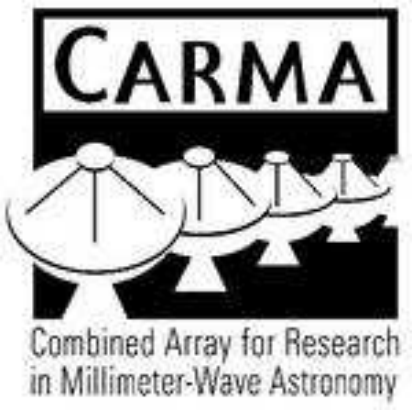

CARMA Memorandum Series #29

IF Module Software Documentation

A. D. Bolatto

Radio Astronomy Lab, University of California at Berkeley

August 9, 2004

ABSTRACT

This is a functional description of the IF module, explaining what it measures and controls, and how it controls it. The overall structure of the software as well as the details of the algorithms used are discussed in detail, so that programers and users not familiar with the embedded XAC code understand its features and nuances. This document also contains information useful for diagnosing hardware/software problems with the module, including a detailed explanation of the error codes and their possible causes.

Change Record

Revision	Date	Author	Sections/Pages Affected
	Remarks		
1.0	2004-July-27	A. Bolatto	1-19
	Initial draft discussing release 1.3.1.B of the software. Comments are welcome.		

1. Overview

The CARMA IF module consists of a variable gain microwave amplifier (the PAM, a name that comes from its use in the Allen Telescope Array as the post–amplifier module) integrated with a DAQcore monitor and control module such as that described in CARMA memorandum #13. The IF module is connected to the laser module (the OTX) which sends the IF over optical fiber back to the correlator in the lab. It is also connected to the IF switch, which selects among 4 possible inputs (i.e., 4 receivers). The role of the IF module is to provide up to 50 dB of gain from $\sim$ 300 MHz to $\sim$ 8 GHz, to control the gain of the PAM over a range of 63 dB using digital attenuators, to measure the output RF power of the PAM, to monitor the physical temperature of the PAM, the temperature regulation of the laser transmitter, and the output optical power, and to control the IF switch. It also has a spare connector that can be used for future expansion, in particular the control and monitor of a chopping tertiary for single–dish total power measurements.

The IF module communicates with the rest of the system using CANbus. The command and control interface for the IF module is defined in API # 224. This interface defines the automatic blanking frame reporting packets, and allows the user to access all the functionality of the module, including, for example, to set its output power to a particular value. Most of the API functionality is also accessible using the serial port RS232 connection and a terminal program in a PC.

This document is a functional description of the IF module focused on how to use it. It does not include a detailed description of the hardware except when it is pertinent for understanding how the software works. The purpose of this document is to serve as a guide for the higher level programs that use the module, and to help whoever is trying to debug/modify the embedded XAC software.

2. Software Structure

The IF module does not use the ORTOS software developed at OVRO, but a combination of code developed by me, and a subset of CANbus communication libraries extracted from the ORTOS system by David McMahon. The software is considerably more compact than the standard ORTOS, and is documented using Doxygen. The `main` file contains the code in the body of the program, which takes care of the one–time module initialization and the main loop which processes incoming CANbus/serial messages and outputs the blanking/monitor and fast frames. The CANbus communication libraries derived from ORTOS are found in the `candvr`, `canmsgs`, `errors`, `genrtn`, and `interrupt` files. The `daqio` file contains the

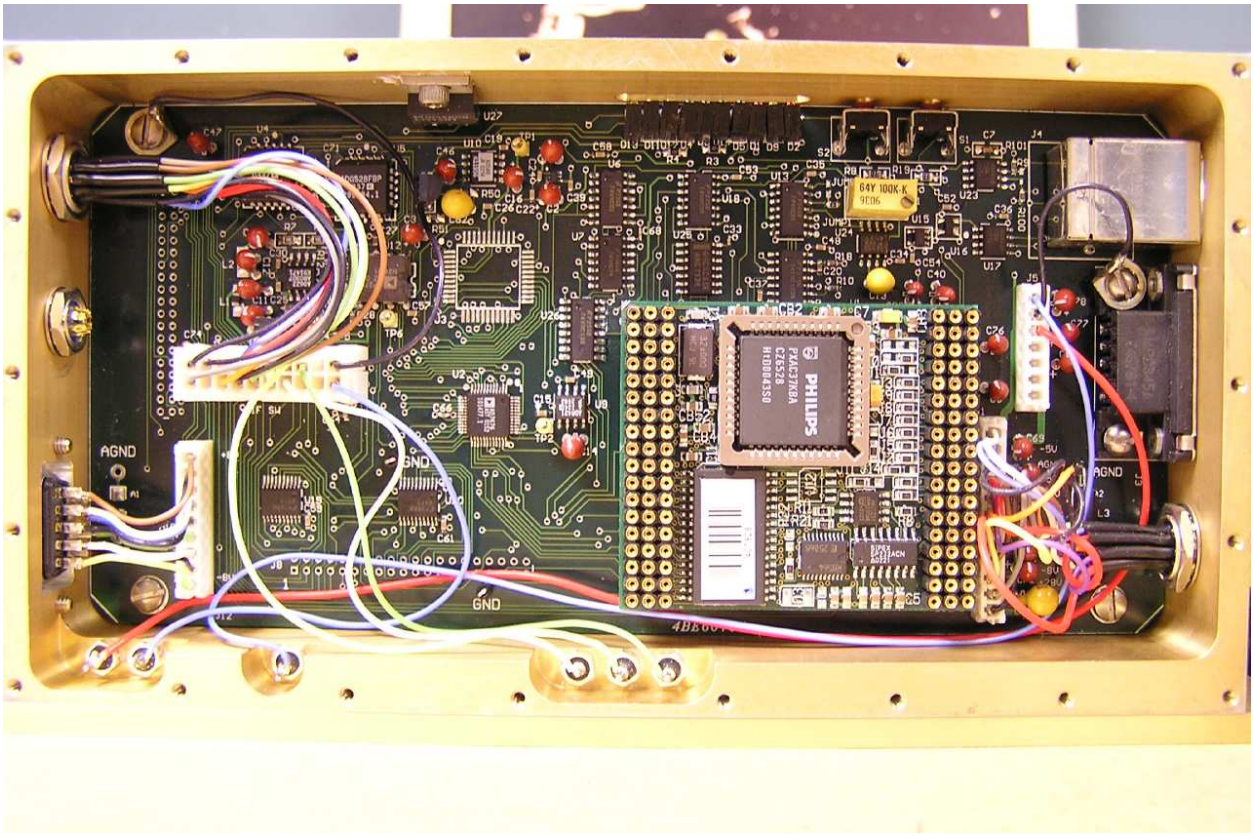

Fig. 1.— DAQcore side of the IF module, showing the Phycore XAC module and the connector and cable arrangement.

part of the code relevant to the DAQcore module itself, such as the A/D and DIO libraries. The `messages` file handles the higher level I/O and the parsing of the RS232 interface. The `pamdrv` file contains the libraries that deal with the PAM hardware, most notably setting the attenuators. The `1wireiddrv`, `spi`, `timerdrv`, and `uart` files have the libraries that deal with DS2401 1-wire ID devices, SPI communications (for the onboard EEPROM), and the internal XAC timers and UART. Finally, `tasks` has the routines that implement each of the API commands and responses, as well as the error logging and the proper initialization of most variables.

A key feature of the software in the IF module is that there is a constant background digitalization of the 4 analog input channels (RF power, PAM temperature, OTX temperature regulation, and OTX power, connected respectively to channels 0 to 3 of the 16–bit/8–channel A/D in the DAQcore) at a rate of 200 Hz (the RF power detector circuit time constant is $\sim$ 20 ms, so this is faster than Nyquist sampling). Every 5 ms the XAC is interrupted and the data acquisition routine samples channels 0 to 3. These samples are stored in a ring-type buffer, where they are used by the different routines. For example, the blanking frames report the averages of the last 100 samples of the different channels, while the query RF power command (message ID 0x105 in the API) reports the median of the last 3 samples. This way, when a fast sampling command is issued, the mode of operation of the module does not change at all. The only difference is that the individual samples are reported in the corresponding fast sampling packets defined in the API. The interrupt routine also increments a counter (the tick count), providing a clock for the system.

The overall architecture of the software is as follows: `main.c` contains the initialization sequence and the main loop, which executes forever. At the same time the data acquisition interrupt is active in the background, waking up every 5 ms, acquiring the data and updating the clock ticks. When the main loop determines that the tick count is close to a multiple of 100 (i.e., an integer number of 0.5 s periods since the last set time command was received) it enters the section that takes care of the blanking, slow monitor, and fast sampling reporting. Otherwise, it checks to see if there are any CANbus messages in the queue or if a keystroke has appeared in the RS232 UART. If there are messages, it processes them with the code found in `tasks.c`. Otherwise it keeps looping.

3. User Interface

3.1. LEDs and pushbuttons

The IF module is equiped with two pushbuttons and several LEDs, which are labeled in the frontpanel (Fig. 2). The buttons have the usual functions of RESET and BOOT, familiar to anybody who has worked with the Phytec XAC module. The LEDs are divided in three groups: green power LEDs, which should always be on, red LEDs indicating the BOOT or RESET lines are active (either because someone is pushing the button, or because the RESET distributed with the CANbus cable is pulled down), and yellow LEDs. One of the yellow LEDs, labeled `2pps`, indicates the operation of the DAQcore hardware timer and will always be off in the current version of the software. The other 3 yellow LEDs, labeled DO12, DO13, and DO14 (connected to digital output bits 12, 13, and 14 of the DAQcore) have the following meaning: DO12 is turned on while the module is receiving a message, either from the CANbus or the RS232 interfaces, otherwise is off. DO13 is turned on while the module is transmitting a message through the CANbus or RS232, otherwise is off. DO14 has a dual purpose: a (mostly) solid light is used to indicate an error has been logged (clearing the error log will turn off the LED). However, whatever the status of DO14 is it should blink twice a second while the blanking frame code is executing.

3.2. RS232 terminal

The serial port communications are programmed at 38400 bps, no parity, 8 data bits, and 1 stop bit. Most API commands can be issued through the RS232, and the blanking frame/slow monitor packets are echoed there in a “human friendly” format. The automatic echo, which

Fig. 2.— IF module front pannel detail, showing LEDs and pushbuttons.

may be annoying, can be toggled on and off by sending an 's' character. Sending a 'z' character allows the user to dump a range of XAC memory addresses for software development/debugging purposes.

To issue a command, the user has to send an 'x' character to grab the attention of the software (which sometimes requires more than one try). If the user has been successful the IF module will reply with a `CODE:` prompt. The module stops reporting during RS232 input, although the background samplig proceeds normally. The input process in the RS232 parallels the definition of the messages in the API: in fact the program just assembles an 8-byte packet with the information provided and sends it to the main "packet parsing" routine which is used for CANbus communications too. The user reply to the `CODE` prompt should be the hexadecimal packet ID as defined in the API (e.g., 100 to select a band using the IF switch). After entering the command code, the module will reply with a request for the rest of the parameters that apply to that particular command using the nomenclature defined in the API (in the example above, it will reply with `IFSWIPOS=` and expect an integer from 1 to 4). Commands that require security payloads (e.g., command 0x000 for module reset) expect them in hexadecimal notation, as they are specified in the API. There is a built-in timeout of a few seconds that is reset every time a character is sent over the serial link — the effect of timing out is typically that that input is set to zero, so if one times out at the `CODE` prompt the module interprets it as a RESET command and requests the security payload (do not despair, it will not successfully reset unless the payload is entered correctly! Just press ENTER 8 times). This "feature" is midly annoying and I plan to correct it in future versions.

There are two negative command codes that the current software will accept which do not belong to the API and are used only for debugging purposes. Code -1 will set the flag that causes the main loop to do a "blanking frame" (in the current version of the software this does not provoke a blanking frame, but it used to and may again in the future). Code -2 prints the current value of the internal module tick clock updated by the data acquision interrupt (a 2-byte integer that shows the data acquisition is running).

4. Module Initialization

During the power cycle startup, or recovering after a hardware/software RESET, the module goes through an initialization process. Very early in this process the serial port communications are initialized, and a wealth of information is echoed throught the RS232. Once the module reports its software version, it sets the PAM internal attenuators to their maximum (63.0 dB), resets the IF switch position to select band 1, and reports the module ID as it has

been stored in EEPROM. There is no attempt to check for the validity of the ID number; if the module has not been some time in the past initialized using an assign ID command (message ID 0x3FE in the API) it will report a bogus ID.

Next in the initialization sequence, the module attempts to read in the RF power sensor calibration from its EEPROM, which consists of a table of sensor voltage–RF power pairs of length ≤ 16. The calibration is stored in EEPROM together with a checksum to help the software determine its validity. If the software finds it to be valid, the table of voltage–power pairs is echoed through the serial port. Otherwise the module reports `INVALID RF POWER SENSOR CALIBRATION` and uses a default 1,000 mW/V factor. This default factor is not accurate. In fact, it is blatantly wrong to emphasize the fact that the calibration is invalid.

In the final part of the initialization the software activates the background sampling, waits for 3 samples to be acquired, and uses the median of channel 0 (the RF total power) to estimate the offset in the A/D input amplifier (typically only a few counts, where 1 count = 76 μV). Of course, this works well only because the attenuation is set to maximum and the module is (hopefully) not receiving some overwhelmingly strong signal at its input. The offset found is also reported in the RS232 terminal.

5. Blanking Frame Activity

The first time the module turns on it will wait for 100 ± 1 ticks of the internal clock (500 ± 5 ms) to send the first set of blanking and monitor frames. This set will have the "initialization request" flag turned on (byte 3 of the slow monitor packet # 1), which will be turned off afterwards. The only means of synchronization of the module are the SET TIME commands (message ID 0x001) issued periodically by the antenna computer: when the module receives one it will reset the internal tick count and set the module time according to the contents of the packet. The assumption built into the software is that these packets will only be sent on blanking frame edges, and the module uses that fact to synchronize with the rest of the system. Essentially, it sends blanking frames whenever the remainder of the internal tick count divided by 100 is ± 1, and it sends slow monitor packets every 10 blanking frames. The other built–in assumption is that the antenna computer will issue SET TIME commands at least once every 5 minutes (the 2–byte counter allows the module to stay synchronized for 327 seconds since the last SET TIME).

After the blanking/monitor frames are sent the software checks if fast sampling is enabled, and if it is it proceeds to assemble and send the corresponding packets. For each fast-sampled channel there will be 50 packets sent in rapid succession inmediately after the

blanking/monitor frames.

6. Algorithms

6.1. Fast Sampling

As discussed in §2, the digitalization always proceeds in the background at the fast sampling rate. The START FAST SAMPLING command only sets a flag telling the software to report the individual samples (rather than only their average) in the next blanking frame edge. As a consequence, this module will always report 100 samples for the entire blanking frame interval, independently of precisely when the fast sampling command was issued. If this is a problem for the software dealing with the piecing together of the fast sampling reports, it could be changed.

6.2. Set Band

The SET BAND command accepts as a parameter an integer from 1 to 4. Numbers outside this range trigger a `BAND_ORANGE` error. If the current switch position, as sensed by digital inputs DI0..DI3, corresponds to the commanded position, no action is taken. Otherwise the blanking frame switch status is set to `SWITCH_CHANGING`, the switch is commanded to the new position, the module waits for 20 ms for the relays to close, then senses the status of the switch in DI0..DI3. If the sensed position does not agree with the commanded position, a `BAND_STUCK` error is logged, and the switch status in the following blanking frame is changed to `SWITCH_STUCK`.

In the current implementation there is the risk that the module will miss the blanking frame reporting during the 20 ms wait, if the switch is commanded to move very close to a blanking frame edge. Since changing bands is not done that frequently, and not done at all in the middle of integrations, this is probably not a big problem. It could be fixed by increasing the time tolerance of the blanking frame reporting. Other fixes are possible, but considerably more complicated.

6.3. Set PAM Attenuation

This command accepts as a parameter a floating point target attenuation in dB, from 0.0 to 63.0. If the target attenuation is within 0.3 dB of the current attenuation, the command

is ignored and no action is taken (the granularity of the attenuator settings is nominally 0.5 dB). If the target attenuation is greater than 63.0 dB, or less than 0.0 dB, it is reset to the corresponding limit and a `ATTEN_UFLOW` or `ATTEN_OFLOW` error is logged, but the command proceeds. The target attenuation is them rounded to the nearest 0.5 dB step, and equally divided among the input and output attenuators of the PAM (if the number is not even the extra 0.5 dB of attentuation will be applied in the output attenuator). At the time the attenuators are set, the blanking frame pam status is set to `PAM_CHANGING`.

6.4. Set Input and Output Attenuators Independently

This command accepts two floating point numbers between 0.0 and 31.5, representing the attenuation in dB. This command proceeds very much as the previous command, logging similar errors and setting the blanking frame PAM status is set to `PAM_CHANGING`. The only procedural difference is that the command is never ignored, and the attenuators are always set, without regard for their current setting.

6.5. Set PAM Level

The SET LEVEL command accepts one floating point parameter, the target output power level in mW. The first action taken is to measure the current RF power level by using the median of the last 3 samples in the data acquisition buffer. If the measured level is within 0.3 dB of the target level, no further action is taken. Otherwise, the module broadcasts a PAM status packet (message ID 0x130 in the API) with the content `PAM_CHANGING`, and it proceeds to iterate to obtain the requested output level. The iterations proceed in the following manner: if the current output power is zero, it will decrease the attenuation until an RF power greater than zero is measured. If this never happens, a `LEVEL_ORANGE` error is logged and the PAM blanking frame status is set to `PAM_IFLOW`. Otherwise the routine uses the measured RF power value at the current attenuator setting to forecast the attenuator setting necessary to achieve the required output power, and iterates repeating the process until the measured output power is within the 0.3 dB tolerance. Currently the maximum number of iterations allowed is 5.

At the end of the procedure a new PAM status packet is broadcast. If the iterations were successful and the measured output power is within 0.3 dB of the target power, the content of the message is `PAM_IFVALID`. If the set level procedure does not reach the target level within the tolerance in the maximum number of iterations, a `LEVEL_ORANGE` error is logged and

a pam status message `PAM_IFLOW` or `PAM_IFHIGH` is broadcast as the PAM status message. Because the attenuators have been changed, the pam status reported in the next blanking frame will always be `PAM_CHANGING`. The step by step process is echoed to the RS232 terminal to help the user debug problems.

In any case, because the set level command takes a while to execute, there is the possibility that the module will miss the blanking frame edge and not report (as in the set band command). The cures for this problem are limited: every measurement of the RF power takes $\sim 15 - 20$ ms (it uses 3 samples) and there may be up to 10 of those steps plus the software overhead and the overhead in echoing the intermediate steps to the RS232. The worst case execution time is probably ~ 350 ms (it could be cut to $\sim 50\%$ by reducing the amount of RS232 output and doing other minor modifications), which is the better part of a blanking interval.

Since the RF calibration tables for the detector were added to the software, I have seen that sometimes SET LEVEL fails to converge when it is started with a large attenuation in the PAM. The causes for this behavior are not clear, but it may be related to the fact that the detector calibration shows it is nonlinear. In any case, in my experience, if the set level command is reissued the procedure succeeds almost immediately, which suggests that the higher level software should just try again if set level fails to converge at any given time.

6.6. Query RF Power

This command simply waits for 3 new samples to be acquired, and returns the median (in mW) in a message with ID 0x142. All the RF power measurements (e.g., this command or the measurements done for setting the level) use a linear interpolation on the calibration table (e.g., Fig. 3) to convert the voltage measured by the A/D in channel 0 to the output power.

6.7. Calibration Table Upload

A new RF detector calibration table can be uploaded at any time. Once the upload is completed it will be automatically burned into the EEPROM, replacing the old calibration table. The upload procedure is separated in two parts: to initiate the upload, the module has to receive a message with ID 0x3F9, containing a 7–byte security payload plus a 1–byte table length, between 2 and 16. After that the module will accept packets with ID 0x3FA containing the individual voltage–power (floating point V–mW) elements in the table. Once

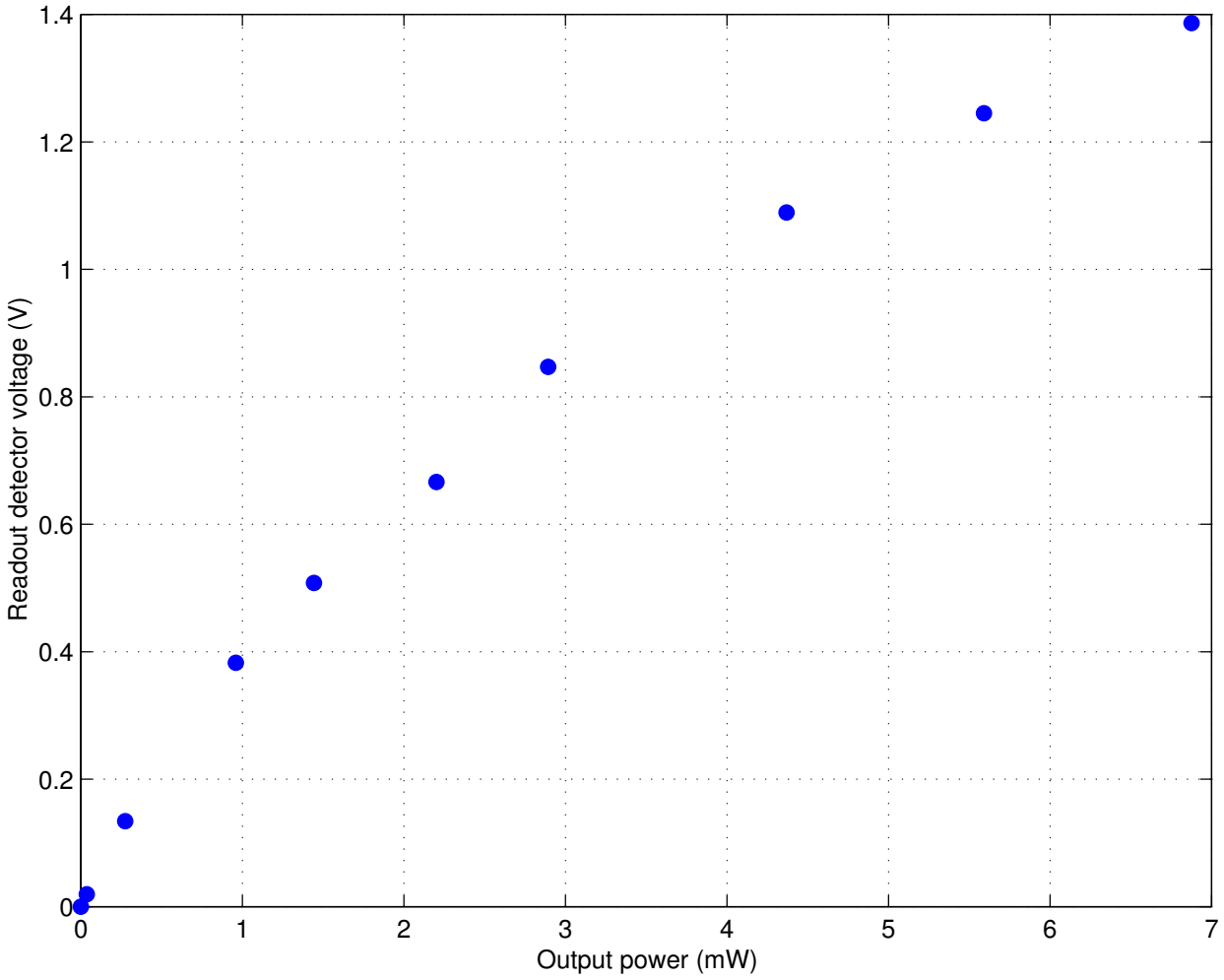

Fig. 3.— Calibration curve for the IF power detector installed in PAM #12.

the module receives a packet count equal to the table length it proceeds to compute a checksum (using the 1–wire DS2401 algorithm) and burn the table into EEPROM, including the checksum. When the EEPROM reports that the burn has finished, the values are read and compared to those in the XAC memory to check their accuracy. If discrepancies are fund, the module logs an `UPLOAD_BURN` error. The upload can be restarted at any moment before the EEPROM burning stage. If a packet with ID 0x3FA is received when the upload process is not active, a `UPLOAD_NOACTIVE` error will be logged.

I have seen the module hang occassionally at the end of the EEPROM burning process. This is apparently caused by the onboard ATMEL EEPROM failing to set a bit high in the SPI interface. Every time this has happened it turns out that, after reset, the table was correctly burn in memory. In future releases I may try to see if this problem is causes by a timing mistake in the SPI interface, or to include a timeout to prevent the hang up.

6.8. ID Request

This module replies to an ID request (message ID 0x3FC) with module type 224, API number 224, module serial number as assigned by the assign ID command (currently modules 1–9 belong to the SZA, and 10–15 to the OVRO antennas, for example), and dongle location 1. In the future, the donge location (which indicates polarization, 1 or 2), will be obtained through the power supply connection. At the time of this development those modules were not ready, and since we currently have only one polarization the dongle location was fixed in the program.

6.9. Status Words

Three status words are reported in the blanking frame packets, in bytes 4, 5, and 6 of message ID 0x111. They correspond to the status of the PAM, the IF switch, and the OTX. The actual numerical values of the different states are documented in the API # 224 document.

6.9.1. PAM Status

The PAM status byte is put together using two pieces of information: the IF output power status, and the physical temperature of the PAM, both averaged during the last 0.5 seconds. The reported PAM status is determined in the following manner: if the attenuators have been changed during the last 0.5 seconds, The IF output power status reported will be

PAM_CHANGING. Otherwise, the module will determine if the output power was within range (0 to +5 dBm, or 1 to 3.2 mW), and report PAM_IFVALID, PAM_IFLOW, or PAM_IFHIGH accordingly. Next, the software will check that the PAM temperature averaged over the blanking frame interval was between 293.0 and 303.0 K, and bitwise OR the PAM_TORANGE value accordingly. Note that band changes are not reflected on the PAM status byte, although the power readings will be meaningless if the band was changed during the last blanking frame interval.

6.9.2. Switch Status

If a successful set band command was issued during the last blanking frame interval, the reported status will be SWITCH_CHANGING. If the switch was determined to be stuck by the set band command (i.e., the sensed position did not correspond to the commanded position) the status will be SWITCH_STUCK and remain SWITCH_STUCK until a successful set band command is issued. Otherwise, at every blanking frame edge the switch position is sensed and reported in the switch status word as SWITCH_BAND1..4.

6.9.3. OTX Status

As in the PAM status, the optical transmiter status is broken into two parts, reflecting the state of the output optical power and the temperature regulation. If the laser power output readout averaged over the last 0.5 seconds is outside the 0.3 to 1.3 V range, a LASER_OPTORANGE value will be reported. Similarly, if the laser regulation error output average voltage falls outside the range 0.0 to 1.0 V a LASER_NOREGUL value will be reported. These values are bitwise ORed to determine the reported OTX status word.

7. Error Handling and Failure Diagnostics

Tables 1 and 2 show the error codes reported by the module, the conditions that cause them, the actions taken by the software, the parameters reported in the error log, and the likely causes of the hardware errors. The philosophy has been to log an error when something appears wrong with the hardware in the module (e.g., ID_NOTPRESENT signaling that the laser 1–wire ID dongle is not plugged in), or when there appears to be a human error in the use of the interface (e.g., UNKNOWN_CODE signaling a packet with an unrecognized ID, not in the API). A failure to meet a status check (e.g., the PAM physical temperature is above a

threshold) is not an error, and is not internally logged (only reported in the blanking frames).

As an aid to the used, byte 7 of the second blanking frame packet contains a count of the errors that were logged during the last blanking frame. Some of the errors occurr only when a command is issued and fails to execute properly (e.g., `LEVEL_ORANGE`), while others occur every time a check is performed, which may be every blanking period (e.g., `ID_NOTPRESENT`). The error log is a ring queue of length 64, therefore the 65th error will overwrite the first error record. It is the responsibility of the higher level software in the antenna computer to keep up with the logging of errors.

8. To Be (or Not to Be) Done

Two of the API commands are not implemented: the CANbus program download command (message ID 0x3FD), for which the downloader/uploader code was not ready a the time of this release, and the query attenuation vs. frequency command (message ID 0x104), which I do not plan to implement. The reply to the latter command was going to be the module gain vs. frequency as measured in the lab for the current attenuator settings. The practical problem is that the gain curves (e.g., Fig. 4) contain many features that need to be sampled at high frequency resolution for spectral line observations (Fig. 5), and given that there are 128 possible attenuator settings the amount of information is such that it cannot be easily contained in the module (we are talking about 1.6 MB approximately). This information should live further upstream, possibly close to the correlator where it will be used for passband correction. The passbands were measured in the lab at Berkeley before sending out the modules.

Table 1: SOFT Error Codes, Conditions, and Actions

Error code	Value	Condition and action
BAND_ORANGE	1	A set band command was received with a band parameter outside the 1..4 range. No action was taken. The requested band is reported.
ATTEN_UFLOW	2	An attenuation command was received requesting less than 0.0 dB. The requested value was set to 0.0 and the set attenuation process continued. The requested attenuation is reported
ATTEN_OFLOW	3	An attenuation command was received requesting $>$ 63.0 dB for the PAM, or more than 31.5 dB for any individual attenuator. Attenuation was set to maximum allowed and process continued. The requested attenuation is reported.
LEVEL_ORANGE	4	Set level command failed to complete in the allowed number of iterations, as measured output power is not within 0.3 dB of request. No action is taken. Measured power is reported.
FASTITEM_ORANGE	5	A start fast sampling command was received requesting a fast sampling item outside the 0..3 range. No action is taken. The requested item is reported.
UPLOAD_NOACTIVE	7	An orphan RF power detector calibration packet was received, without a start upload command issued first. No action is taken. The reported parameter is meaningless.
UNKNOWN_CODE	8	A packet with an unknown ID (one not in the API) was received. No action is taken. The packet ID is reported.
CALTABLE_LONG	9	The start upload command specified a calibration table length that is outside the range 2..16. No action is taken. The requested length is reported.
CALTABLE_NOTFOUND	10	No valid power sensor calibration found at boot time in EEPROM. The default calibration of 1000 mW/V will be used (i.e., the RF power detector readout will be reported in mV. The reported parameter is meaningless.

Table 2: HARD Error Codes, Conditions, Actions, and Likely Causes

Error code	Value	Condition and action
BAND_STUCK	129	A set band command failed with the sensed IF switch position different from the commanded position. The switch status word will set to SWITCH_STUCK until a successful set band command is issued. The reported parameter is the 4-bit nibble read in as the switch position (1's for open relays, 0's for closed relays). The switch is unplugged, or the hardware is malfunctioning.
ID_WRONGCLASS	130	The laser ID 1–wire ID is connected but appears to have the wrong family signature. No action is taken. The reported parameter is meaningless. The DS2401 1–wire ID IC is malfunctioning or its readout is noisy.
ID_WRONGCRC	131	The CRC checksum of the laser ID failed. No action is taken. The reported parameter is meaningless. The DS2401 1–wire ID IC is malfunctioning or its readout is noisy.
ID_NOTPRESENT	132	The DS2401 1–wire ID is not connected (no presence pulse detected). No action is taken. The reported parameter is meaningless. The OTX is unplugged, the wiring is flaky, or the DS2401 IC is malfunctioning.
EEPROM_FAILURE	133	In an assign module ID command, the number read from the EEPROM after the burn is not the assigned ID. No action is taken. The ID read from EEPROM is reported. The ATMEL EEPROM onboard the Phycore is malfunctioning.
UPLOAD_BURN	134	At the end of a calibration table upload, the table read from the EEPROM failed the check. The calibration table is used, but it will be lost after a reset or power cycle. The parameter is the number of differences found. The ATMEL EEPROM onboard the Phycore is malfunctioning.

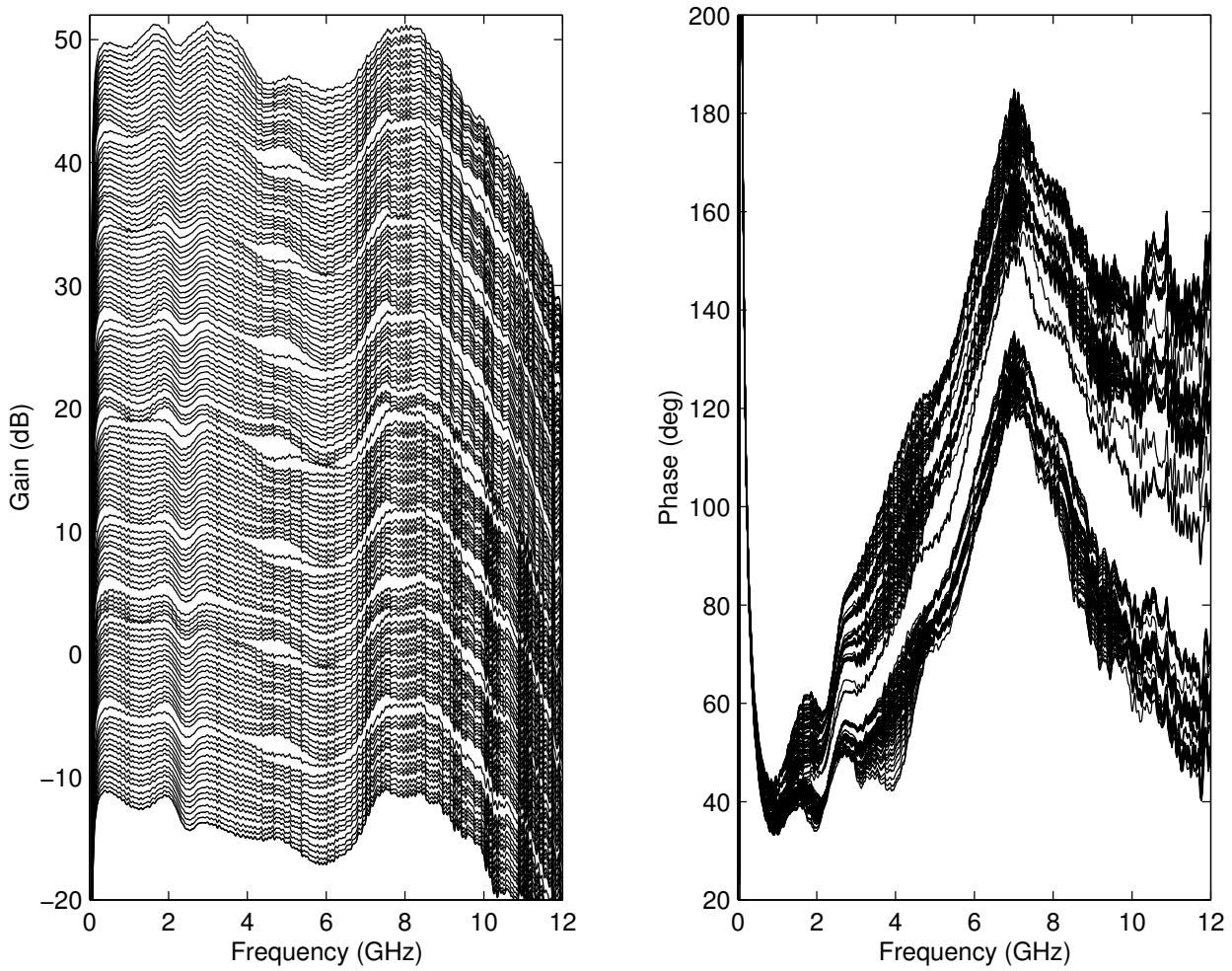

Fig. 4.— Gain and phase passband curves measure for IF module #12, for all 128 attenuator settings.

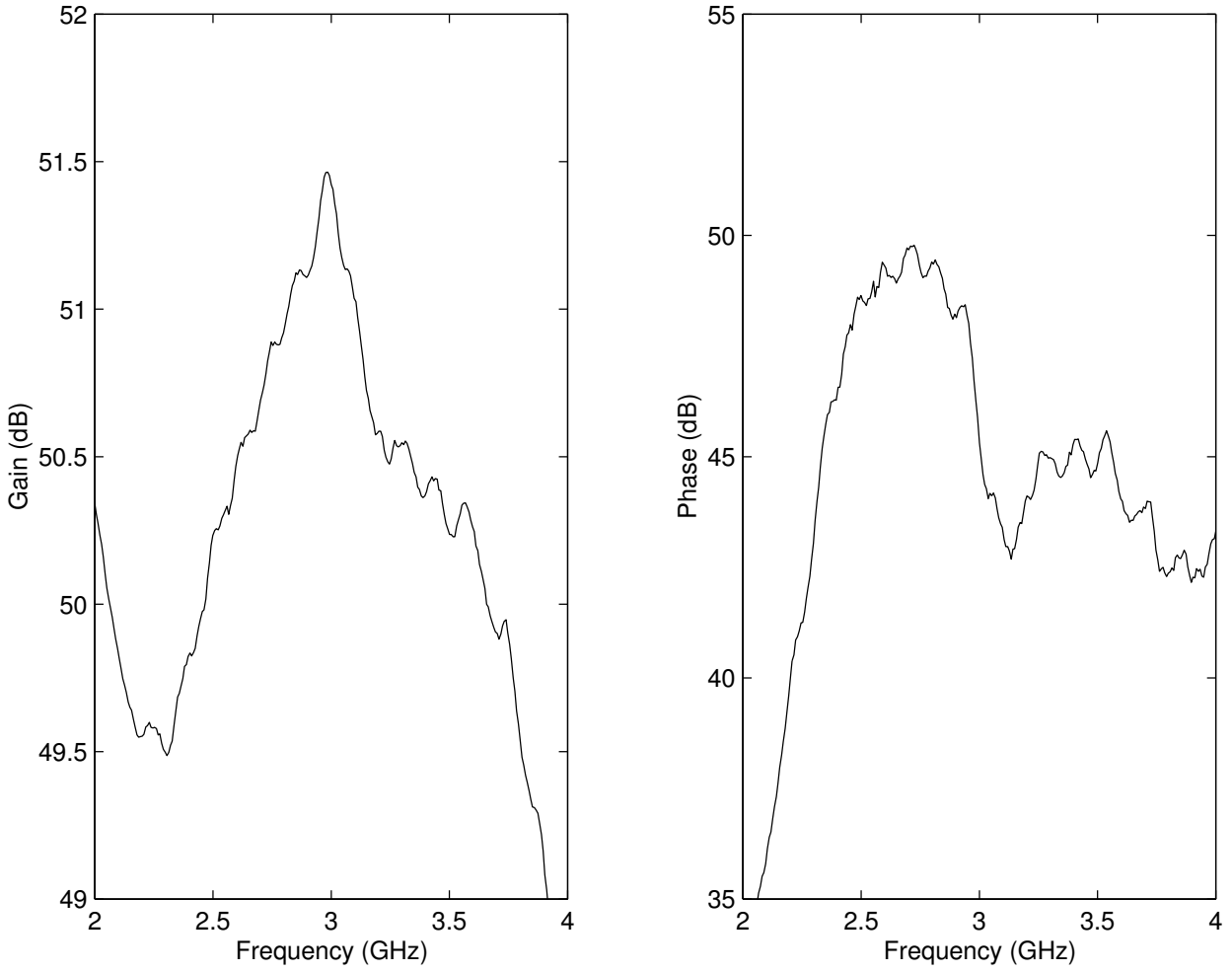

Fig. 5.— Gain and phase passband details for IF module #12, measured at the lowest attenuation between 2 and 4 GHz.