read in 163 ms (16.8 MiB/s)
reading /dtbs/am335x-boneblack.dtb
25080 bytes read in 11 ms (2.2 MiB/s)
Kernel image @ 0x82000000 [0x000000 - 0x38b868]
Flattened Device Tree blob at 88000000
Booting using the fdt blob at 0x88000000
Using Device Tree in place at 88000000, end 880091f7

Starting kernel ...

Next comes the starting of the Kernel:

Uncompressing Linux... done, booting the kernel.
[0.381698] omap2_mbox_probe: platform not supported
[0.548907] tps65217-bl tps65217-bl: no platform data provided
[0.612522] bone-capemgr bone_capemgr.9: slot #0: No cape found
[0.649629] bone-capemgr bone_capemgr.9: slot #1: No cape found
[0.686737] bone-capemgr bone_capemgr.9: slot #2: No cape found
[0.723847] bone-capemgr bone_capemgr.9: slot #3: No cape found
[0.740010] bone-capemgr bone_capemgr.9: slot #6: BB-BONELT-HDMIN conflict P8.45 (#5:BB-BONELT-HDMI)
[0.749588] bone-capemgr bone_capemgr.9: slot #6: Failed verification
[0.756324] bone-capemgr bone_capemgr.9: loader: failed to load slot-6 BB-BONELT-HDMIN:00A0 (prio 2)
[0.772863] omap_hsmmc mmc.5: of_parse_phandle_with_args of 'reset' failed
[0.835449] pinctrl-single 44e10800.pinmux: pin 44e10854 already requested by 44e10800.pinmux; cannot claim for gpio-leds.8
[0.847139] pinctrl-single 44e10800.pinmux: pin-21 (gpio-leds.8) status -22
[0.854424] pinctrl-single 44e10800.pinmux: could not request pin 21 on device pinctrl-single
Loading, please wait...
Scanning for Btrfs filesystems

systemd-fsck[204]: rootfs: clean, 75612/111104 files, 374073/444160 blocks

Debian GNU/Linux 7 beaglebone ttyO0

default username:password is [debian:temppwd]

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

The IP Address for usb0 is: 192.168.7.2
beaglebone login: [27.176084] libphy: PHY 4a101000.mdio:01 not found
[27.181354] net eth0: phy 4a101000.mdio:01 not found on slave 1

NOTE: At this point the enter key needs to be pressed to continue...

Debian GNU/Linux 7 beaglebone ttyO0

default username:password is [debian:temppwd]

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

The IP Address for usb0 is: 192.168.7.2
beaglebone login: root
Last login: Thu Mar 27 16:47:01 UTC 2014 from 192.168.7.1 on pts/0
Linux beaglebone 3.8.13-bone43 #1 SMP Wed Mar 26 14:21:39 UTC 2014 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@beaglebone:~#

==========

Exit from Debug port:

root@beaglebone:~# exit
logout

Debian GNU/Linux 7 beaglebone ttyO0

default username:password is [debian:temppwd]

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

The IP Address for usb0 is: 192.168.7.2
beaglebone login:

Although the use of the Linux command “dmesg” and “ifconfig” and others can be used to grab a lot of the above information, those commands can only be used if you actually succeed in booting to the command prompt or GUI, where as the debug port gives you a view during the entire boot process.

The serial debug port from this point can be used just as would an SSH Telnet connection, the main difference being that you are connected prior to the OS being booted via the debug port. As a matter of fact, I am using the current PuTTy with serial support is my means of using the debug port with the Adafruit Prolific cable running under 32 bit Windows 7.

JTAG Port Header

JTAG stands for Joint Test Action Group and is a nickname for the committee that drafted "IEEE Standard 1149.1 - 1990 Standard Test Access Port and Boundary-Scan Architecture". A good review of ARM based JTAG can be had by a review of the following document:

http://andrewsterian.com/424/Lecture22.pdf

The BeagleBone (White) provided for JTAG support via a JTAG emulator using the USB client port. However the BBB dropped that support as a cost savings initiative, thus making JTAG an added expensive and effort, but also making JTAG on the BBB more powerful.

At this time I am not sure if the JTAG port header will be utilized for in-circuit debugging purposes during development due to the cost. It's nice to have but truly only required for debugging when all else fails to provide the answer as to why something is not working properly.

The BBB provides for JTAG support where the JTAG connector requires soldering to the bottom of the BBB by the end user. A good run down of the details can be found at:

http://www.tincantools.com/wiki/Flyswatter2_BeagleBone_Black_How_To

The JTAG interface:

http://www.tincantools.com/JTAG/Flyswatter2.html

The cable kit:

http://www.tincantools.com/JTAG/BeagleBone-Black-JTAG-Adapter-Kit.html

The Open On-Chip Debugger software:

http://openocd.sourceforge.net/

Once the OpenOCD server is up, you can connect to it via telnet interface (default port: 4444) or via GDB (default port 3333). From there you can issue commands. For more information on the commands see:

http://openocd.sourceforge.net/doc/html/General-Commands.html

USB Host

The USB Host has many potential uses for interfacing to radio devices in addition to devices such as external LCD/Keypad, Keyboard etc. However as its a single port the use of a powered hub either installed internally or external to the BBB-ALE case will be required. Make sure any use of a Hub is made by the Hub being plugged into the USB Host connector and powered on before powering on the BBB.

For radio device interfacing the USB Host port will be supported for use This makes for ease of use for ALE applications of the support being developed for desktop MARS-ALE and PC-ALE for use of the DLP Design Data Acquisition and Relay boards for external Hardware Modem, ATU, ANT SW, Alarms and other uses.

The DLP Design boards will likely be the first USB external interface devices tested under BBB-ALE. The DLP-IO8-G alone would provide 8 ports that in one user selected scenario could provide for the following interface model that supports ALE Radio PTT, LPA low power ATU tune, External Modem PTT Monitoring and ALE Alarms and this is just one model for the the use of the 8 channels:

NOTE: The use of DO represents Data Out and DI represents Data In.

1. [Chan 1/DO] User selectable as follows:

A. Radio hardware PTT keying instead of RS-232 DTR/RTS hardware PTT.

B. Radio Bypass signal during Scanning for QS/S radio modification needs of a D.C. signal source.

2. [Chan 2/DO] Radio Speaker Muting.

3. [Chan 3/DO] LPA keying line bypass for low power ATU tuning.

4. [Chan 4/DO] External Hardware Modem PTT monitoring of either PTT to Ground or PTT on Low to High D.C.

5. [Chan 5..8] ALE ALARMS

[Chan 5/DO] QRZ Call Missed Alarm. Requires user to clear, The QRZ option must be enabled.

[Chan 6/DO] ALE Linked Alarm. Requires user to clear while linked. Clears automatically when ALE link is cleared.

[Chan 7/DO] AMD Message Received WAITING. Requires user to clear.

[Chan 8/DI] CLEAR for all Alarms.

Another scenario would be the use of all 8 ports of a DLP-IO8-G being used for interfacing to an ATU with ANT port selection. Yet another could be selections of antenna coil taps in a mobile or portable antenna configuration where the 8 ports would be assigned to the channels of a scan group. Multiple DLP-IO8-G units can be in use, so can multiple application models. I am also working with the 20 channel I/O unit and the latched relay unit. Then too there is already an Ardruino emulation of the 8 channel I/O unit in place which is an even less expensive option.

USB Audio

I am testing USB audio to avoid use of the Audio Cape under Debian after its use under Angstrom. As you can see from a review of the Kernel information listed previously under the "Serial Debug Port" section, there is some conflict which is affecting the HDMI interface and its audio support that I need to resolve. The HDMI is working properly with the video display and passes high level audio interface tests, but yet I do not have any audio output.

Here is a capture of an attempt to play a serial tone test wave file as an example, were aplay loads the file and lists the details and then the time required to play the file expires and it prints a message that the file was not found:

```
root@beaglebone:/tmp# aplay MDM3001_600Ltest.wav alsa "default:CARD=CODEC"
Playing WAVE 'MDM3001_600Ltest.wav' : Signed 16 bit Little Endian, Rate 48000 Hz, Mono
alsa: No such file or directory
```

However the file is there:

```
root@beaglebone:/tmp# ls -l
total 7736
-rw-r--r-- 1 debian debian 7913612 Sep  5  2012 MDM3001_600Ltest.wav
root@beaglebone:/tmp#
```

I have not yet tested with the standard audio cape under Debian as it requires disabling the HDMI cape and the loose of video display and VNC support. I have however been trying to make use of USB audio to no avail as the HDMI audio seems to be blocking it. However I do get a USB Signalink recognized, I can not select it however as the default device and I can not otherwise address and get any audio out from it.

Here is what the system via "aplay" arguments sees as to sound devices and their capabilities:

```
root@beaglebone:/tmp# aplay --list-devices
**** List of PLAYBACK Hardware Devices ****
card 0: Black [TI BeagleBone Black], device 0: HDMI nxp-hdmi-hifi-0 []
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 1: CODEC [USB Audio CODEC], device 0: USB Audio [USB Audio]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
root@beaglebone:/tmp#
```

And also:

```
root@beaglebone:/tmp# aplay -L
null
    Discard all samples (playback) or generate zero samples (capture)
default:CARD=Black
    TI BeagleBone Black,
    Default Audio Device
sysdefault:CARD=Black
    TI BeagleBone Black,
    Default Audio Device
default:CARD=CODEC
    USB Audio CODEC, USB Audio
    Default Audio Device
sysdefault:CARD=CODEC
    USB Audio CODEC, USB Audio
    Default Audio Device
front:CARD=CODEC,DEV=0
    USB Audio CODEC, USB Audio
    Front speakers
surround40:CARD=CODEC,DEV=0
    USB Audio CODEC, USB Audio
    4.0 Surround output to Front and Rear speakers
surround41:CARD=CODEC,DEV=0
    USB Audio CODEC, USB Audio
    4.1 Surround output to Front, Rear and Subwoofer speakers
surround50:CARD=CODEC,DEV=0
    USB Audio CODEC, USB Audio
    5.0 Surround output to Front, Center and Rear speakers
surround51:CARD=CODEC,DEV=0
    USB Audio CODEC, USB Audio
    5.1 Surround output to Front, Center, Rear and Subwoofer speakers
surround71:CARD=CODEC,DEV=0
    USB Audio CODEC, USB Audio
    7.1 Surround output to Front, Center, Side, Rear and Woofer speakers
iec958:CARD=CODEC,DEV=0
    USB Audio CODEC, USB Audio
    IEC958 (S/PDIF) Digital Audio Output
root@beaglebone:/tmp#
```

Obviously the Linux USB Codec driver for the Burr-Brown devices in the USB Signalink has a lot of wishful thinking going on!

I have now succeeded in getting the USB Signalink to work, still nothing from HDMI audio. I did not have to disable HDMI to get the Signalink to work as is required with the standard audio cape. What lead me in the right direction were details that are part of the dissertation at:

http://walkerips.com/wp/shairport-on-beaglebone-black/

However the output level driving an external speaker, for the given test file being played, is much lower then when the same Signalink wired to the same speaker and using the

same test file is played on a real PC, even though I have the alsamixer set to max and the front panel control on the Signalink wide open.

Specifically what was required was the installation of a number of alsa packages:

sudo apt-get install libpulse-dev
sudo apt-get install alsa-base
sudo apt-get alsa-base-config *NOTE: This one fails to install.*
sudo apt-get install libavahi-client-dev
sudo apt-get install libasound2-dev
sudo apt-get install libssl-dev
sudo apt-get install git
sudo apt-get install make
sudo apt-get install wget

Then the creation of /etc/asound.conf file that configures the USB sound device as the default device:

```
pcm.!default {
      type hw
      card 1
}

ctl.!default {
      type hw
      card 1
}
```

Then in the file /etc/modprobe.d/alsa-base.conf change "options snd-usb-audio index= -2″ to "options snd-usb-audio index=1″ which will allow ALSA to load the USB sound driver as card 1.

NOTE: The built in HDMI is card 0. You can disable the HDMI but it really gains you nothing and you might want to use it later.

Then after rebooting, now the the USB Audio CODEC can be selected in alsamixer as the default sound device and it is retained upon restarting alsamixer. However now the TI Black HDMI can be selected and retained on restart.

After rebooting a test was made using:

aplay MDM3001_600Ltest.wav

Which resulted in:

Playing WAVE 'MDM3001_600Ltest.wav' : Signed 16 bit Little Endian, Rate 48000 Hz, Mono

Where the USB Signalink as the default device does indeed now play the test serial tone modem .wav file recorded from the MDM-3001 modem.

Date and Time Supporting

ALE requires the proper Date and Time to accurately display and store information pertaining to ALE intercepted signals. Accurate Time is also a requirement to provide ALE Link Protection (LP) support.

As the BBB does not contain an on-board battery backed-up real time clock the choices are:

1. Manually User Entered Date and Time
2. Real-Time Clock Module with Battery Back-up
3. GPS Receiver
4. Internet Time Server
5. ALE Over-the-air Time Server

The issues with item 1. is that the user must enter the Date and Time each time BBB-ALE is booted from a call start, which if it were to always run and accurate information was entered, would suffice. Also, whenever it does require a restart for any reason the system would have to prompt the user to enter the Date and Time if found to be at default values.

The issues with item 2. is that of additional cost and packaging real estate. A Real Time Clock Module is required interfaced to the BBB where the I2C bus would be the best approach using a COTS DS1307 (http://datasheets.maximintegrated.com/en/ds/DS1307.pdf) based RTC module of which there are many inexpensive modules available.

The issues with item 3. is that of additional cost and packaging and use of 12 GPIO pins if the GPS Cape (http://www.exploitsys.com/ENG/GPS_GPRS%20BB%20Cape.html) is used. In addition, as with the Audio Cape if used, HDMI needs to be disabled due to hardware resource conflicts and I do not know if the Audio Cape and GPS Cape can coexist at this time. If an outboard serial GPS unit is used then add the dedicating of a UART port for GPS use. The use of a NMEA compatible USB port based GPS device or GPS dongle which are rather inexpensive would seem to be the best GPS hardware solution as only a powered USB hub is required if any other USB devices are being used. Make sure any use of a Hub is made by the Hub being plugged into the USB Host connector and powered on before powering on the BBB.

The issues with item 4. deal with having Internet connectivity to connect with the Internet Time Standard Server configured.

NOTE: Under Angstrom this support must all be configured. Under Debian, which is the targeted OS for BBB-ALE, it comes setup with the official distribution image.

The issues with item 5. deals with standards to bring it about. This type of support would work in conjunction with other ALE stations configured to be Time Servers. This support will debut in MARS-LP-ALE in support of LP and can be implemented in BBB-ALE. It

is dependent upon others being operational as an ALE Time Server with an accurate Time source reference.

In all the cases above, after initial boot and all scripts configured running, when the BBB-ALE API starts, if the default Date and Time are found to be prevent than the user will be prompted to enter the current Date and Time into the system for BBB-ALE to continue. The user entered Date and Time will be verified to be more recent than the last saved Date and Time to continue unless it is the first start of the BBB-ALE API on the host.

Expansion PCB

The best approach to a turn-key BBB-ALE solution as to hardware I/O requirements would be to design a custom Expansion Printed Circuit Board (EPCB) which the BBB via its two header connectors would attach. The design of a cape would be out of the question due to the foot print size of a cape.

What would be just right is an EPCB that was to the mini-ITX 6.7 × 6.7 in form factor to make use of commercial cases.

The EPCB would provide:

- Connection to 13.8/12v D.C. nominal (8-28vdc) systems and on board fusing
- Regulation from 13.8/12v D.C. to 5.1v D.C. and any other required voltages, distribution and fusing for the BBB and other devices
- USB 4 or 6 port hub option or standard feature on the board would be a big plus
- An Audio Codec with TXCO and radio impedance isolated interfacing
- BBB/UART to RS-232 and perhaps an RS-232/RS-485 configurable port with CTS, RTS and DTR
- GPIO to real world I/O interfacing for PTT and other uses
- All serial port, audio and GPIO signal connections to the real world via standard DB-9 and DB-25 connectors

A detailed definition as to the GPIO lines use for radio application would be required for such a EPCB. I idea would be to allow for use of the lines that support HDMI as auxiliary GPIO in case the end user wanted to actually use HDMI for video display.

The I2C bus although perhaps used on the EPCB for the audio codec and or other uses would also need to support auxiliary uses such as from LCD/Keypad support.

The EPCB should also permit either direct connection to the 6 pin serial debug port or extend its access via interconnection to the EPCB.

The OEM of the Codec board of interest, also designs their products on header boards with project specific circuitry such as serial I/O, voltage regulation etc. and plug in the audio Codec PCB and BBB as can be seen in the following image.


Sierra Radio Systems - #270 BB Linux Host Adapter - Rev 2 Build 7
LD1117AV12
MAX3232
E/77777

Packaging BBB-ALE

The BBB-ALE system for some could just the basic unit in a minimal case that supports the required interfacing for radio control and audio I/O under PC control using PuTTy or some GUI interface.

The BBB has a small-footprint and all alone it can be mounted in an Altoids can, the Logic Supply case below however is likely the smallest best choice and as you can see it fits in the palm of an average male's hand.

At a minimum for the BBB-ALE application an audio codec device will be required mounted to the BBB such as the standard Audio Cape in a larger case if used or a co-mounted codec PCB which can fit in much smaller case. Also required will be at least one RS-232 serial port for HF SSB radio control unless the radio supports USB or TCP/IP. The prospect of a radio with a USB codec being supported depends on drivers for Linux being available.

For others it could include the USB LCD/Keypad and perhaps a USB (wired or wireless) Keyboard and no PC.

Then also these is the option of a larger case package with integrated LCD/Keypad and perhaps some may want to make use of the HMDI video support?

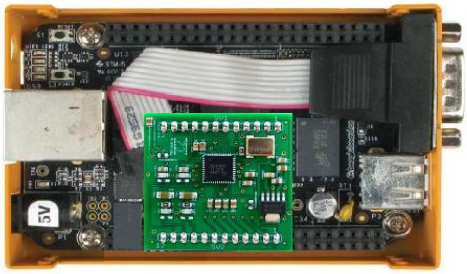
**The LCD displayed initialization message on the Logic Supply case is a mockup at this time*

The standard RS-232 cape is of course available for use in development and some COTS cases for deployment. Then too there is a COTS solution from Logic Supply for $20USD simplifies packaging of an RS-232 port into a small case and supports configuration for DTE or DCE use.

Logic Supply also offers small, inexpensive metal cases for mounting the BBB and their RS-232 option board as well as offering long ribbon to D connector cables for larger cases, see: http://www.logicsupply.com/components/beaglebone/capes/cbb-ttl-232/

The case with CBB-TTL-232 installed is pictured below in Orange, Black is also available. All ports on the BBB are exposed to the world for connectivity.

It can hold not only the CBB-TTL-232 board and its ribbon cable but should also hold the codec PCB with TXCO option that is of interest for data modem adhered to the serial board for use as pictured in the mockup image below.



The pop out switch hole next to the ethernet port connector can be used to bring out a cable with ground, audio in/out and radio PTT lines retained with a nylon tie on either side of the case or perhaps a connector like PS/2 mouse or S-Video could be used.

Then an even larger case will be required should any needed PCB(s) for ADC or GPIO or UART to RS232 and 12v DC to 5.1v DC operation and possibly and internal USB hub be utilized. Not to mention a case large enough to accept an internal LCD/Keypad for a truly functional and professionally packaged appearance. However as this is a build-it-yourself approach the packaging is all up the end users needs and budget.

The case below by Logic Supply provides an internal 4x20 LCD/keypad. These units can also be had specifically package for automotive use where proper D.C. cabling and even ignition interface wiring can be had as seen below in a rear view.

Logic Supply also sells their case with Crystalfontz 4x20 LCD display as a separate package intended for their Atom based Mini-ITX form factor boards they market. However the ARM based BBB and its out board codec board could be installed into it as well. The case with 4x20 LCD display its currently priced at $175USD where the details can be found at http://www.logicsupply.com/ml251-lcd/?___SID=U

There are so many suitable cases to mount the BBB or Intel Atom boards in that I can not list them all here. However a decent looking $50USD case which could house the supported 4x20 LCD display where the rear would become the front so as to mount the LCD display in the opening for a rear panel. It can be had via Amazon or direct from the manufacture at:

http://www.amazon.com/dp/B0087TS1S4/ref=asc_df_B0087TS1S43061659?tag=thefind0131442-20&creative=395261&creativeASIN=B0087TS1S4&linkCode=asn or at http://www.mitxpc.com/proddetail.asp?prod=557

The same outfit for one stop shopping also has the Intel Atom Marshalltown board (http://www.intel.com/content/www/us/en/motherboards/desktop-motherboards/desktop-board-dn2800mt.html) if one is interested in going the build it all yourself route vs. a tested COTS solution, see: http://www.mitxpc.com/proddetail.asp?prod=ITLDN2800MT However taking this route moves from the fanless industrial computer to a fan requirement which the above case and many others support.

User Display and Data Input Interface

All ALE radios and external stand alone ALE modem/controllers have LCD displays and some method of user entry for configuration and mode selection and other uses such manual PTT for data modems and message display and message entry with respect to ALE. For my vision this type of capability will be a supported option for MARS members that desire such capabilities as it adds cost. However it will be inherently supported in the software so that the user can add the option at any time. I plan to provide data display via a 4x20 line LCD display where the user can not only display messages

live but also scroll back through the last messages received during the current use and perhaps even select saved messages from previous use depending on Data Modem vs. ALE applications.

Also the capability to compose a message on-the-fly via just a basic LCD/keypad interface via navigation and character selection shall be provided for Data Modem and ALE. In addition, if a keyboard is found available for use as an input device then it will be used for message generation with the message displayed on the LCD display.

Below is a 4x20 LCD display packaging option offered by Crystalfontz, the same LCD display manufacturer used in the Logic Supply unit above. This is a light weight external USB port hand held configuration in a metal case with 9 foot cable with all the same display, control and status features of the front panel display seen in the embedded computer above. It is suitable for use with the BBB or development PC in my efforts via a USB port.

The external USB LCD units depending on display options run $120-130USD where the internal units are just under $100USD. The provided XES635BK-TFE-KU model seen below is clear, sharp and bright, characters change without delay and keypad response is immediate and the keys have a great tactile feel.

The data on this particular model is at:

http://www.crystalfontz.com/product/XES635BKTFEKU

I made a pair of short demo videos using the developers tool which are at:

www.n2ckh.com/MARS_ALE_FORUM/MOV04236.MPG

and a shorter video with less talk and no keypad demo at:

www.n2ckh.com/MARS_ALE_FORUM/MOV04233.MPG

Debian on the BBB comes with FDTI drivers installed that work just fine with the USB Crystalfontz display which in the BBB is recognized as listed using “dmesg”:

[1.919360] usb 1-1.5: new full-speed USB device number 5 using musb-hdrc
[1.999378] usb 1-1.5: ep0 maxpacket = 8
[2.000844] usb 1-1.5: default language 0x0409
[2.007712] usb 1-1.5: udev 5, busnum 1, minor = 4
[2.007735] usb 1-1.5: New USB device found, idVendor=0403, idProduct=fc0d
[2.007748] usb 1-1.5: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[2.007759] usb 1-1.5: Product: Crystalfontz CFA635-USB LCD
[2.007770] usb 1-1.5: Manufacturer: Crystalfontz
[2.007780] usb 1-1.5: SerialNumber: CFnnnnnn
[2.008251] usb 1-1.5: usb_probe_device
[2.008270] usb 1-1.5: configuration #1 chosen from 1 choice
[2.008436] usb 1-1.5: adding 1-1.5:1.0 (config #1, interface 0)

It takes about 8 seconds after applying power when booting from eMMC for the USB LCD/Keyboard to come alive as the FDTI drivers load early for the serial debug console support. At that point the display that is stored in the LCD displays flash RAM is displayed. It takes another 22 seconds before the desktop GUI comes to life, just how soon a user application (e.g. BBB-ALE) configured to automatically start would be able to commence processing and interface with the LCD display is to be determined.

Here is a short video that captures the BBB boot and LCD and desktop GUI sequence of events:

www.n2ckh.com/MARS_ALE_FORUM/MOV04266.MPG

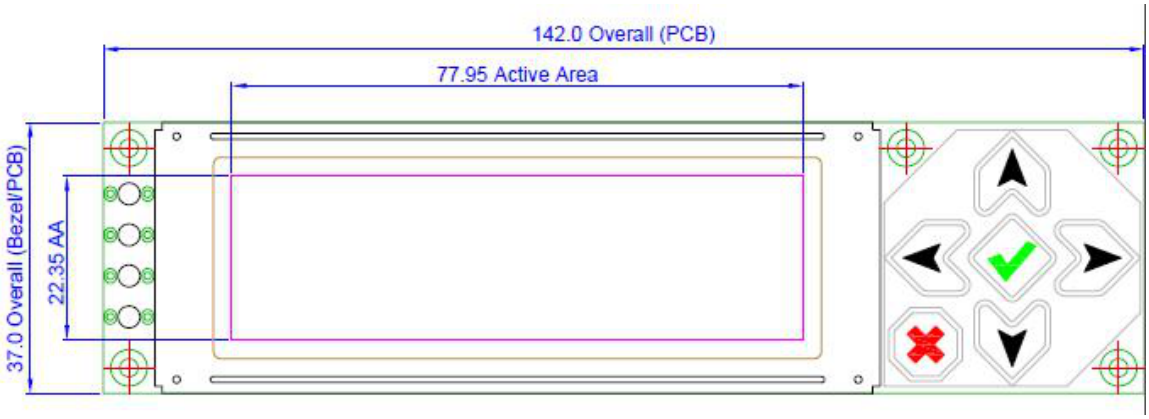
The scheme that I have in mind on boot and BBB-ALE startup as to the LCD display is to ping to validate if one is installed and if so read the first line and interrogate the version and build information (or if factory fresh), which if not current, then update it and store it to flash RAM. At the same time the bottom line will start to show activity such as with the 3 periods increasing to 4, then 5, then 6 and with a flashing cursor, and also make use of the 4 status LED's to alert the user that the BBB-ALE is performing its start up initialization sequences.

If implemented and if connected to a router and the internet and configured for use, the user may then be prompted to check for BBB-ALE updates and perhaps other house keeping prior to moving forward to ALE operational modes.

For packaging within a case Crystalfontz offers 5 ¼ bay mounting as well where the BBB other supported ARM board can be housed in such an enclosure to accept an I2C or RS232 version of the display. The 5 ¼ drive bay kit front panel that supports any of the Crystalfontz 4x20 LCD panel displays, a similar bracket exists for their 2x16 display line as well.

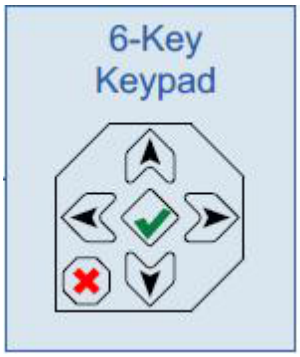
Below is one of the display models for mounting in the above bay kit where either I2C, RS232 or USB models are available, the unit pictured is a USB model with USB cable.

End users with the ability and cost savings incentive can make their own cut out panels and bezel, all the details are provided from the manufacturer in the form of drawings and templates.



End users with the ability and cost savings incentive can make their own cut out panels and bezel, all the details are provided from the manufacturer in the form of drawings and templates.

The 6 keys are Up, Down, Left, Right which are obvious, Enter being the green check mark in the center and Exit being the red X button. The Keypad on units with it integral is also used for Power On via the Enter key and Power Off via the X key as desired.



The type of keyboard activity events returned are:

1. KEY_UP_PRESS 1
2. KEY_DOWN_PRESS 2
3. KEY_LEFT_PRESS 3
4. KEY_RIGHT_PRESS 4
5. KEY_ENTER_PRESS 5
6. KEY_EXIT_PRESS 6
7. KEY_UP_RELEASE 7
8. KEY_DOWN_RELEASE 8
9. KEY_LEFT_RELEASE 9
10. KEY_RIGHT_RELEASE 10
11. KEY_ENTER_RELEASE 11
12. KEY_EXIT_RELEASE 12

As such in addition to normal use additional functionality can be achieved by monitoring the duration of time between a key press and key release, such as the X key being pressed longer than x to validate that Power Off is desired as the result.

The 4 bi-color status LED's of the 4 line displays will get plenty of use as well in providing status on states and events taking place in a multi-functional use of each LED based on what operations are being performed and use of steady state or changing state and color of the given LED.

The 4x20 LCD display has great potential for data entry and editing with the data being entered for sending also being displayed on the LCD panel and no keyboard required in a pinch but supported for use with the LCD panel, which I really like the idea of providing, as a full keyboard and the LCD display is an even more powerful combination. However the 4x20 LCD panel alone is just the thing for a bare bones operation of Data Modem or ALE in the field where a PC or even a connected video monitor or keyboard to send and display messages with is undesirable in many poor conditions as we all know such devices all fail and frequently as well.

There are of course many makes and models of LCD displays, however it is the 2x16 and 4x20 Crystalfontz displays that will be the focus of my development where the 4x20 will definitely be supported with perhaps 2x16 support provided as well. Support for similar models from other another manufacturer as an alternate source may be provided at a later

date if deemed required. I will be supporting the USB and I2C and perhaps the RS-232 display models as both the BBB and Atom units have numerous RS-232 ports and a I2C port on the board.

The prospects of supporting LCD Graphics or LCD Graphics Touch screen interface for use with USB keyboard also exist, however for an internal graphics display the packaging becomes more of a challenge. Then too also interfacing options are possible, however beyond the LCD/Keypad and use of on board video options on the given host to external displays, all the options add cost and complications.

KEYBOARDS

Now let's add some keyboards to the picture drive home the image of MIL-STD Data Modem or ALE communications without the use of an external PC and working from direct 12v D.C. power in the field, even batteries due to the lower power consumption requirements. My favorite Go Kit keyboards are the under $20USD SoundLogic Waterproof, Yellow, Roll-Up Portable full size USB keyboard that you can run down to Walmart and purchase.

Also, the $20USD wireless (latency issues?) keyboard which is also available in green, but not MIL-SPEC olive drab green.

Local and Remote User Interface Details

The planned user interfaces are detailed below, both the standard Remote interfaces (see items 4, 5 and 6) and optional Local LCD/Keypad interface (see items 1 and 2 below) are summarized.

The user interface options to be explored for the Data Modem Configuration and Remote Control and for OTA Data communications will tentatively be:

1. **Local:** Internal LCD/6-key Keypad display or External USB port LCD/6-key Keypad module. The USB port for attaching the LCD/Keypad may be physically specified as Rear, Bottom Left, USB port [1] = LCD/Keypad. The 6-key Keypad interface can be used for two-way comms by building messages live via individual character selection (AMD, DBM, DTM) or from pre-saved message selection and possibly from a full Dictionary mode similar to AQC-ALE Dictionary mode.

   NOTE: Letter Orderimg for Keypad character selection for message construction will be user selectable as: **A** *through* **Z** *or user selected in Zim Ordering*: **ETAONRISHDLFCMUGYPWBVKJXQZ** *where Upper or Lower case can be user selected. Numerals will be displayed in the order of* **1** *through* **0** *followed by punctuation symbols in the order of* **? - : ( ) . , ;** */ where a key sequence for selection between number and punctuation characters being displayed for selection will be provided. The message being entered will be displayed and scroll on lines 1 and 2 and be scrollable for editing. The dictionary of characters for the selection of message characters will be on lines 3 and 4. When the flashing enabled for message entering CURSOR is positioned on lines 1 or 2 the ARROWS will be for outgoing message scrolling and navigation to a point of editing. When the CURSOR is on lines 2 and 4 the ARROW keys will be for navigating message character selection. The use of the Enter and Exit keys will be multitasking and combined with durations pressed, all TBD. The 17th character of line 4, a SPACE character will insert a SPACE when selected and 18th character, a "D" will switch character dictionaries, 19th character, a n "S" will SEND the message and the 20th and last character on the line will EXIT the dictionary and dump the message. Below are two simulations of what is planned for implementation beginning with Upper Case Characters and Number selection:*

TEST DE NNN0WWL AR
NN_
ETAONRISHDLFCMUGYPWB
VKJXQZ1234567890 DSX

Lower Case Characters and Punctuation selection enabled:

Test DE NNN0WWL AR

NNNN_
etaonrishdlcmugypwb
vkjxqz?-:().,;/ DSX

NOTE: The extra features of the I2C interface for the LCD module will be developed when the hardware becomes available.

2. **Local:** Internal LCD/6-key Keypad display or External USB port LCD display and USB Keyboard module. The USB port for attaching the LCD/Keypad may be physically specified as Rear, Bottom Left, USB port [1] = LCD/Keypad. The USB Keyboard will be attached via any unused hub port. When a keyboard is attached the idea will be to only accept data from it for two-way comms use and not enable the 6-key Keypad interface. The first three lines of the display will be for the out going message where the last line will key count of characters entered and perhaps other parameters.

   NOTE: See notes in item 1 above.

3. **Local:** Graphical User Interface (GUI) via mini HDMI port (using proper HDMI/DVI-D or other adapter or straight to HDMI cable) connected Video Monitor, USB Keyboard and USB Pointing device with a GUI interface somewhat similar to that of the desktop PC-ALE tool may come about at some point. I know this is appealing to many, especially in the field or vehicle with a 12vdc Lilliput or sun visor monitor. However if a GUI interface comes about, it will not be for some time after an initial release takes place.

   NOTE: For development an HDMI/DVI-D adapter is being utilized for Linux OS GUI acccess as well as VNC GUI remote access. The HDMI/DVI-D cable is an inexpensive item available from: http://www.amazon.com/Aftermarket-Product-Converter-Universal-Transformer/dp/B008J7E69C

   NOTE: If an LCD/Keypad is installed all remote control functionality nay not be accessible when the GUI interface is active.

4. **Remote:** Ethernet based TCP/IP SSH connection via a Cross Over Cable or LAN connection with sockets for Remote Control and Data messaging.

   NOTE: If an LCD/Keypad is installed some or all remote control functionality may not be accessible when TCP/IP SSH Remote Control is enabled.

   NOTE: For Cross Over Cable support on a laptop or other PC with only Wireless support a USB to RJ-45 Ethernet adapter such as the Sabrent USB 2.0 to RJ-45 Gigabit 10/100/1000 Ethernet Adapter will be required.

   NOTE: For ALE it may be possible to use a USB to Wireless adaptor dongle, however they are not all Linux compatible, refer to: http://wireless.kernel.org/en/users/Devices/USB recommended: https://www.thinkpenguin.com/gnu-linux/penguin-wireless-n-usb-adapter-gnu-linux-tpe-n150usb

and https://www.thinkpenguin.com/gnu-linux/clearance-penguin-usb-bluetooth-micro-adapter-gnu-linux which are well known to work on Linux distributions.

5. **Remote:** Serial Asynchronous I/O via RS-232 with one port for Remote Control and one port for Data messaging. The ports may be physically specified as COM1 = Remote Control and COM2 = Data. A pair of Null Modem cables will be required for this interface method for all features.

   NOTE: All ports are TTL-UART thus RS-232 level conversion will be required for radios that are not TTL level.

   NOTE: COM1 can be used for Radio PTT via either the RTS line regardless of this being the active interface mode or not. When it is not being used as part of the current interface COM1 it can be used for CAT control for PTT. The alternate to this approach would be to make use of a GPIO pin for PTT.

   NOTE: COM3 is TX only, it can be use for CAT control and RTS PTT.

   NOTE: It may be possible to also make use of the "Serial Debug Port Header" interface via and FDTI or Prolific USB cable interface for serial remote control.

   NOTE: If an LCD/Keypad is installed some or all remote control functionality may not be accessible when RS-232 port Remote Control is enabled.

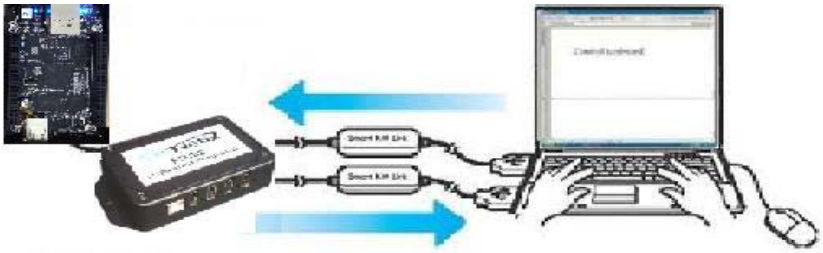
6. **Remote:** Serial Asynchronous I/O via the LS/FS/HS USB 2.0 Host Port using a USB 2.0 Bridged Cable.

   NOTE: Should MIL-STD data modem support be provided, 2 USB ports will be required for its stand alone use with one port for Remote Control and one port for Data messaging. A pair of USB 2.0 Bridged Cables will be required for this interface method for all features. To use the LS/FS/HS USB 2.0 Host Port a powered USB hub will also be required for MIL-STD data modem support. Make sure any use of a Hub is made by the Hub being plugged into the USB Host connector and powered on before powering on the BBB.

   NOTE: The HS USB 2.0 Client Port may also be supported for ALE remote control and data, in limited testing to date multiple instances of PuTTy have been used to connect via the USB Client Port.

   NOTE: If an LCD/Keypad is installed some or all remote control functionality may not be accessible when USB port Remote Control is enabled.

The ability to use a given interface selection above for OTA communications will be predicated on a combination of sensing any required device being attached and user configuration where the details are to be determined.



I still need to map out an entire interface tree and fallback scenario as to dealing with the lost of communications with any of the devices involved with any of the interface options as to which interface will come alive predicated on devices present even though not currently selected as the active interface. The only obvious path is when there exists an integral front panel LCD/6-key Keypad to fallback to for use interface support when another interface was in use, however its not standard feature as will COTS hardware modems.

To Be Determined

There are a number of issues that require solution which are being added to a developing TBD list of items at this early stage of the exploratory process aside from just the technical issues.

There are also distribution issues that are TBD as well, for one thing I am leaning toward a full turn key distribution to include the Debian Wheezy Hard image that would include the application software pre-installed and configured for particular hardware option configurations for all MARS uses to maintain system integrity and consistent performance while also locking out access to OS access to make changes. However for Amateur Radio use that may not be the case as, the user may be allowed to just use the factory Debian image and install the software and continue to have access to the OS for additional uses.