

**Digital Low-Level Radio Frequency Control and
Microphonics Mitigation of Superconducting Cavities**

By

Nathanael Robert Usher

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

2007

ABSTRACT

Digital Low-Level Radio Frequency Control and Microphonics Mitigation of Superconducting Cavities

By

Nathanael Robert Usher

Superconducting Radio Frequency (SRF) cavities designed for proposed linear reaccelerators at the National Superconducting Cyclotron Laboratory (NSCL) have an unloaded quality factor (Q) on the order of 10^9 . The high Q results in a very narrow bandwidth, so small perturbations in the cavity's resonant frequency result in large changes in the magnitude and phase of its electric field for a fixed frequency driving signal. As a result, it is necessary to use a controller that can compensate for perturbations as they occur.

This project develops a new digital low-level radio frequency (LLRF) controller for this application, including DSP-based hardware, software algorithms, and a graphical user interface. The digital LLRF controller provides two types of control over cavity perturbations. The LLRF implements a PID control on the cavity driving signal to compensate for nonlinearities and small random disturbances. Significant cavity detuning caused by narrow-band vibrations, referred to as "microphonics," are reduced by the LLRF controller using a piezoelectric tuner controlled by an adaptive feedforward cancellation algorithm.

The completed prototype is a small, standalone, economically viable, ethernet connected controller. Through a MATLAB interface, it will allow advanced algorithm development by control engineers who are not necessarily familiar with the low-level workings of the controller.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Xiaobo Tan and Dr. John Vincent of the National Superconducting Cyclotron Laboratory (NSCL) for giving me the opportunity to work on this project. I would also like to thank everyone in the electronics department at the NSCL for their help. Without help from Mark Davis and John Priller, I would not have been able to develop the network interface and graphical front-end. Adam Molzahn laid the groundwork for this project with his digital phase meter and introduced me to the fundamentals of how the system works. Without Adam's work to build upon and the help and guidance of the NSCL staff, I would not have been able to complete this project.

TABLE OF CONTENTS

ABSTRACT	ii
ACKNOWLEDGMENTS	iii
LIST OF TABLES	vi
LIST OF FIGURES	vii
1 Introduction	1
2 Signal Digitization and Interlocks	9
2.1 Collecting Input Samples	9
2.2 CORDIC Algorithm	12
2.3 DSP Interface	15
2.4 RF Output Creation	18
2.5 Interlocks	19
3 Tasks Performed by the Digital Signal Processor	21
3.1 DSP Peripherals	21
3.2 Control Program Overview	22
3.3 Data Received From Host Processor	23
3.4 Control Thread	24
3.5 Status Thread	25
3.6 Software Interrupts	26
4 Control	28
4.1 RF Amplitude and Phase Control	28
4.2 Cavity Tuning	29
5 The ZWorld RabbitCore Host Processor and Graphical User Interface	32
5.1 Module Initialization	32
5.2 User Interface	33
5.3 Main Screen	34
5.4 PID Screen	36
5.5 AFC Screens	37
5.6 Startup Screen	40
5.7 Advanced Screen	40
6 Testing and Performance	42
6.1 Cavity Simulation Circuit	42

7	Conclusion and Future Work	50
7.1	Conclusion	50
7.2	Required Changes	51
7.3	Future Hardware Upgrades	52
7.4	Future Software Upgrades	53
A	FPGA Implementation of a CORDIC Algorithm	55
B	DSP Daughter Board Schematics	59
	BIBLIOGRAPHY	71

LIST OF TABLES

- 2.1 CORDIC algorithm example. 15
- 2.2 Description of FPGA memory locations. 16
- 3.1 Responses from the DSP are received by the host two words after the command was transmitted. 24

LIST OF FIGURES

1.1	Overview of the LLRF controller as it will be connected to an SRF cavity. The LLRF controller consists of everything inside of the dashed lines. . . .	4
1.2	LLRF controller motherboard. DSP daughter board is attached on the lower right corner. Host processor board attaches to back side. Front side contains the reference phase-locked loop and all input signal conditioning components. Back side contains the high-speed ADCs and DAC, FPGA, and output signal conditioning components.	5
1.3	DSP daughter board. DSP is underneath the large black heat sink.	6
1.4	ZWorld RabbitCore daughter board.	6
2.1	Sampling a 50 MHz signal at 40 MSPS produces a 10 MHz alias whose phase is measured relative to the phase of the 10 MHz reference signal. . .	11
2.2	DAC output spectrum when producing output at 40 MSPS (left) and IF after 50 MHz band-pass filter (right).	19
2.3	DAC output spectrum when producing output at 80 MSPS (left) and IF after 50 MHz band-pass filter (right).	20
4.1	Block diagram of PID control. The cavity behaves linearly when operating near its resonant frequency, so it can be approximated by the transfer function, $G(s)$	29
4.2	Adaptive feedforward cancellation control for the cancellation of a single sinusoidal disturbance [22].	30
5.1	Main controller screen.	35
5.2	PID tuning screen.	37
5.3	AFC screen.	38
5.4	FFT screen shows FFT of cavity output in dBm, cavity magnitude response, and cavity phase response in degrees. All plots show 0 to 1000 Hz linear scale.	39
5.5	Startup screen.	40
5.6	Advanced screen.	41
6.1	Block diagram of circuit used for cavity simulation. The two transfer functions are replaceable RC networks.	43

6.2	Each of the transfer function blocks is made up of a replaceable RC network.	44
6.3	Ideal magnitude response (top) and magnitude response calculated by LLRF controller (bottom).	46
6.4	Ideal phase response (top) and phase response calculated by LLRF controller (bottom).	47
6.5	FFT of RF input with 60 Hz sinusoidal disturbance. Because the magnitude of the disturbance is large, its third harmonic is also present.	49
B.1	System Overview.	60
B.2	Voltage regulation and clock generation.	61
B.3	DSP power connections.	62
B.4	DSP bypass capacitors, UTOPIA port, HPI port and reserved pins.	63
B.5	DSP serial connections.	64
B.6	DSP JTAG emulation port and general purpose port.	65
B.7	DSP external memory interface A.	66
B.8	External SDRAM.	67
B.9	DSP external memory interface B.	68
B.10	External non-volatile memory and FPGA connection.	69

CHAPTER 1

Introduction

The desire to do further research into particle physics has driven a demand for new high-energy accelerators. The reduced surface resistances of superconducting cavities make them capable of supporting high fields with low power losses. The quality factor (Q) of a cavity of given geometry is an indicator of how high a field it can achieve. While cavities with high quality factors can achieve higher internal fields, this comes at the expense of narrower bandwidth. The very narrow bandwidth associated with high Q cavities makes maintaining amplitude and phase control in the presence of perturbations increasingly difficult [1].

Feedback control is required to maintain a stable cavity field in the presence of a disturbance. One can apply a PID control to maintain the amplitude and phase of the cavity field at a given set point. As perturbations shift the cavity's resonant frequency away from the RF driving frequency, the low-level radio frequency (LLRF) controller measures the error in the cavity field and compensates. Although this method works well for small perturbations, it will not work for large perturbations, since the required input power to maintain the cavity field will be too high. Therefore, for high- Q superconducting cavities, it is necessary to have an active tuning mechanism to cancel the disturbances to keep the driving frequency within the cavity's bandwidth.

A properly designed continuous wave system will only have limited narrowband sinusoidal disturbances due to vibrations (microphonics). Simrock et al. were the first to demonstrate using a piezoelectric actuator to cancel these disturbances [2]. Due to the complex transfer functions of multi-cell superconducting cavities, a feedback control would be exceedingly difficult to implement. However, Kandil et al. demonstrated an adaptive feed-forward cancellation (AFC) algorithm capable of canceling narrow-band vibrations in a multi-cell cavity using a dSpace development board connected to a PC [3]. Although the AFC algorithm was shown to work, the implementation required a PC and several other pieces of equipment to control a single cavity. Therefore, it was not practical for use in a running linear accelerator, which will have many cavities that need to be controlled. The control needs to be implemented on a low-level radio frequency (LLRF) controller to be useful, which is the subject of this work. The LLRF controller will use the AFC algorithm to attenuate cavity detuning from significant narrow-band disturbances, and use a PID control to continuously adjust the amplitude and phase of the RF driving signal to compensate for other disturbances.

Digital LLRF controllers are currently being designed at several accelerator labs to replace analog controls. Implementing controls in the digital domain reduces the size and cost of the hardware and allows the implementation of controls that are not feasible on analog systems [9]. Most of the new LLRF controllers being developed for applications that do not require a lot of computing power rely on a field programmable gate array (FPGA) to perform the RF measurements and control algorithms. The controllers being developed for the Spallation Neutron Source (SNS) accumulator ring [5] and International Linear Collider (ILC) [6] are both based on FPGAs. For applications that require more processing

power, an FPGA with an embedded processor is commonly used [7]. Although some of the other LLRF controller designs provide a lot of the functionality required for the linear reaccelerator upgrade to the NSCL and could possibly be adapted if necessary, they do not meet all of the requirements for the reaccelerator upgrade. The problem with either of these designs is that upgrading the software for the system requires extensive knowledge of the LLRF controller and experience with FPGA programming. One of the requirements for the LLRF controller for the NSCL is that it must be possible for control engineers to update the software controls using tools they are familiar with, without needing to know the details about how the hardware is set up. Other requirements are that the LLRF controller must be completely standalone, requiring only a network connection to communicate with the EPICS control system [8], it needs to provide tuner outputs for microphonics mitigation, and it needs to have a user-friendly interface for setup and monitoring.

Figure 1.1 shows an overview of the LLRF controller developed here, which was inspired by the LLRF controller designed by Lawrence Berkeley National Laboratory [9]. The LLRF controller consists of a motherboard (Figure 1.2), a DSP daughter board (Figure 1.3), and a control host board (Figure 1.4). The input signal conditioning portion of the motherboard is unchanged from the previous iteration of the project: the digital phase meter developed by Adam Molzahn [10]. However, there have been major changes to the digital sections. The biggest change is that a high-performance DSP has been added to the controller. All of the digital processing is now done by the DSP, so all data is sent from the FPGA directly to the DSP instead of the host processor. The DSP runs the PID based amplitude and phase control algorithms and feedforward based microphonics mitigation algorithms and sends data to the control host processor as necessary. This frees up enough

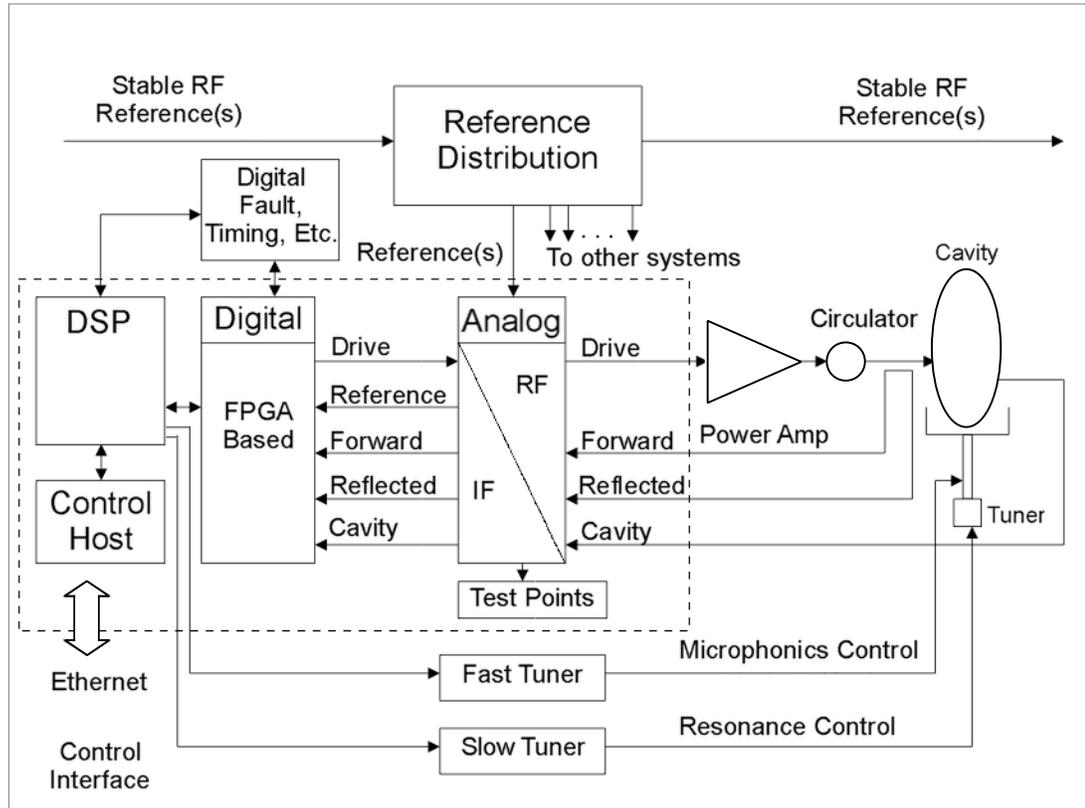


Figure 1.1. Overview of the LLRF controller as it will be connected to an SRF cavity. The LLRF controller consists of everything inside of the dashed lines.

FPGA logic to allow it to perform a 16-bit coordinate rotation digital computer (CORDIC) algorithm [11] on all incoming channels to obtain their phase and magnitude information. The parallel nature of the FPGA makes it the ideal place for doing the phase and amplitude conversions, since it can do all input channels simultaneously without slowing down the rest of the system. By converting all input channels simultaneously, we guarantee that a read of all input channels returns amplitude and phase information taken at the same sample time.

There are five RF inputs on the front of the controller module. One is a reference signal (10 MHz), used to control the sample clock for the high-speed analog-to-digital converters

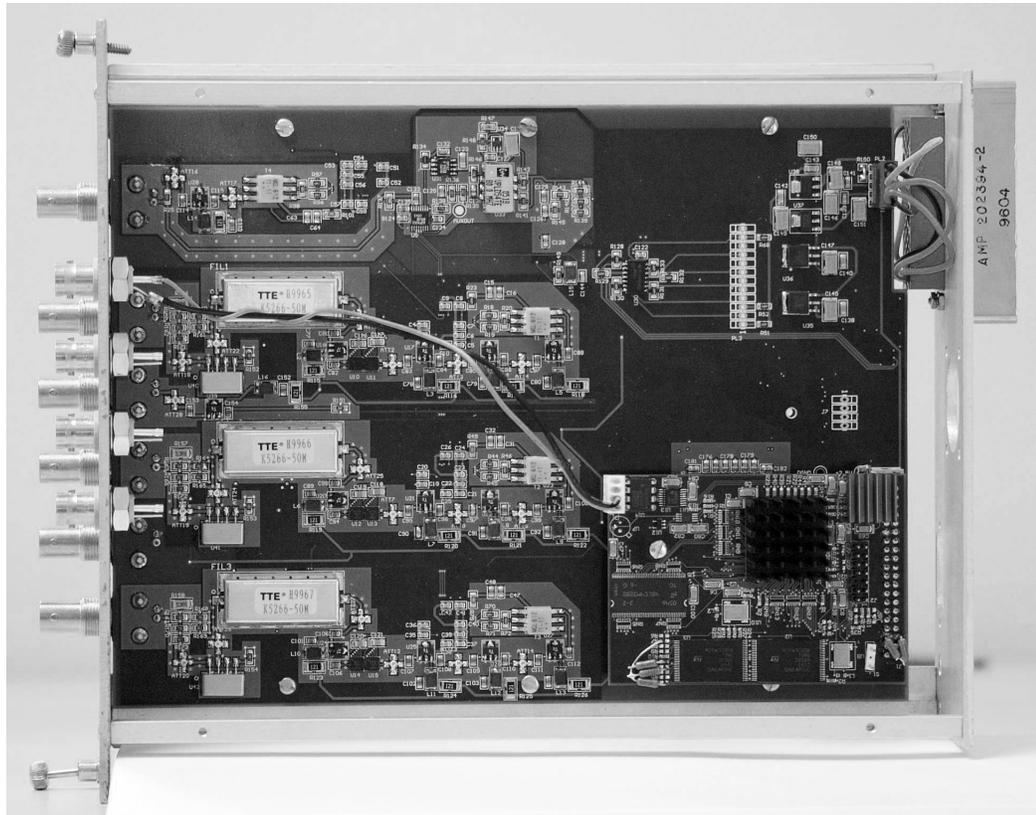


Figure 1.2. LLRF controller motherboard. DSP daughter board is attached on the lower right corner. Host processor board attaches to back side. Front side contains the reference phase-locked loop and all input signal conditioning components. Back side contains the high-speed ADCs and DAC, FPGA, and output signal conditioning components.

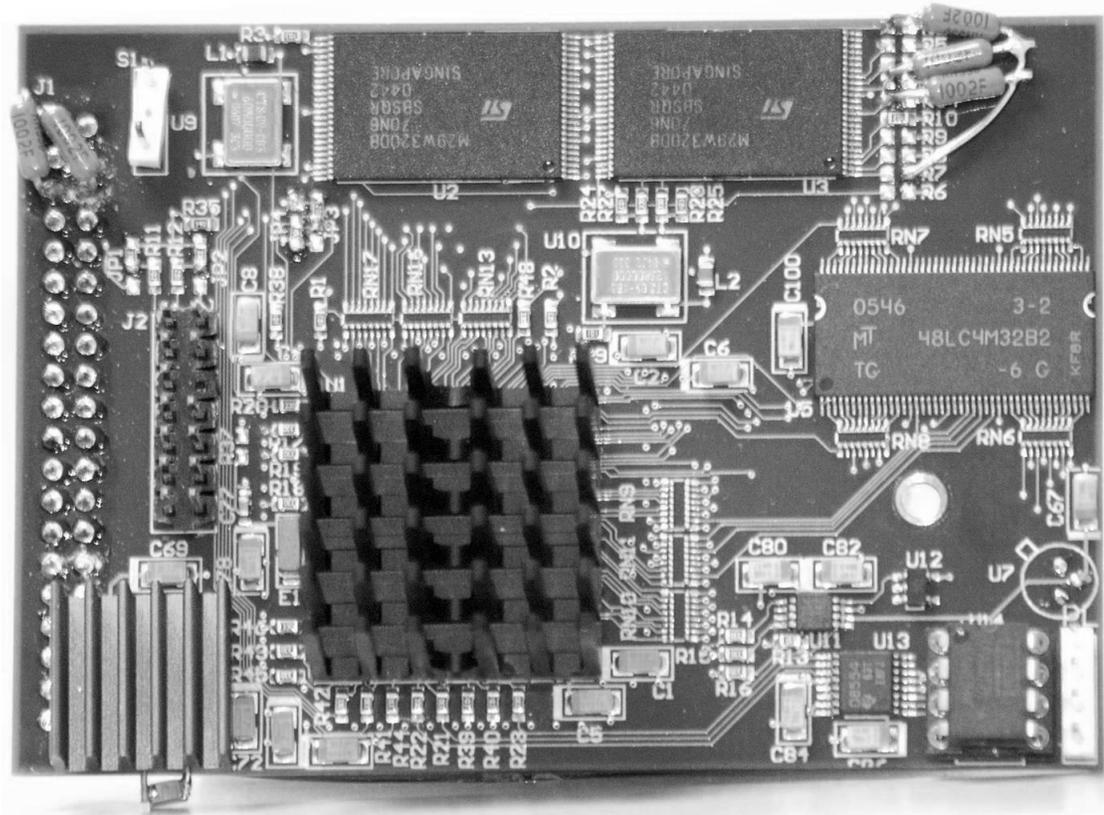


Figure 1.3. DSP daughter board. DSP is underneath the large black heat sink.

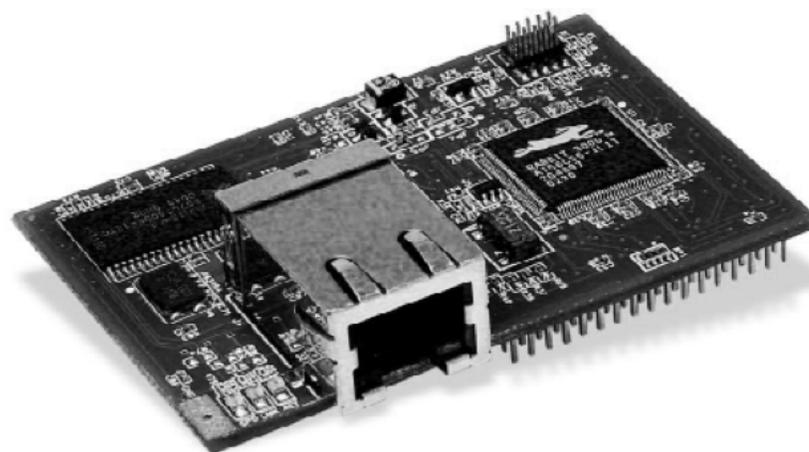


Figure 1.4. ZWorld RabbitCore daughter board.

(ADCs) used to digitize the inputs and to synchronize multiple modules when they are implemented together in an accelerator. Three of the RF inputs are the input signals to be measured (forward power, reflected power, and cavity output). Depending on mixer and filter parts selection when assembling the modules, the inputs can be between 2 MHz and 1.5 GHz, with power levels from -7 dBm to +33 dBm. However, there are some frequencies within that range that will not work well due to interactions of the harmonics that may be present within the intermediate frequency. The final RF input is the local oscillator (LO) input, which must be at a frequency 50 MHz higher or lower than the RF input frequency and phase-locked to the reference signal. The LO signal is mixed with each RF input to convert them to the 50 MHz intermediate frequency (IF) for sampling. The fixed 50 MHz intermediate frequency is used, because it is easier to digitize a signal with a fixed, known frequency. An ideal mixer produces frequencies which are the sum and difference of its two input signals as shown in Equation (1.3). Since the LO signal is chosen 50 MHz higher or lower than the RF frequency, one of the components of the mixer output is at 50 MHz, and the other is filtered out using a 50 MHz band pass filter. Also, since the LO is phase-locked to the reference signal, θ_2 is fixed and can be calibrated out. Therefore, phase changes in the RF input directly affect the phase of the IF signal.

$$RF = A_1 \sin(\omega_1 t + \theta_1) \quad (1.1)$$

$$LO = A_2 \sin(\omega_2 t + \theta_2) \quad (1.2)$$

$$RF * LO = \frac{A_1 A_2}{2} \cos((\omega_1 + \omega_2)t + \theta_1 + \theta_2) - \frac{A_1 A_2}{2} \cos((\omega_1 - \omega_2)t + \theta_1 - \theta_2) \quad (1.3)$$

The remaining front panel connectors are the RF output, a tuner output, RF On/Off input, reset input, and a fault output. The RF output is produced by creating an output at the IF, then mixing it with the LO to produce a signal at the desired frequency. The fast tuner output is an analog signal that is designed to cancel microphonics up to 1 kHz. A slow tuner output is provided that can be used to compensate for slow drifts in the resonant frequency of the cavity. The rest of the signals are digital control signals.

CHAPTER 2

Signal Digitization and Interlocks

The field programmable gate array's (FPGA) primary task is to collect the data sampled by the high-speed ADCs and buffer it until it is requested by the DSP. The configurability and parallel nature of the FPGA makes it ideal for collecting and buffering the sampled RF signals. The FPGA's other tasks include setting the digital attenuators on the input channels and taking care of some of the interlock signals. The FPGA used in this project is a Xilinx Spartan-II XC2S150. Although it is adequate for this project, future versions will probably move to a newer, faster FPGA with more available logic.

2.1 Collecting Input Samples

To be able to calculate the amplitude and phase of the signal, we need a vector-based measurement of the signal with respect to a reference. If the samples are taken at 90° intervals, then the vector realizations can be equated to the real and imaginary components of a vector in the complex plane. If the first sample is taken as the real component of the signal, then the second sample is 90° lagging and is equated to the imaginary component of the signal. In the absence of a DC sampling error, the third sample is the negative of the first sample and the fourth sample is the negative of the second sample. The first and

third and second and fourth samples can be averaged to remove any DC sampling error. Once the real (I) and imaginary (Q) components are known, the phase and magnitude can be computed using:

$$Phase = \tan^{-1} \left(\frac{Q}{I} \right) \quad (2.1)$$

$$Magnitude = \sqrt{I^2 + Q^2} \quad (2.2)$$

The phase information gathered with this method can be used to tell the relative phase between inputs, but is not meaningful when using multiple LLRF controllers in a system, since the first sample is taken at an arbitrary time. Therefore, one of the RF inputs is used as a reference signal. The phase of the reference signal is subtracted from the phase of each input to produce phases relative to the reference signal, which allows multiple LLRF controllers to be synchronized with each other.

In a variable frequency system, it can be difficult to take samples every 90° since the time delay between samples depends on the signal frequency. However, since all of our inputs have been mixed to 50 MHz, we have a known frequency, and just need to choose an appropriate sample rate [10]. To take samples every 90° of the input signal, we need four samples per period, and therefore, a sample rate of 200 MSPS. ADCs that can sample at 200 MSPS are expensive, so it is not cost-effective to use them, and it was necessary to find another way to get the I and Q values. By removing the condition that all four samples in a measurement must come from the same period of the input signal, we can use a lower speed, less expensive ADC. Therefore, a sample rate of 40 MSPS was chosen. Sampling at 40 MSPS produces a 10 MHz alias with the same amplitude and phase as the 50 MHz IF signal (Figure 2.1) and gives us samples taken every 90° as desired.

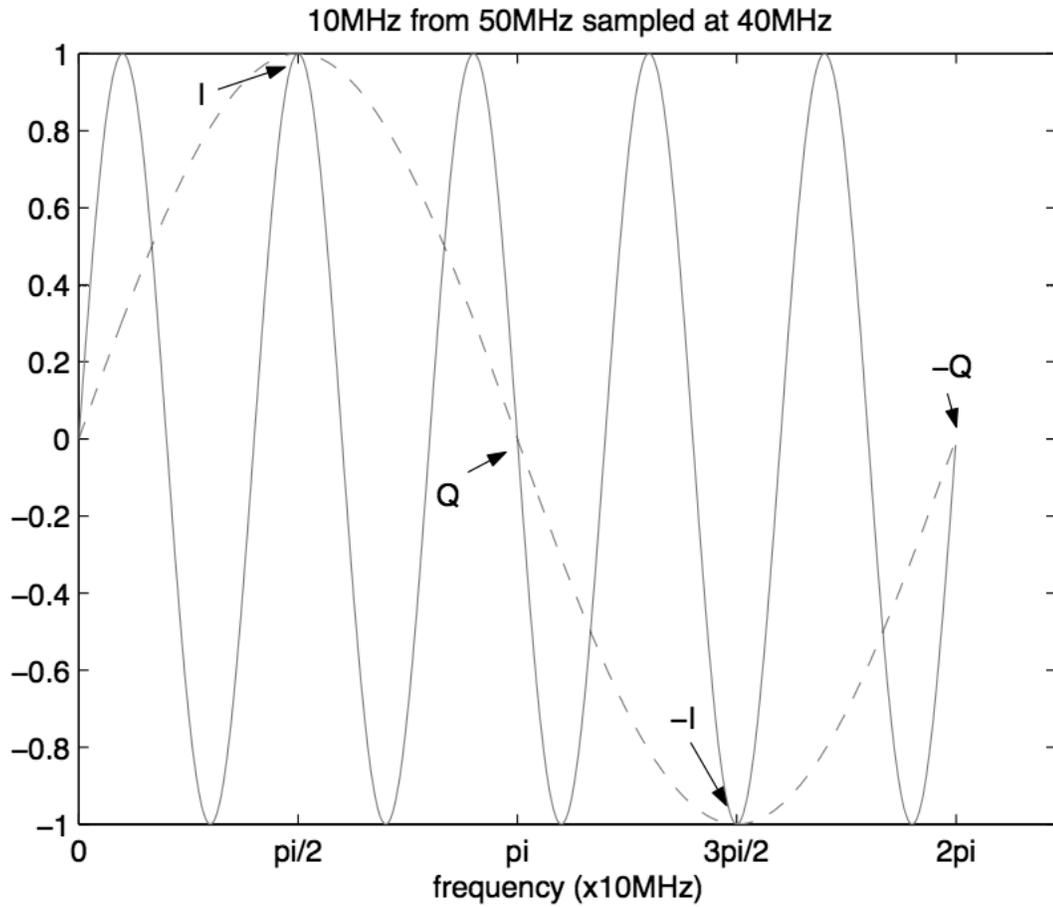


Figure 2.1. Sampling a 50 MHz signal at 40 MSPS produces a 10 MHz alias whose phase is measured relative to the phase of the 10 MHz reference signal.

Each ADC samples its input on the rising edge of the 40 MHz sample clock. The FPGA reads the 14-bit data output of each ADC on the falling edge of the clock, when the data becomes valid. The FPGA samples all inputs simultaneously, so the relative phases of each channel are preserved, even if one or more of the input channels are drifting in phase. As seen in Figure 2.1, by sampling the intermediate frequency (50 MHz) at 40 MHz, the system produces samples that are 90° apart, and if the phase of the input is constant, the input sequence will repeat every four samples. The FPGA passes every input through a 10 MHz first-order digital band-pass filter to try to remove noise that might have been picked up by the ADC during sampling [12]. Next, each set of samples is saved as either I , Q , $-I$, or $-Q$, depending on the value of a 2-bit counter. The FPGA then averages the I and $-I$ and Q and $-Q$ pairs and buffers them so they can be accessed by the DSP. The averaging removes any DC offset from the samples, to give the real and imaginary components of the vector.

2.2 CORDIC Algorithm

Since most vector calculations done by the DSP need to be done in polar coordinates, it is necessary to implement an efficient algorithm for conversion between the rectangular and polar coordinates. In the previous iteration of this project [10], these conversions were done by the host processor using equations (2.1) and (2.2). However, doing the conversions in the host processor is much too slow to be reused here. One option to speed up the conversions was to implement the same conversion in the DSP. However, implementing the conversion in the DSP uses a surprisingly large amount of CPU time and would

greatly reduce the amount of computing time available for the control algorithms. The best option is to implement the conversions in the FPGA using the Coordinate Rotation Digital Computer (CORDIC) algorithm. The CORDIC algorithm was designed in the early days of digital computers as a way to perform conversions between polar and rectangular coordinates quickly using a small number of gates. It uses a series of multiplications by known complex numbers to shift the input vector to 0° , while keeping a running total of the amount of phase shift it has added. After shifting the input vector to 0° , we know that the output vector's real component is the input vector's magnitude, and the input vector's phase is the negative of the total phase shift. By selecting the phase shifts carefully, each step of the algorithm can be implemented using only right-shifts and addition or subtraction [13].

Another advantage of the CORDIC algorithm is that it does not have to produce its output in any particular type of unit, so we are not restricted to using radians or degrees to represent the phase. To speed DSP calculations, the system represents phase as a raw 16-bit value, with each bit representing $\frac{360}{65536}$ degrees. This is advantageous because the DSP no longer has to check whether its results are between 0° and 360° , since any result that would normally fall outside this range simply overflows and wraps around to the desired value.

In each stage of the CORDIC algorithm, we check the sign of the Q value to decide whether to add or subtract phase. The first stage is a 90° shift. If Q is positive, the I and Q values are swapped and the new Q value is negated to produce a -90° shift, and 90° is added to the cumulative phase. If Q is negative, the I and Q values are swapped, the new I value is negated to produce a $+90^\circ$ shift, and 90° is subtracted from the cumulative phase.

$$(I + Qj)(0 + j) = -Q + Ij \quad (2.3)$$

$$(I + Qj)(0 - j) = Q - Ij \quad (2.4)$$

In subsequent stages, the shifts are performed by multiplying the intermediate vector by a vector of the form $1 + 2^{-n}j$ or $1 - 2^{-n}j$, depending on the sign of Q .

$$(I + Qj)(1 + 2^{-n}j) = (I - 2^{-n}Q) + (Q + 2^{-n}I)j \quad (2.5)$$

$$(I + Qj)(1 - 2^{-n}j) = (I + 2^{-n}Q) + (Q - 2^{-n}I)j \quad (2.6)$$

Since the FPGA stores numbers in binary format, $2^{-n}I$ is equal to I shifted to the right by n bits, and $2^{-n}Q$ is equal to Q shifted to the right by n bits. The second stage of the algorithm uses $n = 0$, and n is increased by 1 in each subsequent stage. The algorithm gains approximately 1 bit of resolution in each stage it performs, so it runs 16 stages as set up in the FPGA. The FPGA performs each stage of the routine on the rising edge of the ADC sample clock, and it does 16 stages per conversion, so the FPGA is able to convert every fourth set of I , Q , $-I$, and $-Q$ values for each channel. This produces phase and magnitude pairs 2,500,000 times per second, which is much faster than the DSP will request them.

One issue that needs to be taken into account with this algorithm is that after the first stage, each multiplication changes the magnitude of the vector being converted, not just the phase. However, the total magnitude scaling is a constant dependent only on the number of iterations of the algorithm. Therefore, the magnitude scaling inverse can be calculated ahead of time and can be used to cancel the scaling effect. Table 2.1 shows an example conversion including the magnitude scaling constant. The cumulative phase converges toward the actual phase of 53.8° , and I (divided by the magnitude constant) converges to

Table 2.1. CORDIC algorithm example.

Stage	I	Q	Cum. Phase	Mag. Constant	Next Phase Shift
0	150	205	0°	1	-90°
1	205	-150	90°	1	45°
2	355	55	45°	1.41421	-26.565°
3	382.5	-122.5	71.565°	1.58114	14.036°
4	413.125	-26.875	57.529°	1.62980	7.125°
5	416.484	24.766	50.404°	1.64248	-3.576°
6	418.032	-1.265	53.98°	1.64569	1.790°

the actual magnitude of 254.02.

In order to produce the high-speed DAC output from the desired phase and magnitude information, it is necessary to perform the CORDIC algorithm in the reverse direction. The algorithm is almost identical, except that the FPGA starts with the desired magnitude in I , the desired phase in the phase accumulator, and $Q = 0$. Then, it performs shifts based on the sign of the phase value until the phase is shifted to 0. Again, there is a scaling factor in the conversion, so the DSP must multiply the desired magnitude by the scaling factor's inverse before sending the phase and magnitude pair to the FPGA.

2.3 DSP Interface

The fastest way to transfer data between the DSP and a peripheral is through one of its two external memory interfaces (EMIF). To connect the FPGA to the DSP, I set up the FPGA to appear as a 16-bit asynchronous RAM, then defined memory locations that the DSP can read or write to send or receive data [14]. Because of the small number of available pins on the FPGA, I could only set up 32 memory locations, but this is sufficient for the current system. The memory locations are shown in Table 2.2.

Table 2.2. Description of FPGA memory locations.

Add.	Description	Read/Write	Add.	Description	Read/Write
0	Ch1 - Ref. Phase	Read-only	16	Ch1 I	Read-only
1	Ch1 Amplitude	Read-only	17	Ch1 Q	Read-only
2	Ch2 - Ref. Phase	Read-only	18	Ch2 I	Read-only
3	Ch2 Amplitude	Read-only	19	Ch2 Q	Read-only
4	Ch3 - Ref. Phase	Read-only	20	Ch3 I	Read-only
5	Ch3 Amplitude	Read-only	21	Ch3 Q	Read-only
6	Always Outputs 0	Read-only	22	Ref I	Read-only
7	Ref. Amplitude	Read-only	23	Ref Q	Read-only
8	Output Phase	R/W	24	Output I	Read-only
9	Output Amplitude	R/W	25	Output Q	Read-only
10	Error Register	Read-only	26	I Input	R/W
11	Digital Attenuator	R/W	27	Q Input	R/W
12	Ref. Phase	Read-only	28	I Input	R/W
13	Unused	R/W	29	Q Input	R/W
14	Output (shifted 225°) I	Read-only	30	Phase Output	Read-only
15	Output (shifted 225°) Q	Read-only	31	Magnitude Output	Read-only

The phase and magnitude information discussed in the previous section is available to the DSP in memory locations 0 to 7. When the DSP performs a sequential read of the input channels, the FPGA temporarily stops updating the data. This guarantees that the data from each channel was obtained at the same time. The FPGA begins updating the data when the DSP reads or writes any other location. The phase of each channel is given to the DSP relative to the reference phase. As a result, the reference phase in register 6 is always 0. In the event that the absolute phase of the reference signal is desired, it is available in register 12. The I and Q values for each input channel are available in memory locations 16 to 23.

The DSP sets the RF output by writing a desired phase and amplitude to registers 8 and 9. The FPGA automatically runs the new settings through its CORDIC routine and produces corresponding I and Q values in locations 24 and 25. The I and Q values are sent to the high-speed DAC to produce the RF output. When producing the output at 80 MSPS,

the FPGA uses a total of 8 samples, and therefore needs two sets of I and Q pairs. The extra pair is produced by adding 225° to the desired output phase before passing it through the CORDIC routine. The extra I and Q values are available in locations 14 and 15.

Six of the memory locations (26 to 31) are for an additional CORDIC routine. When the DSP writes a set of I , Q , $-I$, and $-Q$ values to registers 26 through 29, the FPGA runs the CORDIC algorithm on the set of input data and produces a phase and magnitude pair in registers 30 and 31. These registers are used when the DSP is measuring the transfer function of the cavity, to find the value of the phase shift and magnitude at each measurement frequency.

The value of the digital attenuators is controlled by register 11. The digital attenuators have six control bits, so the uppermost ten bits of this register are not used and are read as 0. A value of 0 corresponds to the minimum attenuation, and a value of 63 corresponds to the maximum attenuation.

Memory location 10 shows the over-voltage status of each ADC and the status of each interlock. Each ADC has a pin that is set high if the input voltage to the ADC is too high to produce a valid reading. The over-voltage pins are connected to the 4 least significant bits of register 10. If the digital attenuators are set up correctly and there is no malfunction in the module, these bits will read 0. Bits 4 and 5 correspond to the reset signal and on/off signal, respectively. During normal operation, these bits will both read 0.

There is one unused register, in location 13. It can be read from and written to by the DSP, but currently does not perform any function in the FPGA.

2.4 RF Output Creation

The input sampling process can be reversed to produce the RF output. Since the DSP provides the output set points as phase and amplitude pairs, the FPGA first passes the set points through the CORDIC routine to produce the I and Q values. Since it takes 400 ns for the FPGA to complete a single CORDIC conversion, and the FPGA is set up to continuously perform conversions on the values stored in the phase and amplitude registers, it takes between 400 ns and 800 ns before a new setting affects the output. The $-I$ and $-Q$ values are created by negating the I and Q values. The final step necessary before sending the values to the output DAC is to convert them into the proper format. The FPGA represents numbers internally as 16-bit two's complement numbers (-32768 to 32767), while the DAC expects 14-bit straight binary numbers (0 to 16383). The FPGA converts each output sample by complementing the most significant bit to put each sample into 16-bit straight binary format and ignores the two least significant bits to shrink the number to the required 14 bits.

The FPGA passes the $I/Q/-I/-Q$ values to the output DAC at 40 MSPS to produce the RF output. The DAC produces a square wave, so the output is not made up of a single frequency component; the output contains a 10 MHz signal and its odd harmonics (30 MHz, 50 MHz, 70 MHz, etc.) as shown in Figure 2.2. Since we only want the 50 MHz signal to be present, the DAC output is passed through a 50 MHz filter, which attenuates the unwanted frequencies.

Increasing the sampling rate of the DAC should provide both a better signal-to-noise ratio and increase the power of the 50 MHz signal [15]. The output DAC supports sample

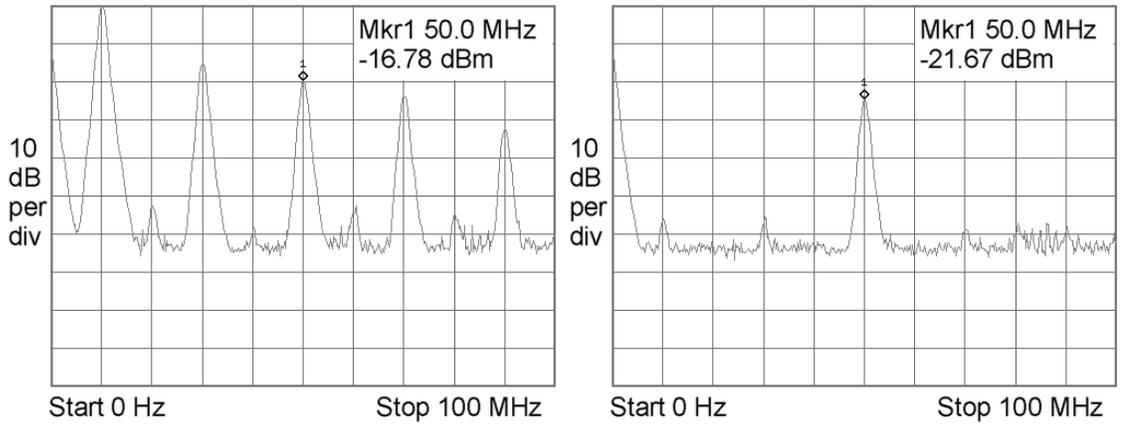


Figure 2.2. DAC output spectrum when producing output at 40 MSPS (left) and IF after 50 MHz band-pass filter (right).

rates up to 165 MSPS, but the current RF board design limits the DAC sample rate to 80 MSPS [16]. To create the RF output at 80 MSPS, we need to cycle through eight samples, taken at 0° , 225° , 90° , 315° , 180° , 45° , 270° , and 135° . Four of these samples (0° , 90° , 180° , and 270°) correspond to the I/Q – I/Q samples used when producing the output at 40 MSPS. To produce the other four samples, the FPGA adds 225° to the desired phase value and then sends the new phase and amplitude pair through the CORDIC routine. Producing the RF output at 80 MSPS yielded an approximately 5.3 dB increase in power at the intermediate frequency and approximately a 3.0 dB improvement in the signal-to-noise ratio as shown in Figure 2.3.

2.5 Interlocks

The module has three external connections for safety interlocks: RF enable, fault, and reset. All of the interlocks are TTL signals that are connected through the FPGA so that

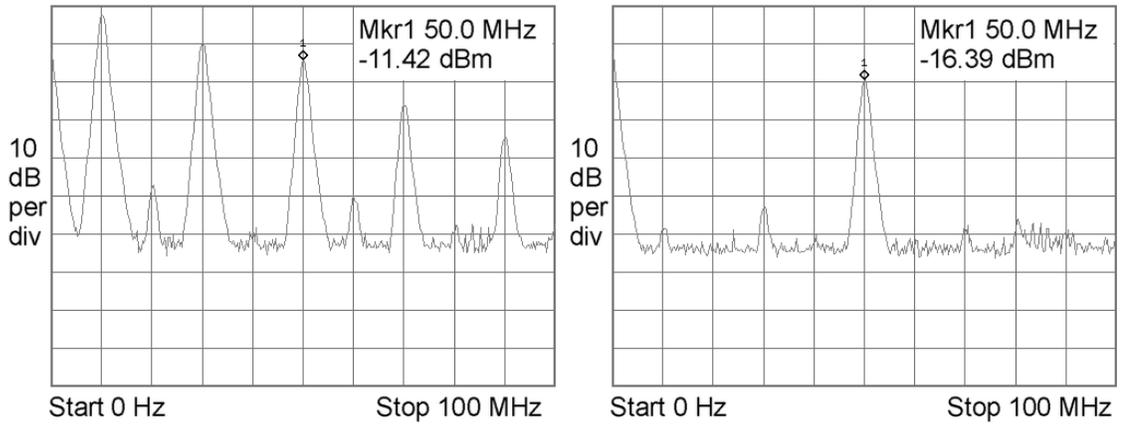


Figure 2.3. DAC output spectrum when producing output at 80 MSPS (left) and IF after 50 MHz band-pass filter (right).

their behavior can be reconfigured if necessary. The module's RF output must be disabled unless the RF enable interlock is low. This is accomplished by tying the RF enable signal to the power down pin on the high-speed DAC. The fault interlock is an output, which can be used to signal the rest of the RF control system. It is currently unused, since the module is not programmed to detect faults. If the module were to detect a fault, it would disable the RF output, set the fault signal, and wait for a reset signal. When a rising edge is detected on the reset input, the module will reset the fault signal, and enable the RF output (if the RF enable interlock is low).

CHAPTER 3

Tasks Performed by the Digital Signal Processor

The control processing is performed by a Texas Instruments TMS320C6414T fixed-point DSP. It was chosen because of its high speed and available software tools. In the current iteration of the module, the DSP is running at 360 MHz, but if more performance is required in the future, the DSP clock speed can be increased to 1 GHz. Although all of the current code is written in C and assembly, code can be generated from MATLAB using the Real-Time Workshop Embedded Coder. This will allow control engineers using the module to change control algorithms using tools they are familiar with.

3.1 DSP Peripherals

The DSP has two 4 MB flash EEPROM modules connected via the same memory bus the FPGA uses. The first is an 8-bit wide memory divided into four 1 MB pages and is used to store the programs. After reset, the DSP checks the status of two of the host processor output pins to determine which program it should boot. Then, it sets the page of the program ROM accordingly and loads the program into internal RAM. The second

module is 16 bits wide and is divided into two 2 MB pages. It is used as storage for large lookup tables and any measurements that need to be stored in non-volatile memory, but need to be accessed quickly. The current program stores a 16-bit sine lookup table and the measured cavity transfer function in this ROM.

One of the DSP's serial ports is connected to a Texas Instruments DAC8554 4-output digital-to-analog converter. The DAC has a maximum sample rate of 200 KSPS on all outputs [17]. Output A is used to produce driving signal for the cavity's piezoelectric tuner, and output B can be used to produce a slow tuning signal, which can be used to compensate for slow drifts in the resonant frequency of the cavity. The other two outputs are currently unused and are available for future applications. The cavity tuner signals are connected to an operational amplifier to amplify their signals.

Although the DSP has 1 MB of internal RAM, an external 16 MB SDRAM is provided in case future programs need more memory for their calculations. The SDRAM is 32-bits wide and operates at 125 MHz.

The DSP printed circuit board also includes a temperature sensor to measure the temperature inside the module. It is used as a status indicator to signal if the module is not getting enough airflow to prevent overheating.

3.2 Control Program Overview

The control program is set up as a multi-threaded program; with each thread being triggered by a different hardware or software interrupt as necessary. Higher priority interrupts are allowed to preempt lower priority threads. Receiving a data word from the host processor

triggers the highest priority thread. The main control thread has the second highest priority, and is triggered by a counter, so that it runs at fixed intervals. The final hardware-triggered thread runs much less frequently than the other two, and does some low-priority status calculations. The threads triggered by software interrupts are used for routines that take a long time to perform and need to be done outside the hardware interrupt threads to prevent them from missing their deadlines.

3.3 Data Received From Host Processor

The serial connection between the DSP and host processor is full duplex and uses the same clock and synchronization signals for data traveling in both directions [18]. This means that the DSP must have data buffered and ready to transmit back to the host processor when the next word is received, because data is transmitted simultaneously in both directions as shown in Table 3.1. If the DSP is unable to respond to a command before the next command is received, incorrect data could be displayed on the user interface, so the thread that processes host commands was given the highest possible priority. This guarantees that the DSP will finish processing the current command and have a response ready for the host processor before the next command is received.

Each data word sent between the DSP and host processor is 32-bits, and is sent most significant bit first over the host processor's SPI port. Each command from the host is broken into three parts: the first bit is a 0 for a write and 1 for a read (this bit is ignored for read-only registers), the next 15 bits are a register address, and the last 16 bits are an optional argument. The exception to this format is when the host is sending a new program

Table 3.1. Responses from the DSP are received by the host two words after the command was transmitted.

Transmission Word	Command to DSP	Response to DSP
1	Command 1	Null Response
2	Command 2	Null Response
3	Command 3	Response 1
4	Command 4	Response 2
5	Command 5	Response 3
6	Command 6	Response 4
7	Command 7	Response 5
8	Null Command	Response 6
9	Null Command	Response 7

to the DSP. In this case, the host initiates the transfer by writing the program number and block number to register 0x0102, and then it sends a 64 kB block of the new program to the DSP. After the block of data is done being programmed, the host processor repeats the process for the remaining blocks (if necessary).

The DSP response format varies depending on the command it has received. If it receives a command to update a setting, it echoes back the setting to acknowledge that the setting has been updated, or in cases where the setting cannot be changed, the DSP replies with -1 . If the host processor reads a register, the DSP simply sends back the value of that register in whatever format it uses for calculations; the host must then convert the value to a meaningful format before sending it to the GUI.

3.4 Control Thread

The control thread has the second-highest priority. It runs at fixed intervals based on one of the DSP's internal timers. In the current setup, this thread runs 16,387 times per second,

but the timer can be modified to run at virtually any interval. The primary tasks for this thread are to perform the phase PID, amplitude PID, and adaptive feedforward cancellation (AFC) algorithms. Additional lower priority tasks are performed by other threads..

The control thread begins by reading the phase and amplitude of each input channel from the FPGA. It also stores the I and Q values of the cavity output in a buffer so that a fast Fourier transform (FFT) can be computed. If the FFT buffer is full, this thread posts the software interrupt that initiates calculation of the FFT. The FFT is used to identify low frequency (0 to 1000 Hz) sinusoidal disturbances affecting the cavity. Next, if a user is tuning the PID parameters and has pulsing enabled, the thread takes care of switching between the two phase or amplitude set points. After this is done, the thread runs the PID, cavity transfer function measurement, and AFC algorithms (if enabled) and updates the output phase and amplitude and sets the fast tuner. The cavity transfer function cannot be calculated correctly if either PID or AFC control is enabled, so logic is implemented to prevent them from being enabled at the same time. This is because the cavity transfer function algorithm assumes the cavities RF input has constant amplitude and phase, and it uses the fast tuner output to shake the cavity.

3.5 Status Thread

The final hardware thread runs much less frequently than the other two, and performs some low-priority tasks. It is triggered by a read from the ADC connected to the temperature sensor on the DSP's circuit board. The read occurs at the maximum possible interval allowed by the DSP's serial port hardware. This thread calculates the board temperature in

Celsius and the average DSP load over the previous interval and stores the values until the host processor requests them.

3.6 Software Interrupts

There are several threads that are implemented as software interrupts. They are used to perform tasks that take a long time to run and are not time critical. If any of these tasks were implemented within the hardware threads that initiate them, the hardware thread would fail to complete in its allotted time.

The highest priority software thread is the one that computes an FFT on a set of complex data samples. It is based on the `DSP_fft16x32` function included with the DSP programming tools [19]. The function computes an FFT on 32-bit complex data samples and uses a 16-bit “twiddle factor” look-up table to speed up processing. There are two versions of this function included with the DSP programming tools. The first is a hand-optimized assembly language version that is extremely fast, but is not interruptible. The second is an implementation of the same algorithm written in C. Although the C version is not as fast as the optimized version, it is interruptible, and is therefore the better choice for this system. The FFT has complex output, but we are only interested in seeing the magnitude response in the GUI, so as a final step, the FFT thread runs a 32-bit CORDIC routine on the complex FFT output to find the magnitude for each frequency. The output is saved until the host processor reads it and initializes the next FFT calculation.

The last two threads are used to program memory into the flash EEPROMs that are connected to the DSP. One thread programs the measured cavity transfer function into

ROM, so it is not lost at reset or power down. The other thread programs a 64 KB block of the program ROM. Although these threads have almost no calculations in them, they take the longest to complete. Before programming a ROM, the corresponding memory block must be erased, which typically takes around 800 ms. Then, each word can be programmed at a typical rate of approximately $10 \mu s$ [20]. As a result, the time spent in these threads is mostly spent in an idle loop, polling the ROMs to find out when they are ready for the next word to be programmed.

CHAPTER 4

Control

4.1 RF Amplitude and Phase Control

Cavity control is typically done using a PI controller, so a PID control was implemented to control the RF output, using feedback via the controller's cavity input. The DSP program allows separate PID control of the phase and amplitude of the signal, and some other tools have been included in the GUI to ease the PID tuning process. When operating at RF frequencies near a cavity's resonant frequency, the cavity behaves approximately linearly, so we can model the relationship between the RF driving signal and the cavity output using a transfer function. Figure 4.1 shows the closed-loop system. During operation, the RF frequency is fixed, but the amplitude and phase can be adjusted. Therefore, the system input and output are complex numbers. Since the system has a pole at $s = 0$, the system is type 1 and values can be chosen for k_p , k_i , and k_d to ensure the system response to a step input has zero steady-state error [21].

The transfer functions of the cavities being controlled vary dependent on the cavity type and amplifier system used, so the PID parameters must be tuned when equipment is changed. This tuning has historically been done by manually turning the RF output on and off, measuring the system response, and adjusting the PID parameters as necessary.

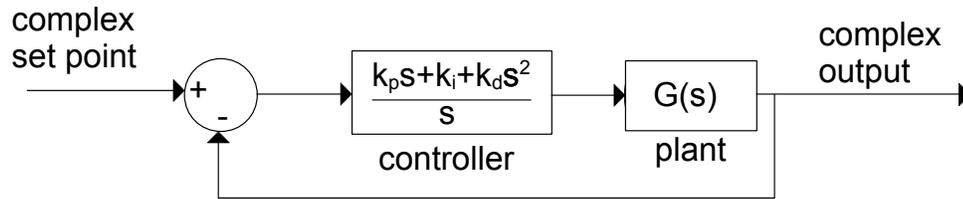


Figure 4.1. Block diagram of PID control. The cavity behaves linearly when operating near its resonant frequency, so it can be approximated by the transfer function, $G(s)$.

The DSP speeds up this process by automatically stepping between two set points and displaying the step response in the GUI. As the users change the PID parameters, they can immediately see the change in the system response. The PID loop compensates for small non-linearities and slowly varying drifts.

4.2 Cavity Tuning

We expect that most disturbances, not controlled by the PID alone, that will affect the cavities in the reaccelerator will be of the narrowband sinusoidal variety and due to rotating machinery, such as vacuum pumps. In order to cancel out these disturbances, the adaptive feedforward control developed by Tarek Kandil [22] was implemented in this module. The DSP is programmed to cancel disturbances at up to 10 frequencies. A block diagram of a single-frequency control is shown in Figure 4.2. $G_T(s)$ is the transfer function from the cavity tuner output to the phase error seen in the cavity output signal. The tuner has the affect of shifting the cavity resonant frequency, which results in an error in amplitude and phase as seen by the controller. When operating at RF frequencies near the cavity resonant

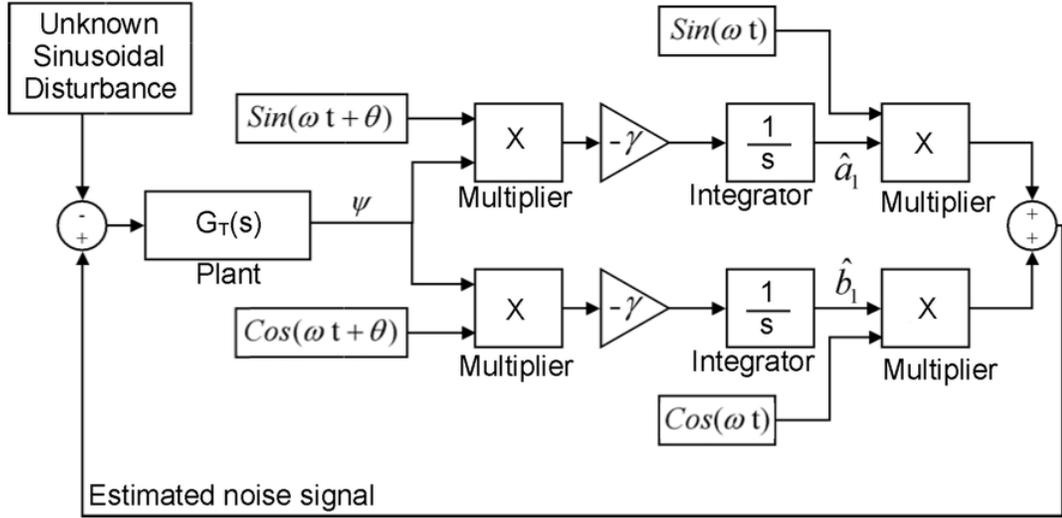


Figure 4.2. Adaptive feedforward cancellation control for the cancellation of a single sinusoidal disturbance [22].

frequency, a given change in tuner output has a relatively large effect on the phase error compared to its effect on the amplitude error, so the phase error is used for the adaptive feedforward cancellation algorithm.

The disturbances we expect to encounter are made up of multiple sinusoidal signals:

$$d = \sum_{i=1}^n A_i \sin(\omega_i t + \beta_i) \stackrel{\text{def}}{=} \sum_{i=1}^n a_i \cos(\omega_i t) + b_i \sin(\omega_i t) \quad (4.1)$$

Before the adaptive feedforward cancellation algorithm can be applied, we need to know the frequency of the disturbance to be cancelled and the phase of $G_T(s)$ at that frequency. To calculate the phase of $G_T(s)$, we start with the RF amplitude and phase set to fixed values (PID disabled), and drive the tuner while measuring the cavity response. The tuner is driven in 1 Hz increments from 1 Hz to 1 kHz, and the magnitude and phase response of the

cavity phase error is measured at each frequency. The measured Bode plot is representative of the system transfer function, $G_T(s)$. The plot is stored in non-volatile memory so it is not necessary to calculate it every time the controller is powered down or reset. The disturbance filters use this information from the measured Bode plot to stably apply the feed-forward algorithm. The value of each ω_i is determined by performing an FFT on the cavity error signal ψ and looking for the frequencies with the highest magnitudes. However, a_i and b_i are unknown, and must be estimated. We estimate these two parameters by multiplying the error signal by a gain factor and two orthogonal signals of known phase and then integrating the results. The control output is then created from the estimated values as:

$$u = \sum_{i=1}^n \hat{a}_i \cos(\omega_i t) + \hat{b}_i \sin(\omega_i t) \stackrel{def}{=} \sum_{i=1}^n \hat{A}_i \sin(\omega_i t + \hat{\beta}_i) \quad (4.2)$$

As long as the measured phase delay θ is within 90° of the actual phase delay and γ is chosen sufficiently small, the estimated values \hat{a}_i and \hat{b}_i will converge to the actual values a_i and b_i , as shown in Chapter 3 of [22]. γ controls the rate at which the estimated parameters adapt to changes in the actual values, so increasing γ can cause the control to also dampen disturbances that are close in frequency to ω_i [23]. This occurs because the large γ allows the adaptive parameters to continuously change phase to match the frequency of the disturbance, even though it is not the exact frequency the algorithm is set up to cancel.

CHAPTER 5

The ZWorld RabbitCore Host Processor and Graphical User Interface

The ZWorld RabbitCore 3200 microprocessor acts as the host processor for the module. The RabbitCore 3200 was selected because of the large amount of code already developed for it for other systems at the NSCL, but any other processor that supports the serial peripheral interface (SPI) protocol and has a sufficient number of I/O pins could be used instead. The host performs initial chip configuration for all the integrated circuits used in the module. After configuration is complete, the host processor acts as an interface between the GUI and the DSP. It forwards commands received via its network interface to the DSP and passes data received from the DSP over Ethernet back to the GUI. In addition, it also supplies an interface based on the Modbus protocol to the main EPICS control system [8].

5.1 Module Initialization

After the system powers up, the host processor must initialize all of the other chips in the module. Most of this is done using the SPI before data transfer can begin. The high-speed ADCs must be set from test mode to normal operating mode, and the phase-locked loop

must be set to the correct multiplier value. The SPI interface is also used to program the FPGA. The RabbitCore module contains a large enough flash ROM to hold both its code and the FPGA's code in compressed format, so it is not necessary to download the FPGA code every time the module is powered up.

After the first three initialization steps are completed, the SPI is used for transmitting commands and data between the host processor and the DSP. The DSP has its own non-volatile memory for program storage, so the host processor does not need to send the DSP its program at startup as it does with the FPGA. The DSP is set up to select between up to four programs when it is reset to make testing and debugging easier. When modifications are made to the DSP program, the new code can be downloaded without erasing the previous versions. Programmers can then quickly switch between different versions of the code to verify that bugs in one version are not present in others. In its initial boot stages, the DSP checks the status of two of the RabbitCore's output pins to determine which program to load.

5.2 User Interface

After module initialization is complete, the host processor's primary task is to act as an interface between the user and the DSP. It passes user commands to the DSP using its SPI interface and passes data back to the user. Unlike previous versions of this project, which used a telnet interface, this version has a graphical user interface. It was written in QT so that it will compile and run on all major operating systems [24]. The GUI is much easier to use than the older telnet interface, since the user no longer needs to memorize all of the

commands. The GUI also has much more functionality, since it can display graphs of the input, cavity transfer function, and FFT of the cavity output.

5.3 Main Screen

A screenshot of the main controller screen is shown in Figure 5.1. The top part of the screen shows the amplitude and phase of each input channel. The phase is shown relative to the reference channel, and the reference phase is always shown as 0.

Below the input status is the attenuator setting. It accepts values between 0 dB and 63 dB. For maximum measurement accuracy, the attenuator should be set to a value that allows the inputs to be as close as possible to 2.3 V without going over.

Below the attenuator box are the RF output control settings. In the open-loop mode, I and Q values are calculated from the phase and amplitude set points and are sent directly to the high-speed output DAC. In the closed-loop mode, a PID control algorithm is used to set the cavity output to the desired settings. The phase and amplitude controls can be set to open-loop or closed-loop independently of each other.

The two graphs on the bottom show the CPU load for the DSP and the ZWorld host processor. The DSP load should appear fairly constant, with the exception that it will spike to 100% when calculating an FFT or writing data to a ROM. The spikes are normal and will not affect the ability of the DSP to run the control algorithms, because the control thread has higher priority than the other threads and will interrupt them as necessary.

The ZWorld Save button saves the current settings as the defaults to load after a power up or reset. The settings include all settings on the main screen, PID screen, and AFC

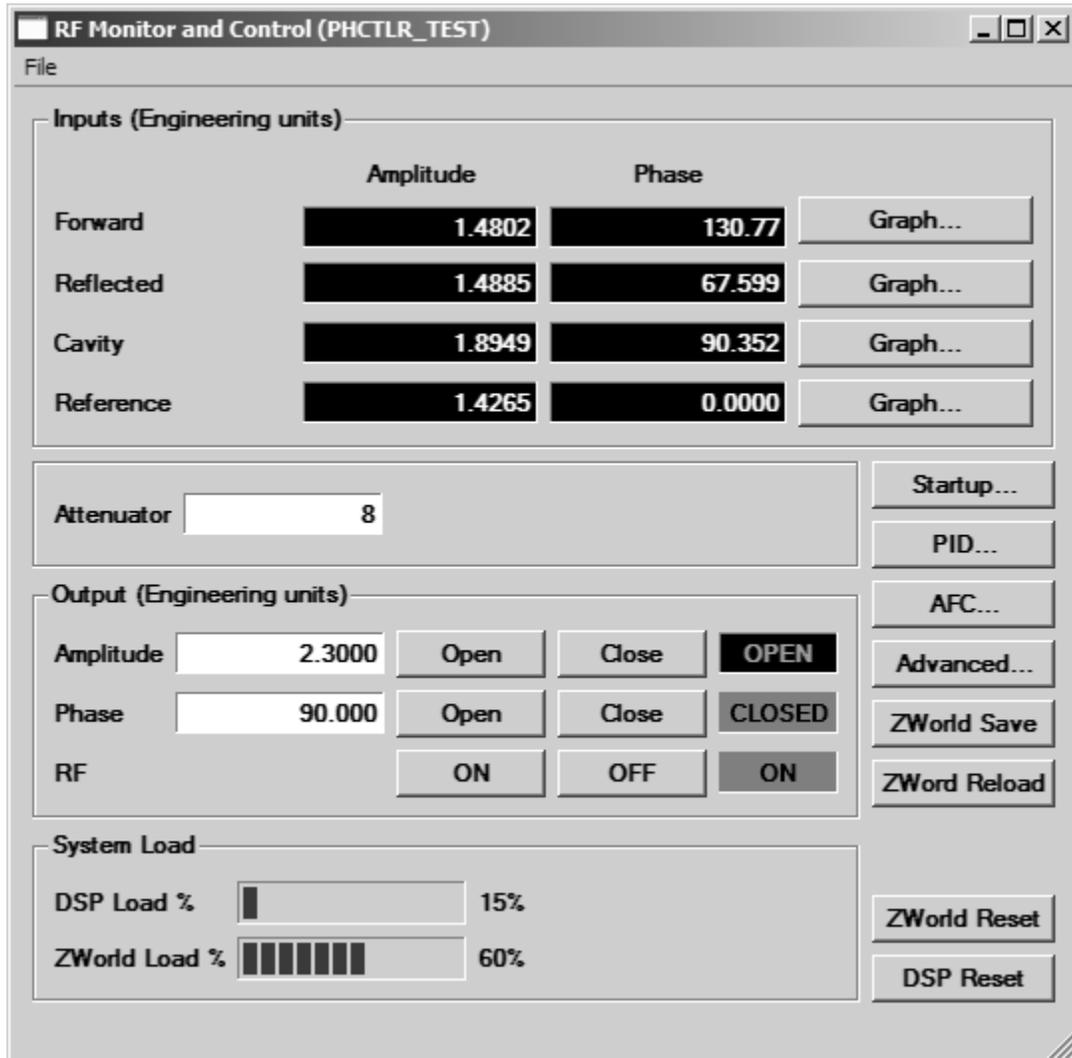


Figure 5.1. Main controller screen.

screen, with the exception of the RF On/Off status. The RF is always off by default after power up. The ZWorld Reload button returns all settings to their defaults without resetting the system. The ZWorld Reset and DSP Reset buttons allow individual resets of the ZWorld host processor and DSP. The DSP Reset button also allows the user to select which program to run on the DSP. The DSP can hold four different programs in separate pages of its program ROM. This allows new DSP code to be downloaded to the DSP's program ROM without overwriting the previous version of the program. If there is a problem with the new program, the DSP can be reset and told to reload the old version without putting the module into an unusable state. The rest of the buttons open screens discussed below.

5.4 PID Screen

The PID tuning screen was designed to ease tuning of the PID control loops by making it possible to see the system response in real-time as the control parameters are modified. The phase and amplitude PID control constants can be tuned independently of each other. To see the response of the cavity, the user simply has to set a nominal and pulsed value and click the run button to start stepping between the two set points. The controller will automatically step between the two set points and the GUI will show the step response. The stepping frequency can be changed to affect how much of the waveform appears on the screen at a time. Figure 5.2 shows the amplitude PID being tuned. It is best to turn off the AFC algorithm when tuning the PID parameters, because the error signal affects both the PID and AFC algorithms. Although the DSP code was written to minimize the effect of the step signal on the AFC algorithm, there is still some ringing produced by the tuner if

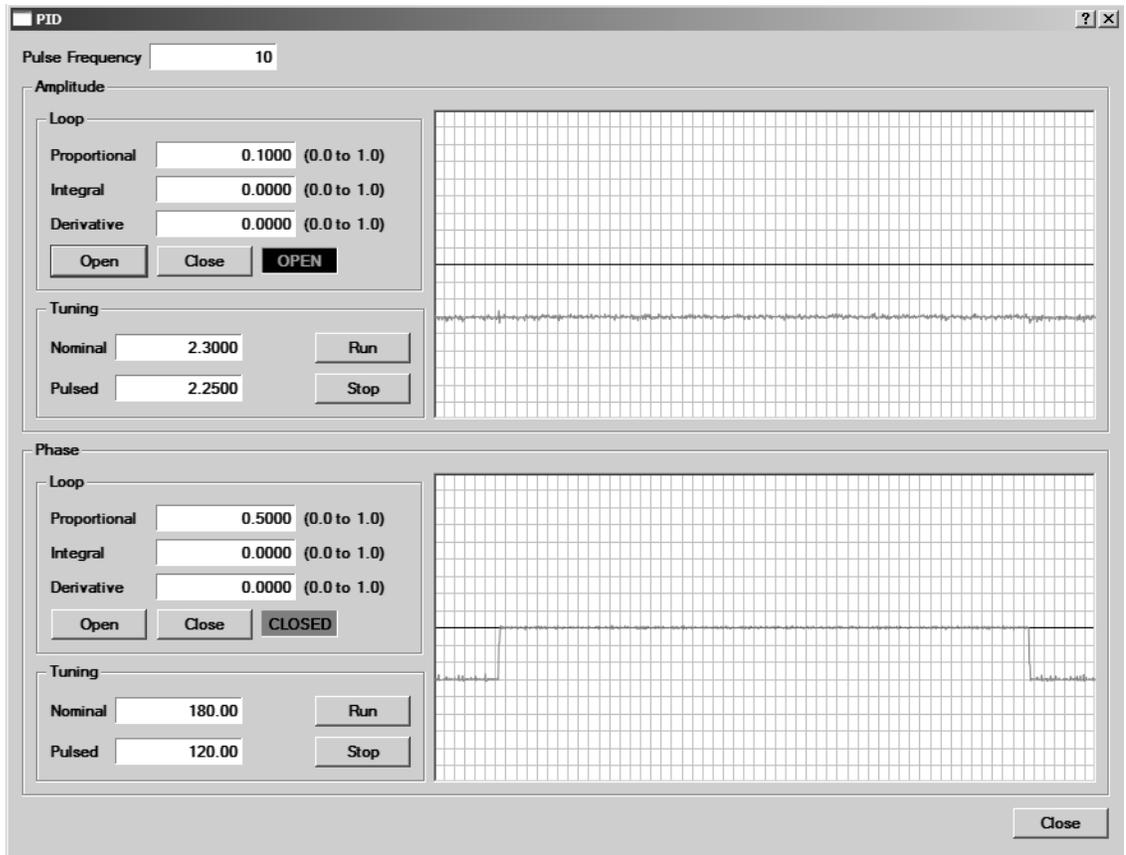


Figure 5.2. PID tuning screen.

it is enabled while tuning the PID parameters.

5.5 AFC Screens

There are three main parts of the AFC screens, the FFT magnitude plot, cavity transfer function, and the AFC cancellation screen. The DSP measures the cavity transfer function by outputting a sine wave on the cavity tuner and measuring the cavity's response. The tuner output is swept from 1 Hz to 1 kHz to produce the graphs shown in the GUI. The cavity transfer function is also stored in non-volatile memory to prevent it from being lost

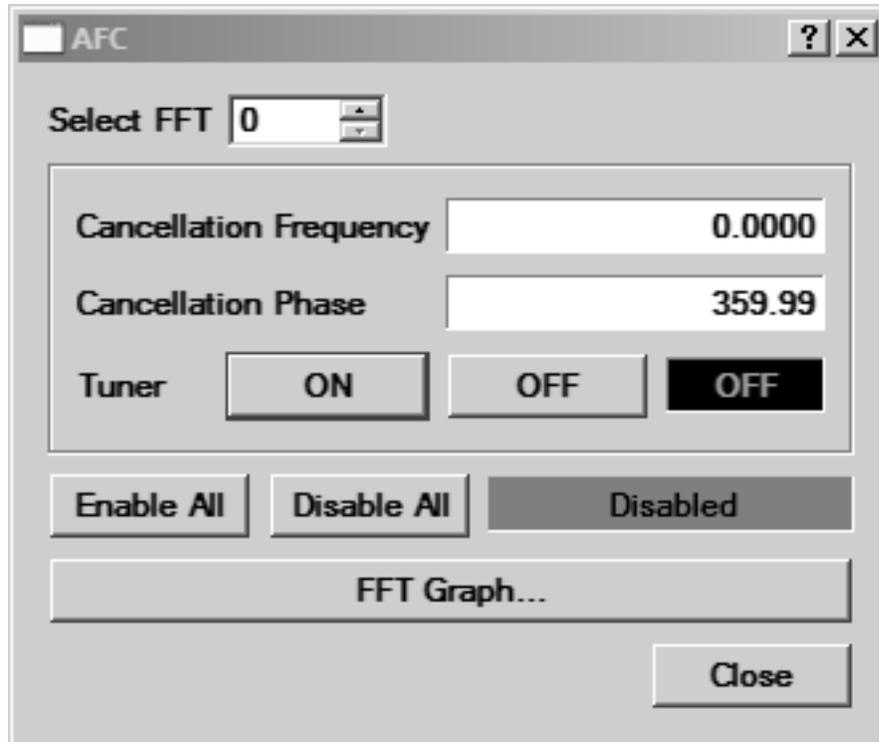


Figure 5.3. AFC screen.

during a reset. Automatic disturbance detection and cancellation has not been implemented yet, so it is up to the user to identify the frequencies at which disturbances are occurring. This is done by finding peaks on the FFT magnitude plot and entering their frequency into the AFC Cancellation Frequency box. The DSP automatically loads the previously calculated cavity phase delay for that frequency, and when enabled, outputs a disturbance cancellation signal at the selected frequency. Up to 10 frequencies can be enabled at a once. There are master enable and disable buttons for the tuner, which can be used to enable or disable the tuner without having to manually select every cancellation frequency.

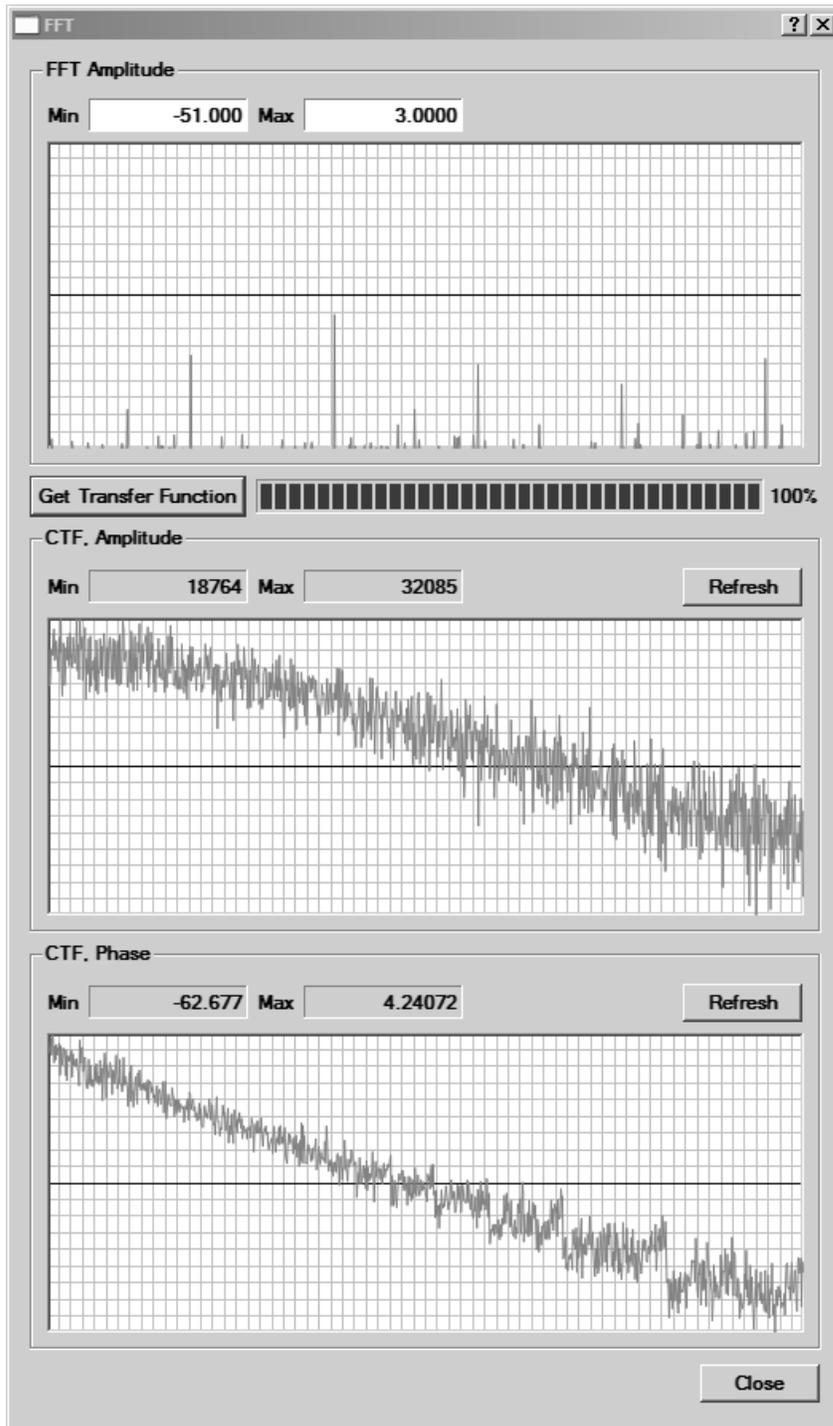


Figure 5.4. FFT screen shows FFT of cavity output in dBm, cavity magnitude response, and cavity phase response in degrees. All plots show 0 to 1000 Hz linear scale.

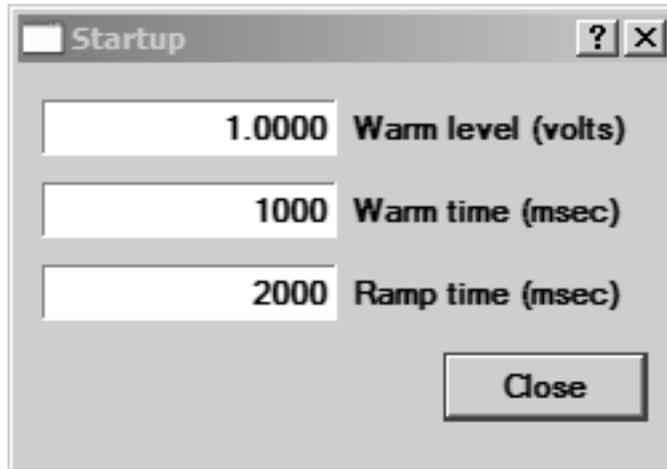


Figure 5.5. Startup screen.

5.6 Startup Screen

The module has a start-up mode, where its RF output gradually ramps to its set point after enabled. The user can specify an initial RF output level, the amount of time to stay at that level, and a ramp time to the amplitude set point. This feature allows the RF system to tune-up gradually before going full-scale and removes the initial overshoot that will occur if the PID parameters are under damped.

5.7 Advanced Screen

The Advanced screen contains features that are not used during normal operation. This screen contains buttons for downloading a new DSP program and calibrating the input channels. Program download is done over Ethernet, and can be done without affecting the program currently running on the controller. The DSP's program ROM can contain up to four independent programs, so a new program can be downloaded without overwriting



Figure 5.6. Advanced screen.

the one that is being run. When a program is being sent to the DSP, the GUI screens will temporarily stop updating, and commands sent via the GUI will be queued, but the module will continue to run as normal. When the download completes, GUI updates will resume and any queued commands will be sent to the module.

The channel calibration button will run a software routine that attempts to remove any constant phase or amplitude measurement errors between the three input channels, when it is implemented. These errors are primarily caused by slightly mismatched impedances on the input channels on the PCB, which prevent the amplifiers and attenuators from performing ideally. These variances cause each of the input channels to have a slightly different level of gain, which results in a fixed error factor between channels. Similarly, the digital attenuators do not attenuate in exact 0.5 dB steps, so at different levels of attenuation the amplitude error may vary. Also, during testing, a very small amount of phase error was measured depending on the attenuation setting for the digital attenuators.

CHAPTER 6

Testing and Performance

6.1 Cavity Simulation Circuit

SRF cavities are rarely available for testing purposes, so I built a test circuit to simulate a cavity with a known transfer function and known disturbances. A block diagram of the simulation circuit is shown in Figure 6.1. The test circuit has inputs for a disturbance signal, tuner signal from the LLRF controller, and the RF output from the LLRF controller. The disturbance signal and tuner signal are added together with a fixed DC voltage offset after the tuner signal passes through a transfer function block. The transfer function is designed to simulate a mass-spring model of the tuner arm. The DC offset is necessary to prevent the simulated disturbance signal from changing signs before it is multiplied with the RF signal, since a sign change would result in a 180° jump in phase of the output signal. In a real cavity, the disturbances cause the cavity resonant frequency to shift away from the driving frequency, in both amplitude and phase error. The phase error is typically used for disturbance identification, because it is relatively large compared to the amplitude error. The changes in phase cannot be easily modeled in a test circuit, so the test circuit simply performs amplitude modulation by multiplying the RF signal with the sum of the disturbance and tuner signals and the amplitude error is used for disturbance identification.

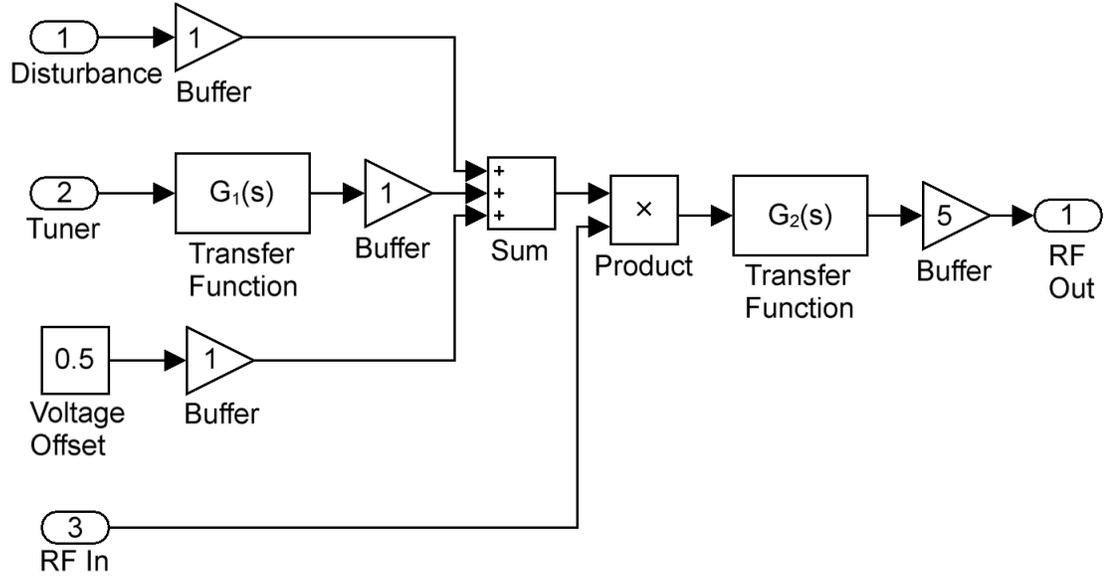


Figure 6.1. Block diagram of circuit used for cavity simulation. The two transfer functions are replaceable RC networks.

After modulation, the RF signal passes through another transfer function block before being amplified and sent back to the LLRF controller.

In order for the controller to perform the adaptive feedforward algorithm described in Section 4.2, it must be able to accurately calculate the phase response of the cavity. To test this, several different transfer function models were placed in the circuit, and MATLAB was used to calculate the phase response of the system. Each transfer function module consists of an RC network as shown in Figure 6.2.

By defining x_1 as the voltage across capacitor C_1 and x_2 as the output voltage, we can write the circuit's state equations [25]:

$$\dot{x}_1 = \frac{u - x_1}{C_1 R_1} + \frac{x_1 - x_2}{C_1 R_2} \quad (6.1)$$

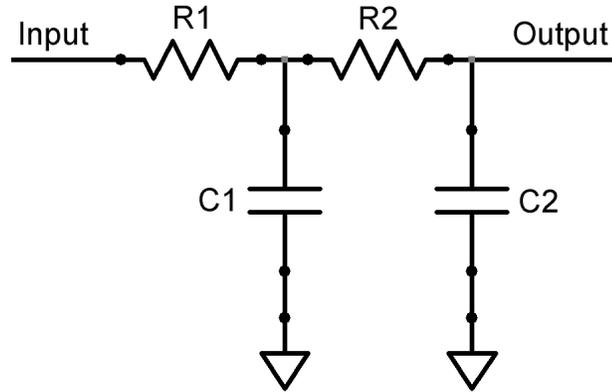


Figure 6.2. Each of the transfer function blocks is made up of a replaceable RC network.

$$\dot{x}_2 = \frac{x_1 - x_2}{C_2 R_2} \quad (6.2)$$

$$y = x_2 \quad (6.3)$$

The state model is then:

$$\dot{x} = \begin{bmatrix} \frac{R_1 - R_2}{C_1 R_1 R_2} & \frac{-1}{C_1 R_1} \\ \frac{1}{C_2 R_2} & \frac{-1}{C_2 R_2} \end{bmatrix} x + \begin{bmatrix} \frac{-1}{C_1 R_1} \\ 0 \end{bmatrix} u \quad (6.4)$$

$$y = \begin{bmatrix} 0 & 1 \end{bmatrix} x \quad (6.5)$$

This leads to a transfer function of:

$$H(s) = \frac{1}{C_1 C_2 R_1 R_2 s^2 + (C_1 R_1 + C_2 R_1 + C_2 R_2) s + 1} \quad (6.6)$$

Using this information, MATLAB can calculate the ideal phase response of the amplitude error of the circuit, which can then be compared to the phase response of the amplitude

error as calculated by the LLRF controller. The LLRF controller calculates the phase response of the amplitude error signal by slowly sweeping its fast tuner output from 0 Hz to 1 kHz. For each frequency, it records the incoming RF amplitude at four points, spaced 90 degrees apart, from which it can calculate the phase and magnitude response. During cavity sweep, there is assumed to be no disturbance and ω_T is much smaller than ω_{RF} , so the circuit's output can be approximated as:

$$V_{Out} = (A_T \sin(\omega_T) G_1(j\omega_T) + V_{Offset}) A_{RF} \sin(\omega_{RF}) G_2(j\omega_{RF}) \quad (6.7)$$

Since ω_T is much smaller than ω_{RF} , the tuner component can be viewed as a scaling constant on the RF signal, and the tuner input will cause a sinusoidal error in the amplitude of the RF output. Therefore, for the test circuit, G_T (Figure 4.2) can be approximated by $G_1(s)$. Figures 6.3 and 6.4 show the ideal amplitude and phase response and the response calculated by the LLRF controller in the presence of the disturbances it will attempt to cancel. The phase plots match very closely, with the calculated plot staying within approximately five degrees of the ideal plot for frequencies below 500 Hz and within ten degrees for frequencies between 500 Hz and 1,000 Hz. The saw tooth pattern seen in the plot created by the LLRF controller starting at about 600 Hz is an artifact of algorithm used to calculate the phase. To speed processing and make sure there were no errors in the original implementation of the phase calculation algorithm, the algorithm was implemented using integer intermediate values, instead of floating point variables. Although the phase calculation is accurate enough for the AFC algorithm to work, it can be made more accurate by using more precise intermediate results or filtering the output data to produce a smoother

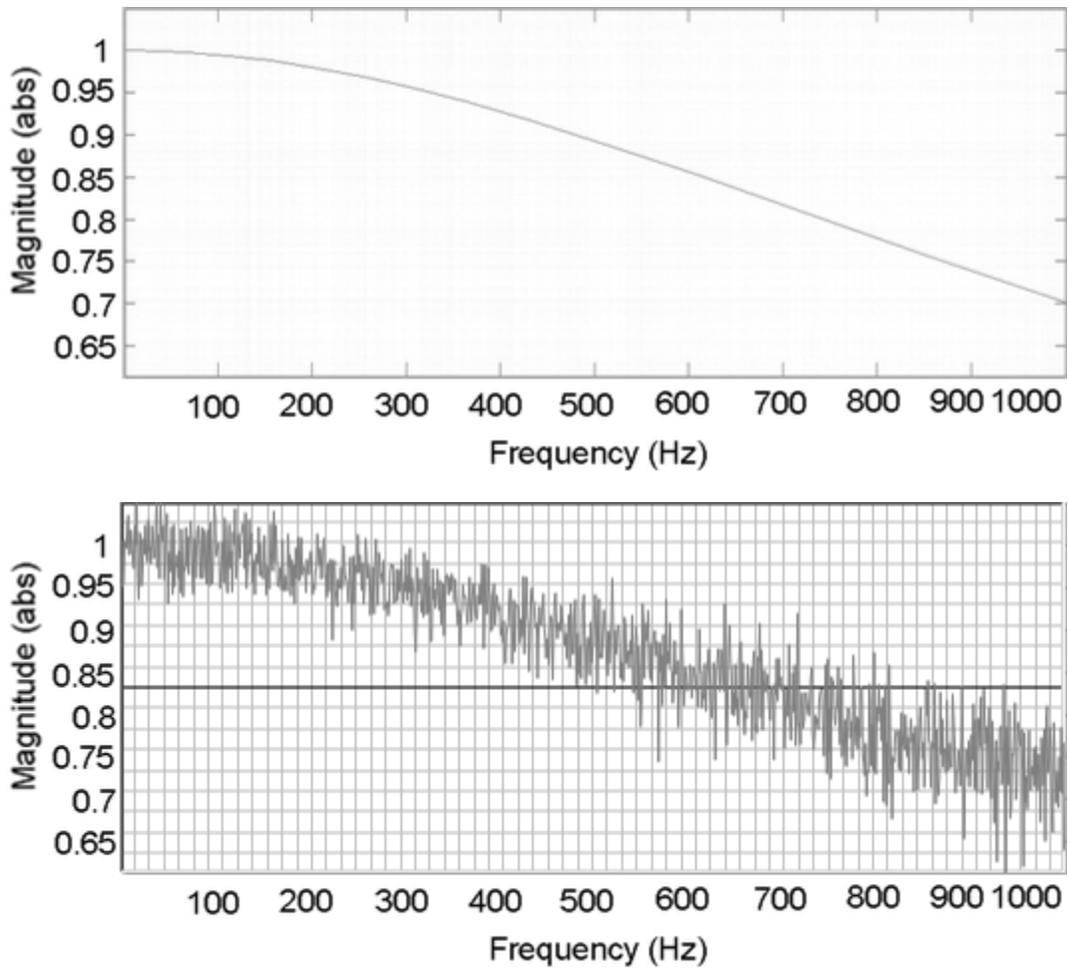


Figure 6.3. Ideal magnitude response (top) and magnitude response calculated by LLRF controller (bottom).

curve; so later versions of the DSP code will be modified to do so.

After verifying that the cavity transfer function calculation was accurate enough to work with the AFC algorithm, the controller was tested with known disturbances. In testing, the AFC algorithm was able to reduce the effect of disturbances to the noise floor with an appropriate value of the adaptation gain γ , regardless of the amplitude of the disturbance. Increasing the value of γ improves the effectiveness of the AFC algorithm by allowing the estimated parameters to converge faster to the actual parameters (Figure 6.5), but setting γ

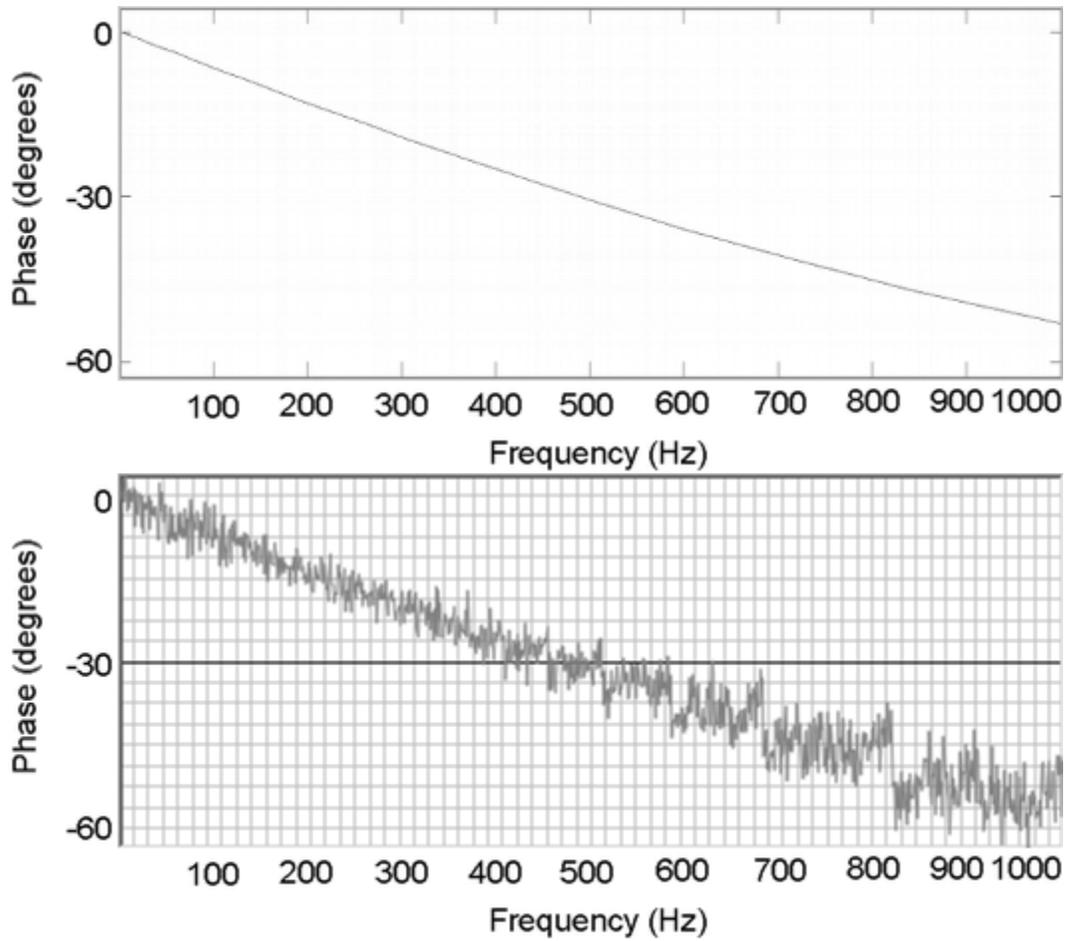
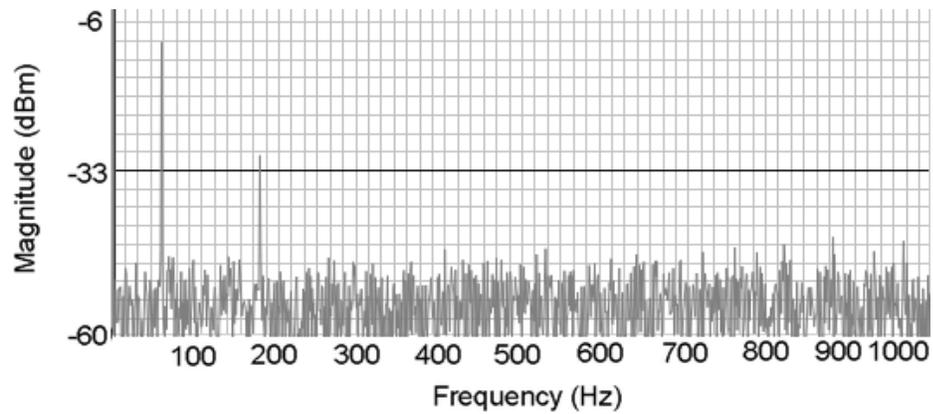
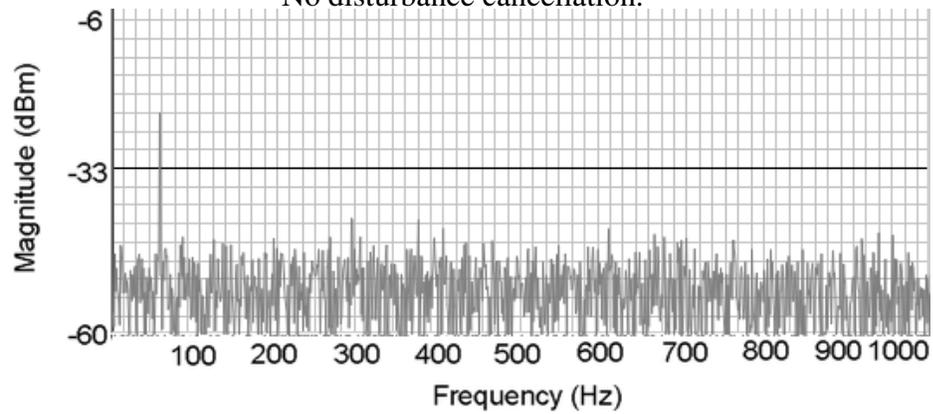


Figure 6.4. Ideal phase response (top) and phase response calculated by LLRF controller (bottom).

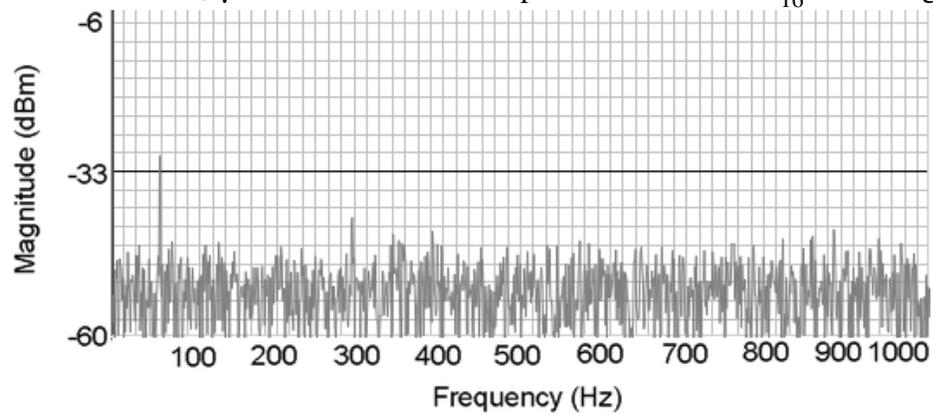
too high can cause instability, so its value must be chosen carefully. The spikes seen at 189 Hz, 378 Hz, and 568 Hz are caused by some unterminated digital connections within the controller. Those connections will be fixed in the next update of the circuit board.



No disturbance cancellation.



AFC cancellation at 60 Hz, $\gamma = 0.125$. Disturbance power is reduced to $\frac{1}{16}$ of its original level.



AFC cancellation at 60 Hz, $\gamma = 0.25$. Disturbance power is reduced to $\frac{1}{64}$ of its original level.

Figure 6.5. FFT of RF input with 60 Hz sinusoidal disturbance. Because the magnitude of the disturbance is large, its third harmonic is also present.

CHAPTER 7

Conclusion and Future Work

7.1 Conclusion

A new prototype LLRF controller has been created for use with high-Q superconducting cavities. In addition to the digital PID amplitude and phase control provided by typical LLRF controllers, the new prototype implements an adaptive feedforward cancellation algorithm for microphonics mitigation with a piezoelectric tuner, and is the first LLRF controller to do so. Microphonics mitigation is necessary, because the narrow bandwidth of high-Q cavities allows microphonics to perturb the resonant frequency of the cavity too far for the PID feedback control alone to work.

Testing was done to make sure that the AFC control does not conflict with the LLRF amplitude and phase control, and there were no conflicts during normal operation. However, during PID tuning the AFC can make it difficult to select the correct parameters. During PID tuning, the RF output steps between two set points so the system response can be seen. If the AFC control is enabled when tuning, it also tries to compensate for the error, since the step function has components at all frequencies. The result is what appears to be some ringing after each set point transition. Although steps were taken in software to minimize the effect on the AFC algorithm, it is currently not possible to completely prevent

the change in output set point from affecting the AFC algorithm.

7.2 Required Changes

Although the LLRF controller was shown to work in testing, there are several modifications that need to be made to the DSP daughter card before the module is ready for production. The biggest problem with the current version of the circuit board is that it runs very hot. The DSP and its voltage regulator are both designed to dissipate heat through the circuit board. However, the circuit board is simply not large enough to get enough airflow for the heat generated. On the prototype, heat sinks were added to the DSP and the voltage regulator to attempt to solve the problem. The heat sink on the DSP was able to sufficiently cool it, but there is not enough surface area on the top of the voltage regulator to add a sufficiently large heat sink to it. In order to solve the cooling problem, the DSP daughter card must be increased in area, and the voltage regulator should be moved farther from the DSP, since those two ICs produce almost all of the heat on the board.

In addition to the cooling problem, there were several minor errors made in the board schematic that made their way into the board layout. There were three pull-up resistors on the control lines for the DSP external memory interface that were mistakenly connected to ground instead of +3.3 V. Also, when programming the EEPROMs, it was found that it is better to let the software check the status of the EEPROMs during program and erase than to use the hardware control pins, so the busy pins should be disconnected from the EEPROMs.

Finally, DIP switches should be added to make it easier to change DSP hardware set-

tings. Switches should be added for the following settings: DSP clock multiplier, JTAG mode, DSP endian mode, DSP boot mode, SDRAM clock select, EEPROM and FPGA clock select, and default program.

7.3 Future Hardware Upgrades

During software development, several potential changes to the hardware were identified that would help improve data flow within the module as well as make sure that parts are available in the future. The FPGA currently used in the module is a Xilinx Spartan-II, which is an older model, which is not as fast and does not have as much logic as the newer Spartan-3 and Spartan-3E FPGAs. The Spartan-II will likely be phased out in the next few years, so it will be a good idea to migrate to the newer model.

Several signal routing changes could be made to the module to improve device utilization. There is an 8-pin DIP switch used to set the device address. In the current version, it is connected through the FPGA to the host processor. By connecting the DIP switch directly to the host processor, 16 FPGA I/O pins would be freed. By connecting these pins to the DSP to change its interface from 16 bits to 32 bits wide, the data throughput would be doubled. Currently, the DSP and host processor both have a free serial port, which could be connected to each other. The addition of a second serial connection between the devices would allow the large block data transfers (program download, FFT plot, etc.) and the control commands to be sent on different serial ports. Doing so would remove the lag in response to user commands during block transfers, since the host processor would no longer have to wait until the block transfer finishes to send the command.

The pass band of the RF output filter must be chosen based on the operating frequency range of the module. The current version of the RF circuit board accepts filters in the YY161 surface-mount package, but for higher frequencies there is not a very good selection of available filters. As a result, for some applications, it is necessary to add an external filter on the output to get adequate filtering. By changing the board layout to accept filters in a more common package, the need for external filters can be eliminated for the whole range of frequencies the module is designed to work with (2 MHz to 1.5 GHz).

7.4 Future Software Upgrades

In future versions of the LLRF control, the PID feedback control will be combined with a feedforward control to improve the system response. This will allow higher gain PID parameters to be used to improve accuracy without affecting the stability of the system. In addition, integrator anti-windup will be implemented to prevent the system output from saturating.

APPENDICES

APPENDIX A

FPGA Implementation of a CORDIC

Algorithm

```
/******  
/* CORDIC section (written in Verilog) */  
/* */  
/* A 16-bit CORDIC routine is used to convert sampled I/Q values */  
/* for each input channel to phase/magnitude ('A' registers). A */  
/* second implementation operates in the reverse direction to */  
/* convert the desired output phase and magnitude to I/Q ('O' */  
/* registers). */  
/******  
// Counter to keep track of stage of routine  
reg [3:0] ctr;  
  
// Phase look-up table  
// Each bit is 360 / 65536 degrees  
wire [14:0] lut;  
wire [15:0] lutn; // Negative of look-up table  
assign lutn = -lut;  
  
ROM16X1 #(.INIT(16'h8000)) lut14 (.O(lut[14]), .A0(ctr[0]),  
.A1(ctr[1]), .A2(ctr[2]), .A3(ctr[3]));  
ROM16X1 #(.INIT(16'h0001)) lut13 (.O(lut[13]), .A0(ctr[0]),  
.A1(ctr[1]), .A2(ctr[2]), .A3(ctr[3]));  
ROM16X1 #(.INIT(16'h0002)) lut12 (.O(lut[12]), .A0(ctr[0]),  
.A1(ctr[1]), .A2(ctr[2]), .A3(ctr[3]));  
ROM16X1 #(.INIT(16'h0004)) lut11 (.O(lut[11]), .A0(ctr[0]),
```

```

    .A1(ctr[1]), .A2(ctr[2]), .A3(ctr[3]));
ROM16X1 #(.INIT(16'h0008)) lut10 (.O(lut[10]), .A0(ctr[0]),
    .A1(ctr[1]), .A2(ctr[2]), .A3(ctr[3]));
ROM16X1 #(.INIT(16'h0012)) lut9 (.O(lut[9]), .A0(ctr[0]),
    .A1(ctr[1]), .A2(ctr[2]), .A3(ctr[3]));
ROM16X1 #(.INIT(16'h002C)) lut8 (.O(lut[8]), .A0(ctr[0]),
    .A1(ctr[1]), .A2(ctr[2]), .A3(ctr[3]));
ROM16X1 #(.INIT(16'h0056)) lut7 (.O(lut[7]), .A0(ctr[0]),
    .A1(ctr[1]), .A2(ctr[2]), .A3(ctr[3]));
ROM16X1 #(.INIT(16'h00A6)) lut6 (.O(lut[6]), .A0(ctr[0]),
    .A1(ctr[1]), .A2(ctr[2]), .A3(ctr[3]));
ROM16X1 #(.INIT(16'h0146)) lut5 (.O(lut[5]), .A0(ctr[0]),
    .A1(ctr[1]), .A2(ctr[2]), .A3(ctr[3]));
ROM16X1 #(.INIT(16'h028C)) lut4 (.O(lut[4]), .A0(ctr[0]),
    .A1(ctr[1]), .A2(ctr[2]), .A3(ctr[3]));
ROM16X1 #(.INIT(16'h0514)) lut3 (.O(lut[3]), .A0(ctr[0]),
    .A1(ctr[1]), .A2(ctr[2]), .A3(ctr[3]));
ROM16X1 #(.INIT(16'h0A22)) lut2 (.O(lut[2]), .A0(ctr[0]),
    .A1(ctr[1]), .A2(ctr[2]), .A3(ctr[3]));
ROM16X1 #(.INIT(16'h1474)) lut1 (.O(lut[1]), .A0(ctr[0]),
    .A1(ctr[1]), .A2(ctr[2]), .A3(ctr[3]));
ROM16X1 #(.INIT(16'h79DC)) lut0 (.O(lut[0]), .A0(ctr[0]),
    .A1(ctr[1]), .A2(ctr[2]), .A3(ctr[3]));

// Input I and Q read from high-speed ADC
reg [15:0] AI, AQ;

// Input registers written by DSP
reg [15:0] Op, Oa;    // Output phase and amplitude

// CORDIC outputs
reg [15:0] Ap, Aa;    // Input phase and amplitude
reg [13:0] OI, OQ;    // Output I and Q

// Intermediate values
reg [15:0] AI2;
reg signed [15:0] AQ2;
reg [15:0] Ap2;
reg signed [15:0] OI2, OQ2;
reg [15:0] Op2;

// Possible values for first stage of CORDIC routines
wire [15:0] AI1;
wire signed [15:0] AQ1;
wire signed [15:0] OQ1;

```

```

// Possible values for stages 2-16 of CORDIC routines
wire [15:0] AI3;
wire signed [15:0] AQ3;
wire signed [15:0] OI3, OQ3;

// First stage depends on the sign of the input Q value
assign AI1 = AQ[15] ? -AQ : AQ;
assign AQ1 = AQ[15] ? AI : -AI;

// First stage depends on the sign of the input phase
assign OQ1 = Op[15] ? -Oa : Oa;

// Stages 2-16 depend on the sign of the intermediate Q value
assign AI3 = AQ2[15] ? (AI2 - (AQ2 >>> ctr)) :
  (AI2 + (AQ2 >>> ctr));
assign AQ3 = AQ2[15] ? (AQ2 + (AI2 >> ctr)) : (AQ2 - (AI2 >> ctr));

// Stages 2-16 depend on the sign of the intermediate phase value
assign OI3 = Op2[15] ? (OI2 + (OQ2 >>> ctr)) :
  (OI2 - (OQ2 >>> ctr));
assign OQ3 = Op2[15] ? (OQ2 - (OI2 >>> ctr)) :
  (OQ2 + (OI2 >>> ctr));

always @(posedge clk40out)
begin
  ctr <= ctr + 1;

  // First stage: shift input +90 or -90 degrees
  // First stage occurs when counter = -1
  if (&ctr)
  begin
    AI2 <= AI1;
    AQ2 <= AQ1;
    Ap2 <= AQ[15] ? lutn : lut;
    OI2 <= 16'h0000;
    OQ2 <= OQ1;
    Op2 <= Op[15] ? (Op + lut) : (Op + lutn);

    // Store completed conversion values from last clock cycle
    Ap <= Ap2;
    Aa <= AI2;

    // Store completed conversion values from last clock cycle
    OI <= OI2;
  end
end

```

```
    OQ <= OQ2;
end

// stage 2-16: perform shift and update intermediate registers
else
begin
    AI2 <= AI3;
    AQ2 <= AQ3;
    Ap2 <= AQ2[15] ? (Ap2 + lutn) : (Ap2 + lut);

    OI2 <= OI3;
    OQ2 <= OQ3;
    Op2 <= Op2[15] ? (Op2 + lut) : (Op2 + lutn);
end
end
```

APPENDIX B

DSP Daughter Board Schematics

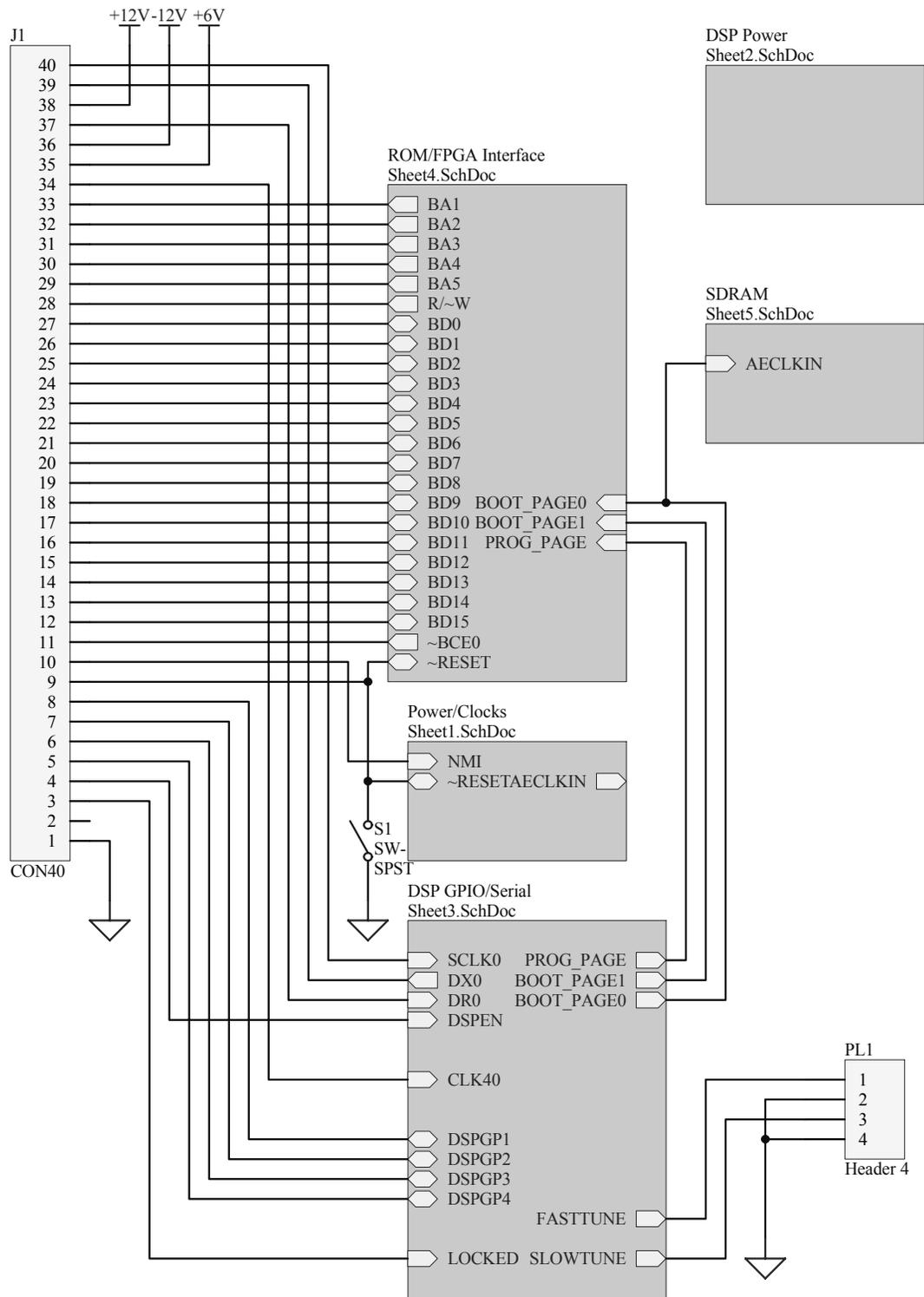


Figure B.1. System Overview.

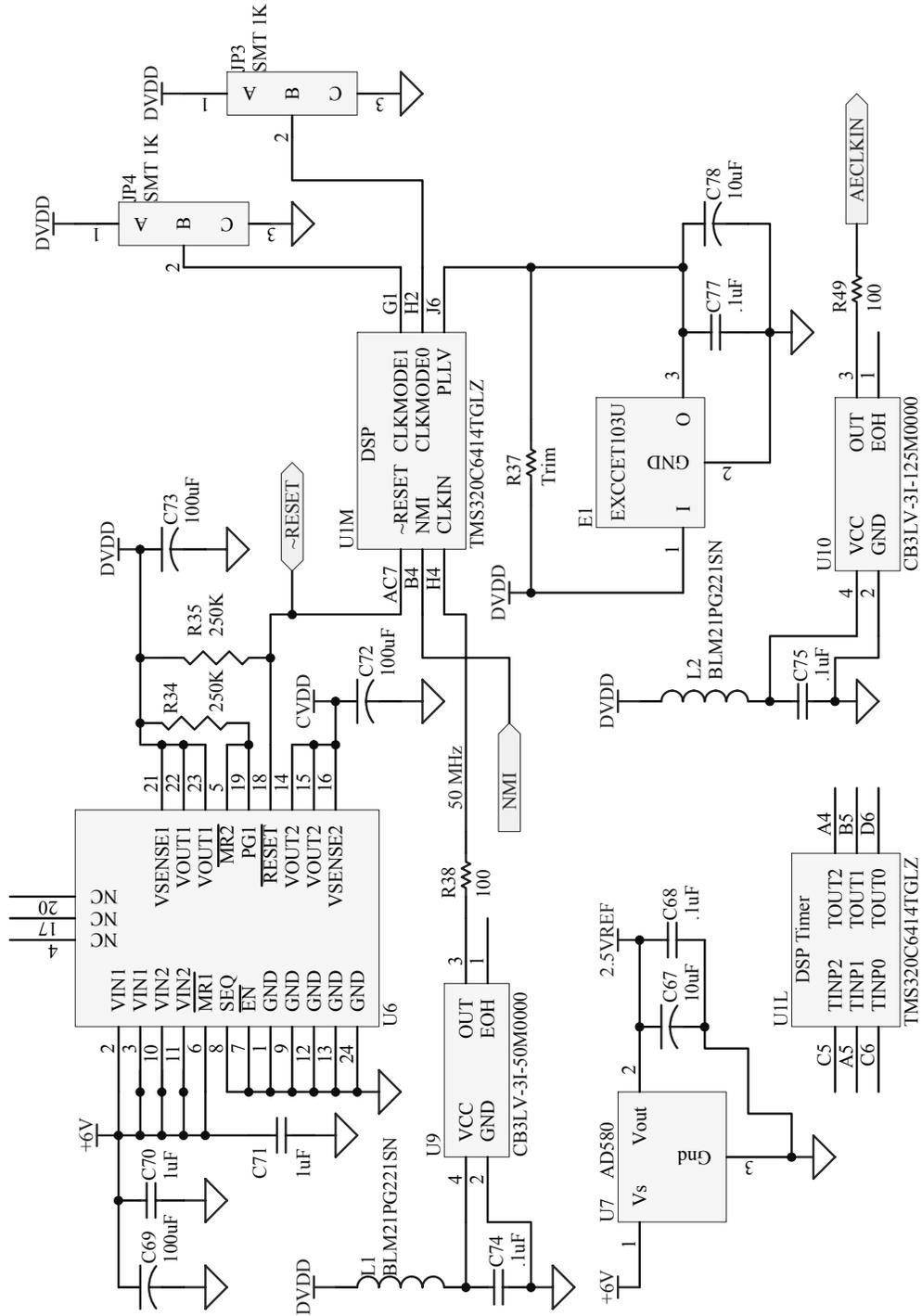


Figure B.2. Voltage regulation and clock generation.

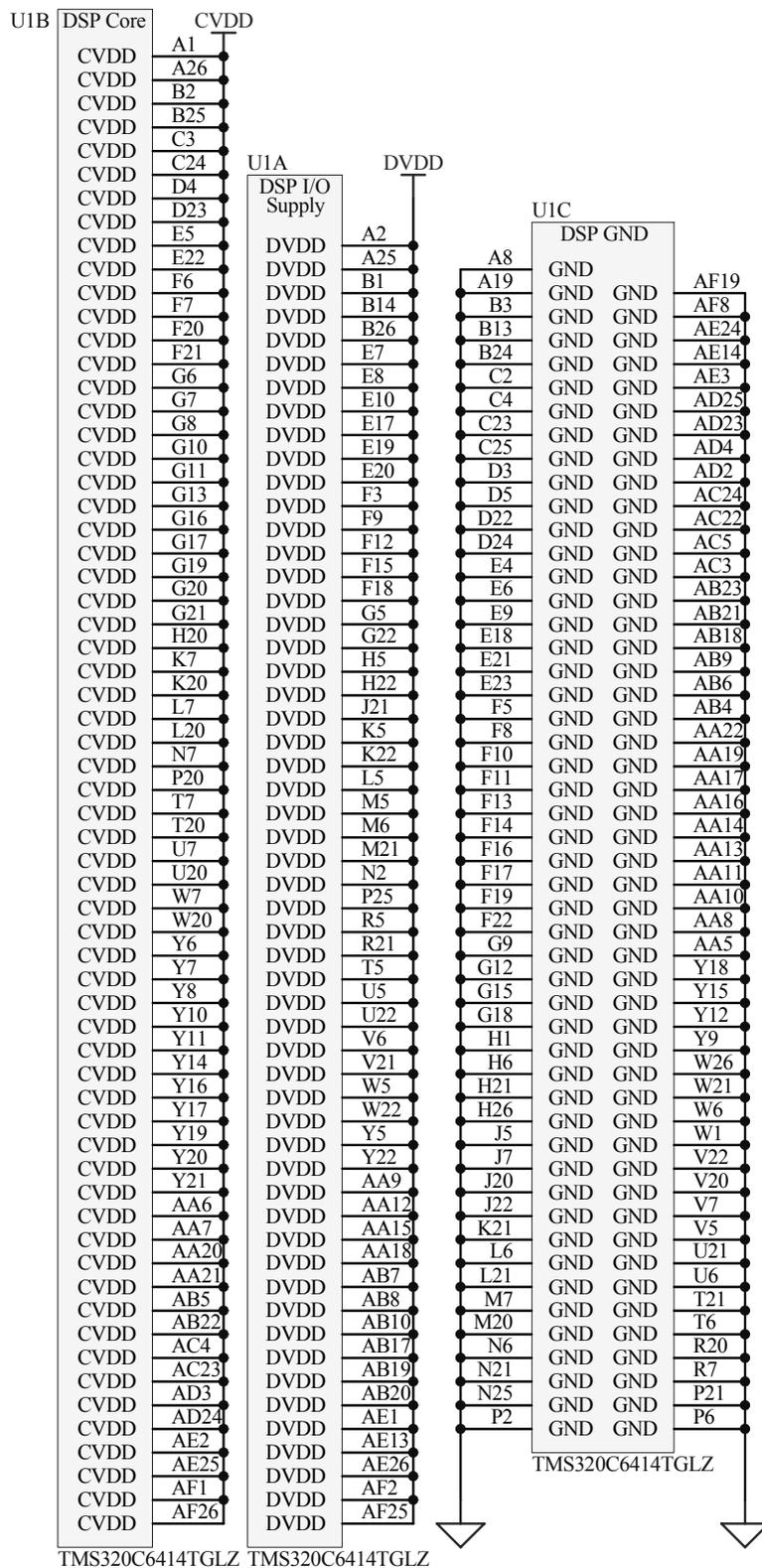


Figure B.3. DSP power connections.

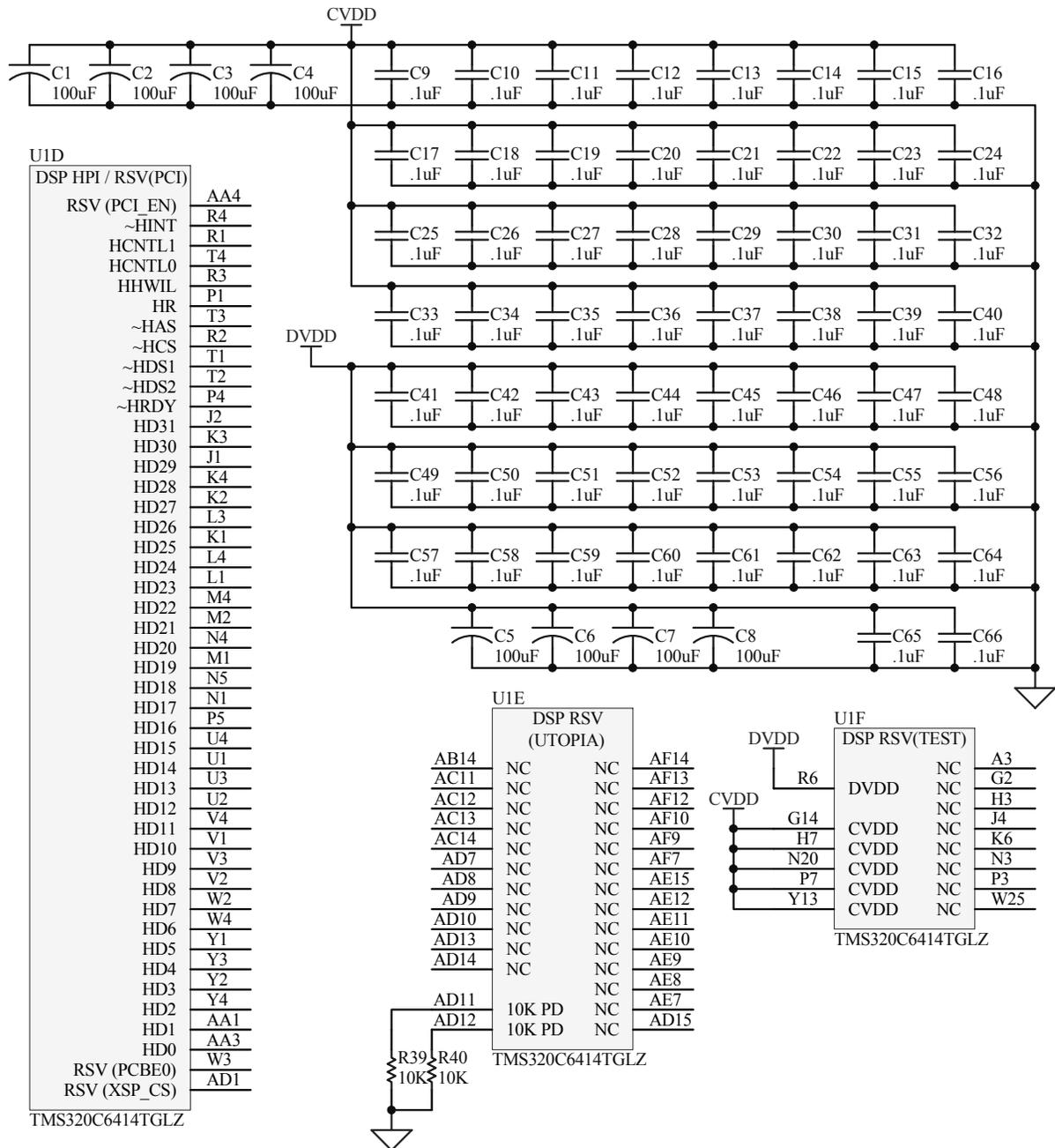


Figure B.4. DSP bypass capacitors, UTOPIA port, HPI port and reserved pins.

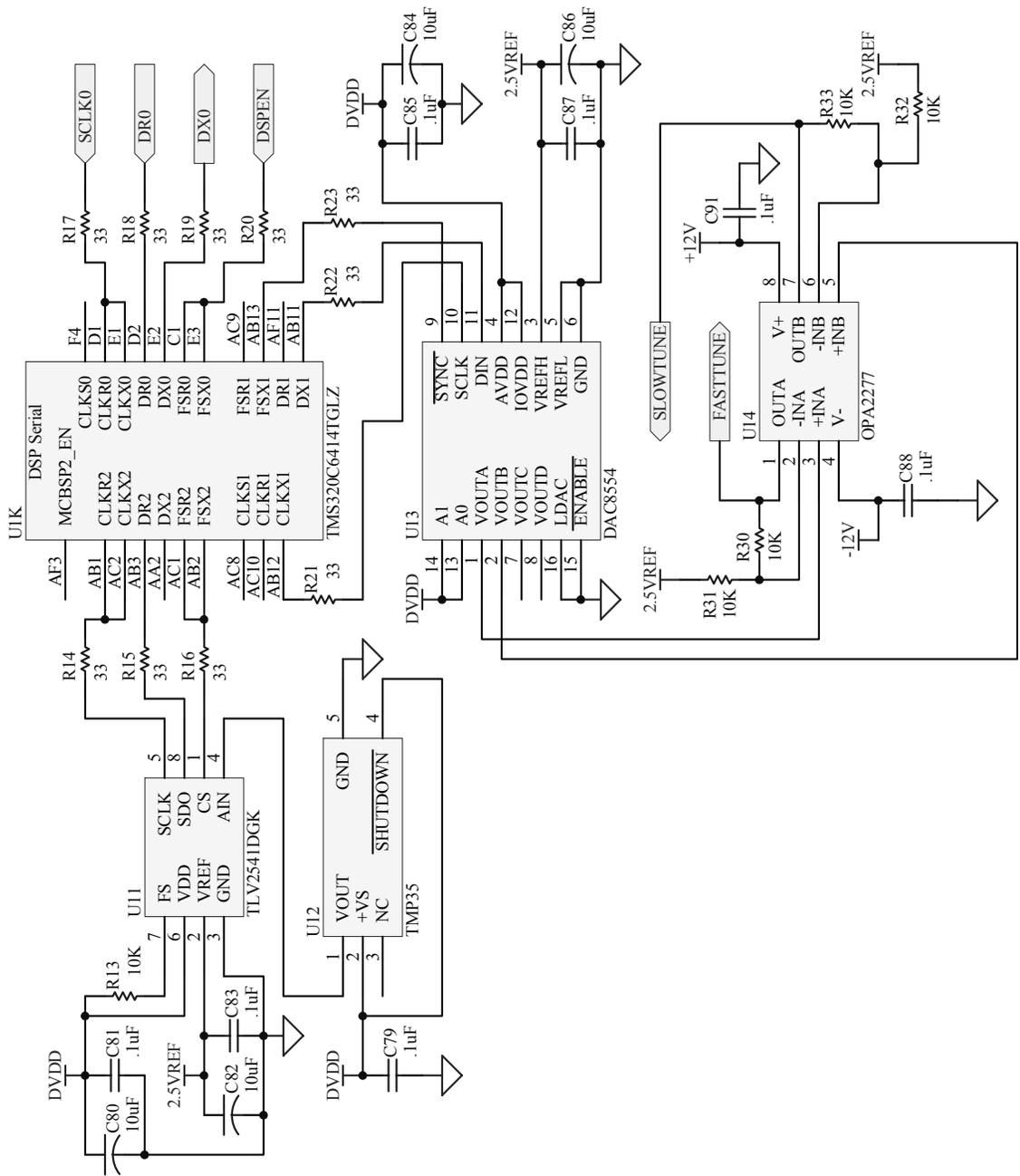


Figure B.5. DSP serial connections.

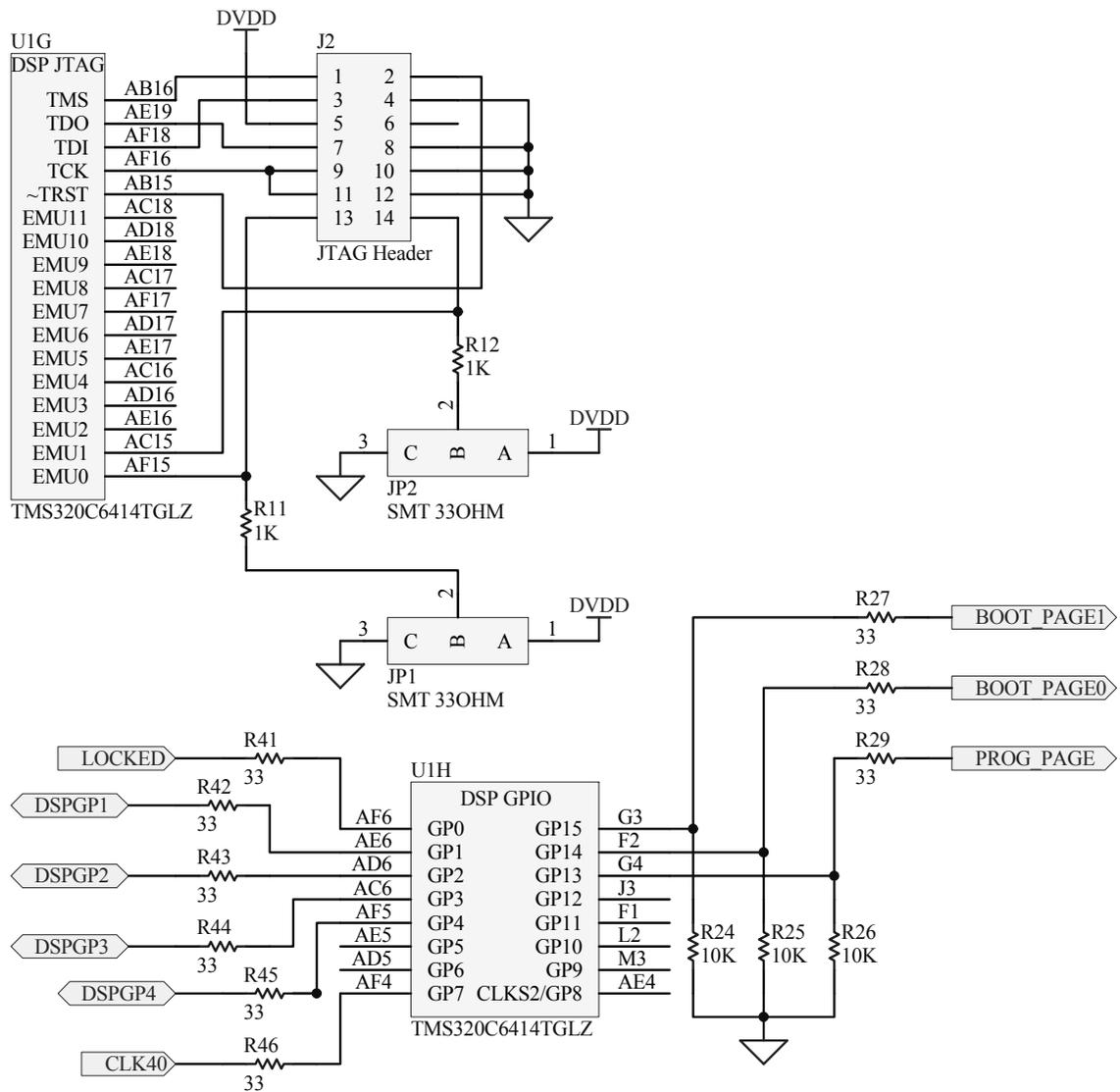


Figure B.6. DSP JTAG emulation port and general purpose port.

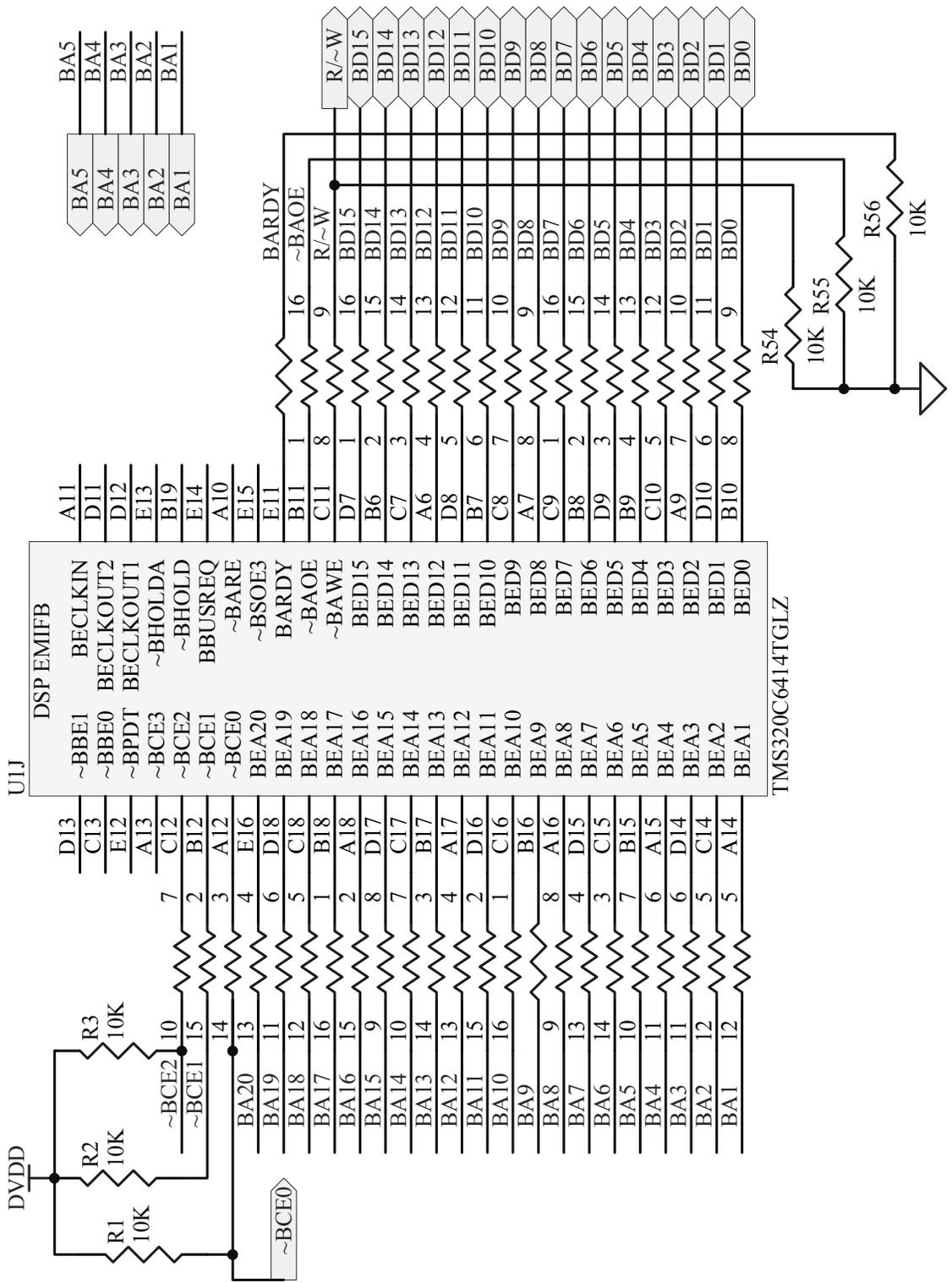


Figure B.9. DSP external memory interface B.

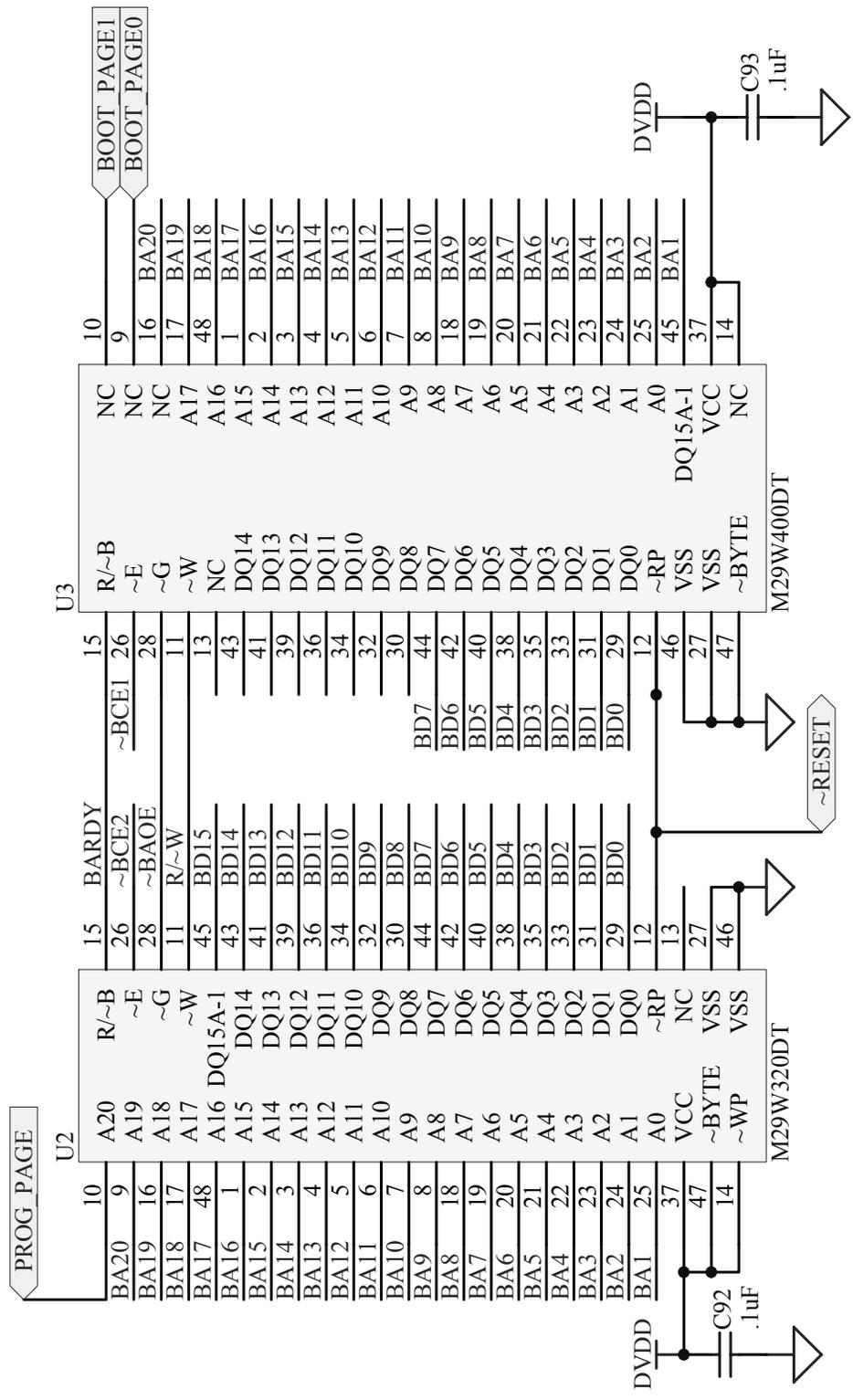


Figure B.10. External non-volatile memory and FPGA connection.

BIBLIOGRAPHY

BIBLIOGRAPHY

- [1] T. L. Grimm, W. Hartung, T. Kandil, H. Khalil, J. Popielarski, J. Vincent. *Measurement and Control of Microphonics in High Loaded-Q Superconducting RF Cavities*. Proceedings of the 2004 International Linear Accelerator Conference, pages 815-817, Lübeck, Germany, 2004.
- [2] S. Simrock, G. Petrosyan, A. Facco, V.Zviagintev, S. Andreoli, T. Paparella. *First Demonstration of Microphonic Control of a Superconducting Cavity with a Fast Piezoelectric Tuner*. Proceedings of 2003 Particle Accelerator Conference, pages 470-472, Portland, Oregon, 2003.
- [3] T. H. Kandil, H. K. Khalil, J. Vincent, T. L. Grimm, W. Hartung, J. Popielarski, R. C. York, S. Seshagiri. *Adaptive Feedforward Cancellation of Sinusoidal Disturbances in Superconducting RF Cavities*. Nuclear Instruments and Methods in Physics Research Section A, Volume 550, pages 514-520, September 2005.
- [4] M. J. O'Farrell. *Low Level Radio Frequency Control of RIA Superconducting Cavities*. Masters Thesis, Michigan State University, East Lansing, Michigan, 2005.
- [5] H. Ma, K. Kasemir, M. Champion, M. Crofford, T. Hardek, M. Piller, Y. Zhang. *Digital RF Control for Spallation Neutron Source Accumulator Ring*. Proceedings of Particle Accelerator Conference 07, pages 611-613, Albuquerque, New Mexico, 2007.
- [6] J. Branlard, B. Chase. *Survey of LLRF Development for the ILC*. Proceedings of Particle Accelerator Conference 07, pages 3810-3812, Albuquerque, New Mexico, 2007.
- [7] J. Weber, M. Chin, C. Timossi, E. Williams. *ALS Mini IOC: An FPGA Embedded Processor Based Control System Module for Booster Magnet Ramping at the ALS*. Proceedings of Particle Accelerator Conference 07, pages 3991-3993, Albuquerque, New Mexico, 2007.
- [8] S. A. Lewis. *Overview of the Experimental Physics and Industrial Control System: EPICS*. Lawrence Berkeley National Laboratory, Berkeley, California, September 1998.
- [9] L. Doolittle. *Low-Level RF Control System Design and Architecture*. Proceedings of Asian Particle Accelerator Conference 2007, Madhya Pradesh, India, 2007.
- [10] A. Molzahn. *Digital Phase Detection in a Variable Frequency RF System*. Masters Thesis, Michigan State University, East Lansing, Michigan, 2005.
- [11] J. E. Volder. *The CORDIC Trigonometric Computing Technique*. IRE Transactions on Electronic Computers, v. EC-8, No. 3, pages 330-334, September 1959.

- [12] A. Ambardar. *Analog and Digital Signal Processing*. Brooks/Cole, Pacific Grove, California, second edition, 1999.
- [13] R. Andraka. *A Survey of CORDIC Algorithms for FPGA Based Computers*. Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays, pages 191-200, Monterey, California, February 1998.
- [14] Texas Instruments, Inc. *TMS320C6000 EMIF to External Asynchronous SRAM Interface*. spra542a, Texas Instruments, Inc, August 2001.
- [15] L. R. Doolittle. *Plan for a 50MHz Analog Output Channel*. Lawrence Berkeley National Laboratory, Berkeley, California, August 2002.
- [16] Texas Instruments. *14-Bit, 165 MSPS Digital-to-Analog Converter*. Device Data Sheet, Burr-Brown Products, May 2002.
- [17] Texas Instruments. *16-Bit, Quad Channel, Ultra-low Glitch, Voltage Output Digital-to-Analog Converter*. Device Data Sheet, Texas Instruments, Inc, June 2005.
- [18] Texas Instruments. *TMS320C6000 DSP Multichannel Buffered Serial Port (McBSP) Reference Guide*. spru580g, Texas Instruments, Inc, December 2006.
- [19] Texas Instruments. *TMS320C64x DSP Library Programmer's Reference*. spru565b, Texas Instruments, Inc, October 2003.
- [20] STMicroelectronics. *M29W320DT M29W320DB*. Device Data Sheet, STMicroelectronics, November 2002.
- [21] G. F. Franklin, J. D. Powell, A. Emami-Naeini. *Feedback Control of Dynamic Systems*. Fifth Edition. Prentice Hall, Upper Saddle River, New Jersey, 2006.
- [22] T. H. Kandil. *Adaptive Feedforward Cancellation of Sinusoidal Disturbances in Superconducting Radio Frequency Cavities*. Masters Thesis, Michigan State University, East Lansing, Michigan, 2005.
- [23] M. Bodson, A. Sacks, P. Khosla. *Harmonic Generation in Adaptive Feedforward Cancellation Schemes*. IEEE Transactions on Automatic Control, Volume v. 39, i. 9, 1939-1944, September 1994.
- [24] *QT 4.3 Whitepaper*. Trolltech ASA, Oslo, Norway, 2007.
- [25] P. J. Antsaklis, A. N. Michel. *Linear Systems*. Mcgraw-Hill, Englewood Cliffs, New Jersey, 1997.