FCDEV3B User's Manual

Version 1.5, last edited 2017/10/01

1. Hardware setup

1.1. Power supply connection

The GSM modem chipset on the FCDEV3B is designed to run on battery power, typically a single Li-Ion cell which produces a voltage of 4.2 V when fully charged, lowering as it discharges. However, the chipset uses only LDO regulators and no switchers, hence when a fixed-output power supply is used instead of an actual battery, running at a lower voltage will result in less heat dissipation and thus greater reliability and longevity. The standard recommended voltage for fixed supply operation is 3.6 V.

The power input connector on the FCDEV3B is a Weidmuller 1510460000 (reference designator J305, the big orange connector readily stands out visually); there are several mating connector parts available from Weidmuller; Phoenix Contact also makes compatible connectors. Unless you have bought a ready-made power supply with your board, you will need to make your own power connection cable. If you are connecting a ready-made cable, the connector only goes in one way, but if you are wiring your own cable, you need to observe the polarity: the positive needs to be on pin 1 (facing inward), the negative needs to be on pin 3 (facing toward the edge of the board), and the middle pin is unused.

If you are connecting an actual battery rather than a power supply, presumably to test truly mobile operation with cell reselection, location updates and handovers, please note that the FCDEV3B will only draw power from that battery and has no provision for charging. Therefore, you will need to use an external charger and connect a fully charged battery for the duration of your mobility tests.

1.2. Serial port connection

The primary means of controlling, programming and communication with the FCDEV3B is a pair of low-voltage serial interfaces, also known as UARTs. Each of the two UARTs is natively a low-voltage interface (2.8 V native logic levels, tolerant of 3.3 V, but NOT any higher voltages) and can be converted externally to RS-232 or to USB as needed for each user's application.

With our standard firmware UART 0 (called MODEM in the Calypso chip documentation) provides a standard ASCII AT command interface (see Chapter 2), and UART 1 (called IrDA in the Calypso chip docs, although we are not using it for that purpose) provides a debug trace and command interface in TI's binary packet format (see Chapter 3). If you are going to be running your own firmware instead of ours, you can naturally use each UART however you like, but in any case you will need to have at least one UART connected in order to make the modem board do anything useful.

The dual UART header on the FCDEV3B is J301, and it has the following pinout (pin 1 is toward the power input connector):

GND	1	2	GND
TX_IRDA	3	4	TX_MODEM
RX_IRDA	5	6	RX_MODEM
NC	7	8	RTS_MODEM
NC	9	10	CTS_MODEM

If you are going to connect to both Calypso UARTs by way of an FT2232D USB dual UART, the convention established by the Mother of FreeCalypso is that Calypso's MODEM UART should be connected to

FT2232D channel A, and the IrDA UART should be connected to channel B. Connect jumper wires as follows:

FT2232D signal	Calypso signal
ADBUS0	RX_MODEM
ADBUS1	TX_MODEM
ADBUS2	CTS_MODEM
ADBUS3	RTS_MODEM
BDBUS0	RX_IRDA
BDBUS1	TX_IRDA

Be sure to connect the ground as well! The FT2232D adapter boards sold by FreeCalypso have two ground pins, one on the ADBUS side and the other on the BDBUS side, which naturally map to the two ground pins on the J301 dual UART header on the FCDEV3B.

If you use an FT2232D adapter as described above to connect the FCDEV3B's UARTs to a Linux PC or laptop, you will have a single USB device with two *ttyUSBx* devices behind it; if you have no other *ttyUSBx* devices, the Calypso's MODEM UART (the one we use for the standard ASCII AT command interface) will appear to your Linux host as `/dev/ttyUSB0`, and the Calypso's IrDA UART (the one we use for the debug trace and command binary packet interface) will appear as `/dev/ttyUSB1`.

1.3. Voice interfaces

The UARTs described in the previous section are sufficient for all data, SMS and ''hacking'' applications except voice. While the AT command interface provided by our standard firmware and accessible via those UARTs can dial and answer GSM voice calls, actually hearing or otherwise receiving the voice call downlink and sending your own voice into the call uplink requires additional interfaces. There are two voice interface options available on the FCDEV3B:

- The standard analog earpiece and microphone interfaces provided by the Calypso+Iota chipset have on-board loudspeaker driver and microphone input circuits connected to them on our FCDEV3B. You can connect a loudspeaker to two-post header J312 and an electret condenser microphone to two-post header J313, and then use those acoustic transducers to exercise GSM voice calls. See Appendix A for more information.
- The Calypso chip has a poorly documented provision for a digital voice PCM interface, and this interface is brought out to a header on the FCDEV3B. This interface has not been fully reverse-engineered yet and should be regarded as experimental at the present. See Appendix B for more information.

1.4. Antenna connection

The gold-plated SMA connector J308 that hangs over the edge of the board is where you need to connect the GSM antenna; the modem will not receive any GSM signals without an antenna connected. Alternatively one can connect a coaxial cable going to a base station simulator.

1.5. SIM socket

The SIM socket on the board is a C&K CCM03-3013 (reference designator J302); it accepts the 2FF classic SIM size that culturally corresponds to traditional GSM/2G.

1.6. Power on/off control

Having battery voltage present at power input connector J305 will not in itself cause the Calypso chipset to power on and boot up — it also needs to sense a low on the PWON control input. The FCDEV3B provides two ways to control the power-on sequence:

- When battery power is applied to the board but the green LED is off, pressing the PWON button will cause the chipset to power on and boot up, and the green LED will turn on.
- If the board is to be used in an unattended environment where requiring a human to press a button is not acceptable, one can put a shorting block on two-post jumper header JP1. This two-post header is wired in

parallel with the PWON button, hence putting a shorting block on it is equivalent to having the button permanently held down. If the PWON button is permanently held down or JP1 is shorted, the chipset will power on and boot immediately as soon as battery power is applied.

The green LED on the board indicates the on or off state of the chipset: this LED lights up when the power-on control in the Iota chip is in the switched-on state, and is off otherwise. Once the chipset has powered on and the green LED is lit, a return to the switched-off state can be commanded only through software, and if JP1 is shorted, a software power-off command turns into a reset operation as the switch-off will be immediately followed by another switch-on.

There is also a RESET button on the board that shorts the nTESTRESET signal to ground; pressing this button will cause the chipset to switch on and reset irrespective of its previous state.

2. AT command operation with standard firmware

Our boards are shipped from the factory programmed with a FreeCalypso firmware image that provides standard end user AT modem functionality with a full commercial product level of stability; at the present time this stable production firmware is FC Magnetite built in the *l1reconst* configuration, which is only partially rebuilt from source, with some components still being in the form of blobs for which we have no corresponding source. We have other firmware versions which are more thoroughly deblobbed (see Chapter 3), but they have not yet reached the level of functionality and stability required for end user product operation.

To use the FCDEV3B as a standard AT modem, connect the power supply, one or both UARTs (only UART 0 is needed for this mode of operation) and the antenna, insert a valid SIM with an associated active service from your local GSM network operator into the SIM socket, and press the PWON button or short JP1. The green LED will light up, the modem firmware will boot, and a working AT command interface will appear on the MODEM UART. You can then use your choice of terminal program on whatever host computer you are using to drive the FCDEV3B to talk standard AT commands to the modem; the default baud rate is 115200 bps and you should have hardware RTS/CTS flow control enabled. (If the host system you are using has no RTS/CTS flow control capability and you leave the CTS_MODEM signal on J301 unconnected, the pull-down resistor on the FCDEV3B will present a CTS-asserted state to the Calypso UART, and the AT command interface will work correctly as long as your host does not miss any output from the modem.)

To bring up the SIM and radio interfaces and connect to your local GSM network, your AT session should begin with the following commands:

AT+CMEE=2 —— enable verbose error messages
AT+CFUN=1 —— enable SIM and radio interfaces
AT+COPS=0 —— connect to the default cellular operator

Our TI-based firmware implements most of the standard AT commands defined in GSM specs 07.05 and 07.07, as well as a large set of non-standard ones added by TI, plus a few added by us. The only reliable documentation for TI's non-standard AT commands is the source code; we have found a few PDF docs describing some of them, but those docs are nowhere close to complete.

Our standard production firmware supports voice, SMS, supplementary services, CSD, fax and GPRS; for data and SMS functionality the standard AT commands should be all you need, but if you are going to be exercising voice call functionality, you will need to issue some additional non-standard AT commands of our own invention to enable the right voice interface (see §1.3):

- If you wish to use the on-board loudspeaker driver to hear the call downlink audio, you need to enable it as follows:

  AT@SPKR=1

  If the sound coming out of the speaker is too quiet (see Appendix A), you can try the following command to set the output level from the Iota chip to the maximum it will allow:

  AT@AUL="spkrmax"

- If you wish to use the digital voice PCM interface on MCSI pins (see Appendix B), you need to switch the voice path in the Calypso DSP as follows:

  AT@VPATH=2

2.1. Firmware version identification

You can query the running firmware for its version ID string with the **AT%VER** command; the same version ID string is also emitted as part of the boot-time debug trace output on the other UART (see Chapter 3) and can be seen in the firmware binary image with *strings*(1). This version ID string includes the firmware family name (currently Magnetite), the build configuration (currently *l1reconst* or *hybrid*) and the build timestamp.

2.2. Cleanly ending an AT modem session

If you wish to cleanly shut down the modem's activities on the GSM network and tell the network that you are signing off (IMSI detach), issue an **AT+CFUN=0** command. Once you have cleanly disconnected from the network, you can power the modem off with an **AT@POFF** command — the green LED on the board will go out. However, if JP1 is shorted, a true power-off is not possible and the **AT@POFF** will effect a reboot instead, by way of the VRPC block in the Iota chip going through a switch-off sequence immediately followed by another switch-on from the grounded PWON input.

Alternatively the **AT@RST** command can be used to effect a firmware reboot by way of a watchdog reset irrespective of whether JP1 is shorted or not.

3. FreeCalypso host tools and other firmware versions

Regular operation of the FreeCalypso modem via standard AT commands as described in the previous chapter does not require any special software — one can use any host computer or even a dumb terminal. However, if you have a GNU/Linux PC or laptop or any other Unix-based or Unix-like host system on which you can install FreeCalypso host tools, you get a whole bunch of extra toys to play with:

- A tool for flashing firmware images into the modem, so you can flash new firmware updates or images you have built yourself;
- Tools for decoding and displaying and/or logging the debug trace output from a running firmware, so you can see what the fw is doing when you AT-command it to perform this or that GSM operation;
- Tools for sending debug commands to the running firmware, so you can peek and poke registers and memory locations, enable additional debug traces and much more;
- Tools for manipulating the modem's flash file system;
- Many more hacks you can discover by studying the source code and docs included in the FC host tools package.

To gain all of these abilities, you need to download and install the FreeCalypso host tools package; the current version as of this writing is fc-host-tools-r7 and the installation involves compiling from source. The rest of this chapter will assume that you have FC host tools installed and working.

It also needs to be noted that our FreeCalypso host tools significantly predate FC hardware, i.e., most of our tools were developed long before we had any hardware of our own, when we were limited to hacking various pre-existing Calypso devices. Keeping this historical fact in mind will make it easier to make sense of much of the documentation included with the tools.

3.1. Reflashing the firmware with fc-loadtool

The utility for performing raw flash operations on FreeCalypso devices is *fc-loadtool*. On sensible devices like our FCDEV3B that have the Calypso boot ROM enabled by the board wiring, *fc-loadtool* can gain flashing access to the device through either of its two UARTs completely irrespective of its previous state, i.e., it is impossible to brick the board no matter what you do to its flash.

Out of the many things you can do with an FCDEV3B board, flashing with *fc-loadtool* is unusual in that it works exactly the same whether you go through UART 0 or UART 1 (`/dev/ttyUSB0` or `/dev/ttyUSB1` in the common FT2232D USB setup). Most other workflows work only with one UART or the other: for the standard ASCII AT command interface with our standard firmwares you need UART 0, and for the RVTMUX binary packet interface (*rvtdump*, *rvinterf*, *fc-shell*, *fc-fsio*, *fc-tmsh*) you need UART 1. But flashing with *fc-loadtool* is the unusual exception in that it works exactly the same through either UART.

If you are seeking to flash a new firmware release from FreeCalypso, the tarball package with the firmware update should contain a file named *flash-script*. It is a command script for *fc-loadtool* that contains the correct sequence of flash erase and flash program commands for flashing the new firmware image. To flash a firmware update using this included script, cd to the directory containing the *flash-script* file and the firmware image it references, and run a command of the following form:

fc-loadtool -h fcfam -B812500 */dev/ttyUSBx* flash-script

The **-B812500** option tells *fc-loadtool* to use the fastest serial baud rate of 812500 bps, which is a GSM-specific baud rate derived from the 13 MHz clock used in the Calypso and other GSM devices. This non-standard (outside of the GSM world) high baud rate is supported by the FT2232D adapter commonly used with FCDEV3B boards, but may not be supported by other kinds of serial hosts. To use the more standard 115200 baud rate, omit the **-B** option:

fc-loadtool -h fcfam */dev/ttyUSBx* flash-script

If you are using the PWON button to power on and boot your board, i.e., if you don't have a jumper on JP1, the most proper way to perform the above flashing operation is to have the board in the ''powered but switched-off'' state, i.e., with the power supply connected but the green LED off, run the *fc-loadtool* command in this state, and as *fc-loadtool* sends its beacons down the serial line, press the PWON button. The Calypso boot process will be interrupted and diverted at the boot ROM, *fc-loadtool* will feed our FreeCalypso *loadagent* to the latter and use this agent to perform the flash operations. The board will be automatically powered off at the end, i.e., the green LED will go out when the operation is finished.

If you need to perform an *fc-loadtool* operation when there is a jumper on JP1 or when the board is in a hung state, run the *fc-loadtool* command and press the RESET button on the board instead of PWON.

It is also possible to use *fc-loadtool* interactively, without running a script:

fc-loadtool -h fcfam */dev/ttyUSBx*

In this mode the tool will stop once it has got *loadagent* running on the target, and present a **loadtool>** prompt to the operator. You can then execute various commands including flash operations interactively; type *help* at the **loadtool>** prompt to get started.

3.2. Playing with the RVTMUX binary packet interface

While UART 0 carries a standard ASCII AT command interface with our firmwares, there is an additional binary packet interface called RVTMUX presented on UART 1. This interface is intended for development and debugging, not for end-use applications, i.e., the latter should NOT depend on having this second UART available, but for developers and tinkerers it enables the following capabilities:

- The firmware continuously emits a fairly large amount of debug trace output on this interface; this debug trace output can be captured, decoded from binary packets into ASCII and displayed and/or logged with our *rvtdump* utility.
- A number of debug, development and test mode commands can be sent to the firmware in the form of RVTMUX binary packets. GPF ''system primitive'' commands can be sent through *fc-shell* and are primarily useful for selectively enabling various more verbose debug traces in the G23M protocol stack. TM and ETM commands can be sent through *fc-tmsh*; many of them invoke L1 and RF test modes which you should **never** invoke unless you wish to operate a rogue transmitter or jammer, but some ETM commands like memory and register peeks and pokes can be useful in ordinary firmware debugging sessions.
- The modem's flash file system can be manipulated with full read and write access over RVTMUX using our *fc-fsio* utility.
- As a new feature added in FreeCalypso, AT commands can also be sent over RVTMUX, appropriately wrapped in RVTMUX binary packets; our *fc-shell* utility performs this feat. This mechanism can only be

used for voice and SMS AT commands which do not involve data connections; for CSD and GPRS you need to use the main AT command interface on UART 0.

3.3. Other FreeCalypso firmware offerings

As mentioned at the beginning of Chapter 2, our current stable production firmware is Magnetite *l1reconst*, which still has many components in the form of blobs for which we have no corresponding source. We also have two other firmware versions that may be of interest to more adventurous users who would be willing to give up some stability in return for having more source:

- Our Magnetite firmware can be built not only in the stable *l1reconst* configuration, but also in a more experimental configuration called *hybrid*. The *hybrid* configuration replaces the stable TCS211 version of the G23M protocol stack (which we have only as blobs with no corresponding source) with a newer version from TI's TCS3/LoCosto program, which we got in full C source form. This configuration constitutes a very significant deblobbing of a major firmware component, and it got its name because it is a TCS2/TCS3 hybrid. However, this hybrid firmware is currently considered experimental and not ready for production use because the new G23M and ACI versions have not had all of their bugs shaken out yet.
- We also have our Citrine firmware which chronologically predates Magnetite. Citrine firmware has the special distinction of being built with gcc (as opposed to TI's proprietary compiler) from source components only (no blobs), but it was one of our early attempts at putting the firmware together, and the Mother of FreeCalypso currently sees it as a dead end. The Mother has a plan to produce a new firmware version that will also be built with gcc from source components only (tentative planned name Selenite), but it will be derived from Magnetite *hybrid* rather than Citrine.

  Citrine uses the same versions of L1, G23M and ACI as Magnetite *hybrid*, but it does not have CSD or GPRS support, only voice and SMS.

3.3.1. Running firmware out of RAM without flashing

A special feature of our FCDEV3B hardware not present on most of the pre-existing Calypso devices is our large RAM (8 MiB) which allows any of our firmwares to run entirely out of RAM without flashing. Given that we have 3 different firmware versions of potential interest (Magnetite *l1reconst*, Magnetite *hybrid* and Citrine), you can put any of the three in flash and play with the other two by loading them into RAM.

When you build any of our firmwares from source, you can build either a flashable image or a RAM-loadable one. The two image types are not interchangeable: if you have a flashable image (`fwimage.bin` or `fwimage.m0` from a Magnetite build or `flashImage.bin` from a Citrine build), you cannot run that same image out of RAM, instead you need to build an appropriate RAM-loadable image: `ramimage.srec` for Magnetite or `ramImage.srec` for Citrine. (RAM-loadable images are always in SREC format.)

Once you have a `*.srec` RAM-loadable image, you can load and run it with the *fc-xram* utility. Just like flashing with *fc-loadtool*, the initial steps performed by *fc-xram* work exactly the same whether you go through UART 0 or UART 1, but your choice of UART for this operation needs to be made with the rest of the workflow in mind:

- If you are going to run the XRAM download over UART 0, your *fc-xram* invokation command should take the following form:

  fc-xram -h fcfam */dev/ttyUSBx* ramimage.srec

  The sequence of PWON or RESET manipulations needed to initiate the operation is the same as for flashing with *fc-loadtool*. Once the XRAM image is loaded, it will start running, and if the serial port used by the *fc-xram* process corresponds to UART 0, then the newly loaded firmware's ASCII AT command interface will appear when *fc-xram* drops into the tty pass-through mode. Of course one can also have an *rvinterf* process running in another window, waiting for RVTMUX debug trace output on the other UART.
- If you are going to run the XRAM download over UART 1, your *fc-xram* invokation command should take the following form:

  fc-xram -h fcfam */dev/ttyUSBx* ramimage.srec rvinterf

> Note the addition of *rvinterf* after the `*.srec` image name. This additional command line argument instructs *fc-xram* to pass the serial channel to *rvinterf* when the loaded image starts executing; such passing is appropriate when the XRAM download is done over UART 1, as the newly loaded firmware will immediately start emitting debug traces in RVTMUX binary packet format on this UART.

When run as shown above, *fc-xram* will use the standard 115200 baud rate for the XRAM image transfer. If you are using an FT2232D USB adapter or some other serial host that can support GSM-specific high baud rates, you can use the **-B812500** option to speed up the image transfer over the serial line.

4. Operation without FreeCalypso firmware

The FCDEV3B has been created for the primary purpose of running FreeCalypso software and firmware. FC firmware came first, running on various pre-existing Calypso hardware targets with the help of FC host tools, and then we have created our own FCDEV3B hardware in order to free ourselves from the limitations of those pre-existing Calypso devices. However, some users have expressed an interest in using our hardware to run our competitors' OsmocomBB software; this chapter addresses the needs of those users.

Because the codebase of OsmocomBB is not owned by us, we are unable to develop or provide our users with any actual code patches for OsmocomBB; instead we can only provide instructions in the form of written English prose, whereas any actual changes to OsmocomBB code will need to be made by someone who is in a more appropriate position to do so.

As of this writing, OsmocomBB does not have explicit support for the FCDEV3B target, however, OsmocomBB firmware built for the *gta0x* target is expected to work on the FCDEV3B with the following caveats:

- Unless you have modified the UART selection code in board/gta0x/init.c, the firmware will use UART 0 for communication, hence when you run *osmocon* to load and run your firmware, you will need to specify the */dev/ttyUSBx* device corresponding to UART 0.
- Do not attempt to flash OsmocomBB firmware into our board. While you won't cause any irreparable damage by doing so (our boards are unbrickable), any flashed OsmocomBB firmware simply will not work. OsmocomBB supports flashing and running from flash only on Compal targets (Motorola C1xx) but not on classic Openmoko-style ones; our hardware is in the latter category. Instead you will need to load the tiny OsmocomBB *layer1* or other firmware image on every boot via *osmocon*.
- Just like the original Calypso modems by Openmoko which we sought to replicate, our boards are shipped with a file system structure in a non-firmware region of the flash — a flash file system (FFS) in TI's format. This FFS contains a full set of RF calibration tables: the correct APC DAC values to produce the set of Tx power levels prescribed in the GSM 05.05 spec, the ''magic gain'' constants for each band needed to derive the actual received signal level from DSP power measurements, the per-channel variations in this ''magic gain'' (RSSI channel compensation), and the characteristics of the VCXO for the AFC algorithm. All of these values are precisely calibrated for each individual unit on the factory production line, which is why they cannot be hard-coded in any firmware and must be stored dynamically in a flash file system instead.

  However, OsmocomBB software does not know how to read these factory RF calibration values out of FFS, neither on our FCDEV3B nor on the original Calypso GTA0x modem by Openmoko, and uses hard-coded values for everything instead. The Tx power levels are the most problematic: the hard-coded values they use have been measured on some Motorola phone and thus stand no chance of being anywhere close to correct on Openmoko or FCDEV3B hardware that uses a different RF PA, and they only have one table for the 900 MHz band which is also erroneously used for the 1800 MHz and 1900 MHz bands. Thus if you run OsmocomBB software on our board with Tx enabled, your Tx power levels will most likely be wildly out of spec.

  If anyone wishes to endow OsmocomBB with the ability to read and make use of our (or Openmoko's) factory RF calibration values, we would be glad to provide detailed guidance in doing so, but the actual OsmocomBB coding work would need to be done by someone in the OsmocomBB camp — not us.
- If you are going to be using voice call functionality and you wish to use the on-board loudspeaker driver to hear the call downlink audio, you will need to modify the code in board/gta0x/init.c to configure Calypso GPIO 1 as an output and to drive this output high — the equivalent of the **AT@SPKR=1**

command one would issue when using FreeCalypso firmware.

4.1. Flash memory usage

The flash memory chip on the FCDEV3B is Spansion S71PL129NC0HFW4B, combined with pSRAM in the same package. This memory chip provides two flash chip select banks of 8 MiB each; on the FCDEV3B these two flash chip selects are wired to Calypso chip selects nCS0 and nCS2. (Calypso nCS1 is wired to the external RAM like in all other known phone and modem designs.)

The flash bank on nCS0 is the bootable one, and is used to hold our flashable FreeCalypso firmware images. If you are only interested in OsmocomBB and not FreeCalypso firmware, and if you would like to have the Calypso boot ROM wait forever for an external serial download (*osmocon*, *fc-iram*, *fc-xram* or *fc-loadtool*) instead of booting our FC firmware, you can simply erase this nCS0 flash bank, i.e., make it blank flash. However, you will need to use our *fc-loadtool* utility from the FC host tools suite (§3.1) to do so, as OsmocomBB does not know how to operate on AMD-style flash, only the Intel type used in Compal phones. If you erase our FC firmware from the flash in this manner, you will still be able to load and run it in RAM via *fc-xram* — see §3.3.1.

The other flash bank on nCS2 is not bootable, and we have allocated the first 2 MiB out of this 8 MiB flash bank for our factory-initialized flash file system which contains the per-unit RF calibration values among other data. This FFS is initialized at the factory and is subsequently accessed for both reads and writes by our official FreeCalypso firmwares, but a non-FreeCalypso firmware that is only interested in the RF calibration values can use a much simpler read-only implementation.

We strongly advise against erasing or repurposing this 2 MiB FFS partition at the beginning of the second flash bank on nCS2. Even if your current software has no ability to make use of these data, you should never throw away factory RF calibration values.

Appendix A: Analog loudspeaker and microphone

TI's Iota ABB chip (part of the Calypso+Iota chipset around which our board is built) is designed to drive 32 Ω earpiece speakers directly, without needing any external active components, but it cannot do the same for 8 Ω hands-free loudspeakers: the ability to drive such loudspeakers directly was added in TI's later chipsets, but is not present in the Iota ABB targeted by FreeCalypso. On TI's own D-Sample development kit which is built around the same Calypso+Iota chipset as our FCDEV3B, the analog voice downlink output from the Iota ABB goes to an external audio amplifier, and the output of that amplifier goes to the speaker in the handset part of the kit; the resulting acoustic volume is loud enough that one can comfortably exercise voice call functionality without lifting the handset out of its cradle.

The design intent for the FCDEV3B was to accomplish the same effect: to be able to connect an 8 Ω loudspeaker to the board and have it be loud enough that voice calls and various Calypso DSP audio features (such as ringtone melody generation) can be comfortably exercised from a lab bench, with the operator sitting near the board, but not needing to bring his or her ear up close to it. Toward this goal our FCDEV3B has an external audio amplifier circuit inserted between the EARN&EARP output from the Iota ABB and the actual loudspeaker connection header J312; because we could not find a copy of the D-Sample schematics, this loudspeaker amplifier circuit has been copied from TI's Leonardo schematics instead.

Unfortunately the Mother of FreeCalypso is not an expert in the field of analog audio and speakers, and so far we have not been able to reproduce the acoustic loudness produced by TI's D-Sample kit. So far the only actual loudspeaker part we have tried connecting to the J312 header on our board has been Soberton SP-1510, picked essentially arbitrarily from Digi-Key, and the choice of the feedback resistor values in the amplifier circuit on the board (which control the gain of the amplifier) has been based on a rather weak understanding of what actually happens in this circuit. With this current choice of feedback resistor values and the physical loudspeaker part, and with the output level from the Iota ABB turned to the maximum with **AT@AUL="spkrmax"**, the voice call downlink can be heard OK if you bring your ear close to the speaker, but is very quiet otherwise.

The Mother's plan is to hire an audio and speaker expert to help us figure out what loudspeaker part we should use and what changes we should make to the amplifier circuit (if any) to bring the acoustic loudness to par with what TI's D-Sample produces with the same Calypso+Iota chipset with the same power supply. Until then, we encourage our users to experiment with different speaker parts; the only constraint is that the impedance needs to be 8 Ω or greater. A workable solution may be to use a longer wire run between the connector mating with J312 on our board and the actual speaker, so you can bring the speaker to your ear while the board remains in its place on your lab bench.

Our board also features a microphone input circuit copied from TI's Leonardo schematics; the actual microphone needs to be of the electret condenser type, and needs to be connected to two-post header J313; there is a **+** mark on the silk screen indicating the correct polarity.

Appendix B: Digital voice interface via MCSI

The Calypso chip features an auxiliary 4-wire synchronous serial interface (data in each direction, bit clock and frame sync) called MCSI. This interface is connected to the DSP part of the Calypso and is controlled by the DSP code (ROM or downloaded patches); the ARM part of the Calypso where regular firmware code runs has no direct control over this interface. The DSP ROM code in the Calypso silicon version we are using supports a mode of operation in which MCSI becomes an external digital voice channel interface; TI called it the Bluetooth mode because they used it in Bluetooth-enabled phones to connect the digital voice channel between the Calypso and their Bluetooth Island chip.

The MCSI signals (4 signal pins plus ground) are brought out to a 5-pin header on the FCDEV3B, but the detailed workings of TI's ''Bluetooth'' digital voice channel are not documented anywhere and remain to be reverse-engineered. The MCSI header on the FCDEV3B is J311, and it has the following pinout (pin 1 is toward the edge of the board, away from the power input connector):

1	MCSI_CLK
2	MCSI_RXD
3	MCSI_TXD
4	MCSI_FSYNCH
5	GND

The signals have 2.8 V native logic levels and are tolerant of 3.3 V, but NOT any higher voltages. MCSI_RXD is always an input to the Calypso, MCSI_TXD is always an output from the Calypso, but the direction of the bit clock and frame sync signals is determined by the DSP code: in pure hardware terms they are bidirectional, but given the level of difficulty in developing custom DSP patches, their direction is fixed in practice by the DSP code in use.

Our FreeCalypso Magnetite firmware provides an extended AT command (**AT@VPATH**) to set the digital voice path routing mode in the DSP; setting **AT@VPATH=2** enables what TI called the Bluetooth headset mode, in which the digital voice path is connected between GSM and the ''Bluetooth'' voice channel on MCSI. Experiments with oscilloscope observation show that when this mode is enabled, MCSI_CLK and MCSI_FSYNCH become outputs, putting out a 520 kHz bit clock and an 8 kHz frame sync, respectively. However, more investigation is needed before this digital voice interface can be put to practical use:

- The format of voice PCM data put out on the MCSI_TXD pin and expected on the MCSI_RXD pin remains to be determined experimentally.
- There appears to be a bug in the DSP code (present in the ROM version and not corrected by TI's TCS211 patches we are using) that causes the interface to unexpectedly shut down under certain circumstances (the clock output disappears); the exact triggering condition needs to be better understood before a reliable workaround can be implemented.

When it comes to this not-yet-understood Calypso digital voice interface, our FCDEV3B should be seen as a development platform to facilitate its reverse engineering.

Appendix C: Calypso JTAG

The JTAG header on the FCDEV3B is J310, and it has the following pinout (pin 1 is toward the power input connector):

TMS	1	2	nTESTRESET
TDI	3	4	GND
Vio	5	6	NC
TDO	7	8	GND
TCK return	9	10	GND
TCK	11	12	GND
EMU0	13	14	EMU1

All JTAG signals have 2.8 V native logic levels and are tolerant of 3.3 V, but NOT any higher voltages. The JTAG interface is needed very rarely; it is not needed at all in normal usage and even during intensive firmware development a need for JTAG arises only very rarely.