A 400MHz
Direct Digital Synthesizer
with the AD9912

Daniel Da Costa
danieljdacosta@gmail.com

Brendan Mulholland
firemulholland@gmail.com

Project Sponser:
Dr. Kirk W. Madison

Project 1160
Engineering Physics 479
The University of British Columbia

January 26, 2012

Part II

Testing of the Prototype Device

Contents

List of Figures

1

Summary

The DDS prototype board has been assembled and is pictured in Figure 1.1. Testing of a prototype AD9912 DDS device was performed and is documented here.

The digital interface was confirmed to work as expected and was found to be compatible with the Quantum Degenerate Gasses (QDG) Laboratory's UTBus system. Programming the AD9912 chip used on the DDS device was attempted but not accomplished; recommendations and suggestions for ongoing testing and debugging are presented. Nonetheless, sinusoidal RF output at a range of frequencies were observed due to default startup functionality of the AD9912. These RF outputs confirmed the basic functionality of the majority of the DDS device's functions.

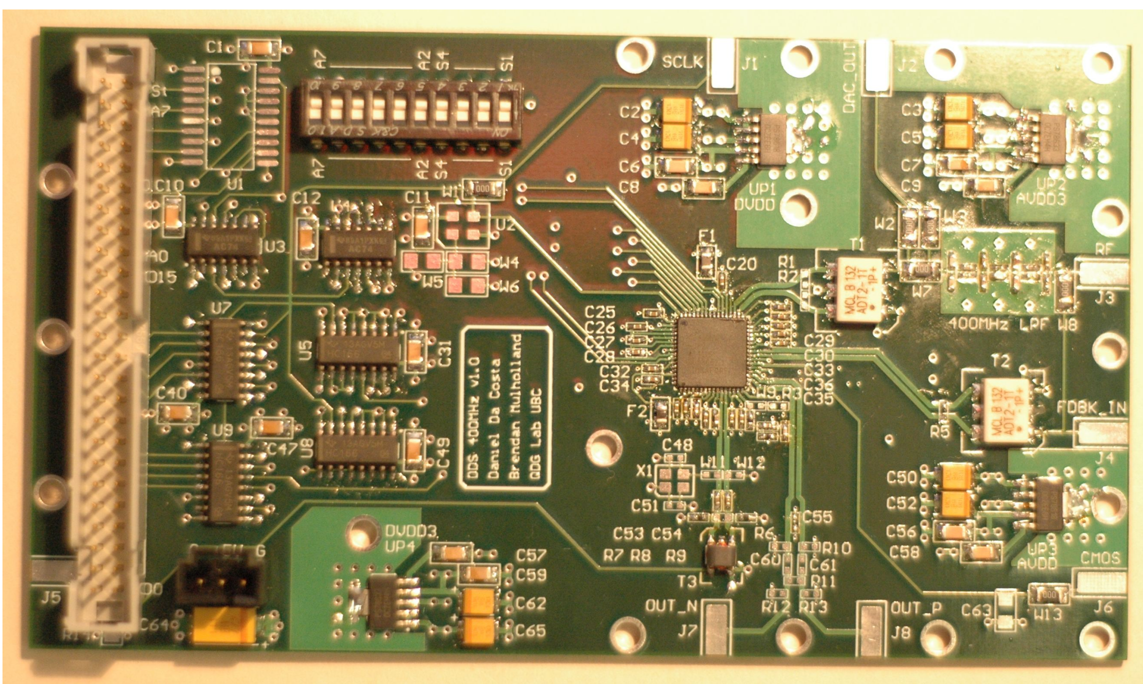

Figure 1.1: Photo of prototype AD9912 DDS device with components installed. Note that the comparator is missing from the photo (U1, the 20-pin IC in the top left) but has since been installed.

2

Testing

Functionality of the digital interface was tested and confirmed to work as expected. The startup mode of the AD9912 was configured to set the AD9912 to output one of a set of pre-programmed frequencies. These frequencies were successfully monitored on the DDS board RF output.

2.1 Digital Interface

The digital interface testing was broken up into two stages. Initially, testing used switches on a breadboard to generate static signals on the device's 50-pin parallel connector. This was tedious and meant a relatively long delay between commands. In the second stage, the prototype DDS device was hooked up to the QDG laboratory's UTBus system. Programming of the AD9912 was attempted but no evidence of successful register writing was found.

2.1.1 Breadboard Control of the UTBus

Initial prototype board testing was performed with a 5V power supply, two function generators - one each for SCLK and SYSCLK - an oscilloscope to monitor signals and a breadboard used to generate digital signals. This setup allowed for testing of the parallel-to-serial converter and supported attempts to program the AD9912.

Due to an issue with the ordering process, initial testing was performed without the address byte comparator. Instead, a wire was soldered to the PCB in place of the comparator output (parallel-to-serial converter enable, or EN). EN was controlled via a switch on a breadboard. The desired behaviour was observed; when $\overline{EN}$=0 the circuit is enabled and, on the rising edge of the strobe bit, the output appears on CSB and SIO. This behaviour is shown in Figures 2.1 to 2.3. When $\overline{EN}$=1, the circuit is disabled; SIO and CSB will remain 0 and 1, respectively. Also shown is the SH/$\overline{LD}$ bit; this controls the loading of the shift registers used to generate SIO and CSB. When low, the shift registers load the data in; they subsequently clock out the data one bit at a time.

As only a 2-channel oscilloscope was available, the data shown in Figures 2.1 to 2.3 was not collected simultaneously. In particular, the same data is used for SCLK and SH/$\overline{LD}$ for each figure. Note that, for this reason, in some cases the noise on CSB does not match up with changes in SIO. Further, for ease of testing, the UTBus was controlled manually using a bank of switches on a breadboard. Since the comparator was not installed, this did not include UTBus pins A1 to A7 (A0, or SZ, is needed as it controls the length of data transfers).

A function generator was used to drive SCLK. Although only 1MHz testing is shown, further testing at 5MHz was performed and no issues were found. Testing at frequencies above 5MHz was not performed but there were no indications that 5MHz is an upper limitation. As such, SCLK frequencies of tends of MHz may well be possible. However, timing issues may be present and detailed testing of higher frequencies would be required to confirm full functionality.

Compare these plots with the breadboard test results shown in Figure 2.19 of Part I of this report. As expected, the first bit of data is no longer held on the CSB or SIO output during the loading state of the shift registers. Thus the first bit of data is only held for one clock cycle, as designed.

Using the same setup, a series of commands that should enable the Serial Data Output (SDO) pin were sent. This command was followed by a command to read a register on this SDO pin. No data output was observed. Next, a command to change the Frequency Tuning Word (FTW), which should change the RF output frequency, was sent. After execution of this command, the chip's IO_UPDATE pin was manually triggered, which should have caused the output to change frequency. No such change was observed. It was speculated that a cause could be due to the long time (order of seconds) between commands sent to the AD9912; this was partial motivation for the testing described by Section 2.1.2.

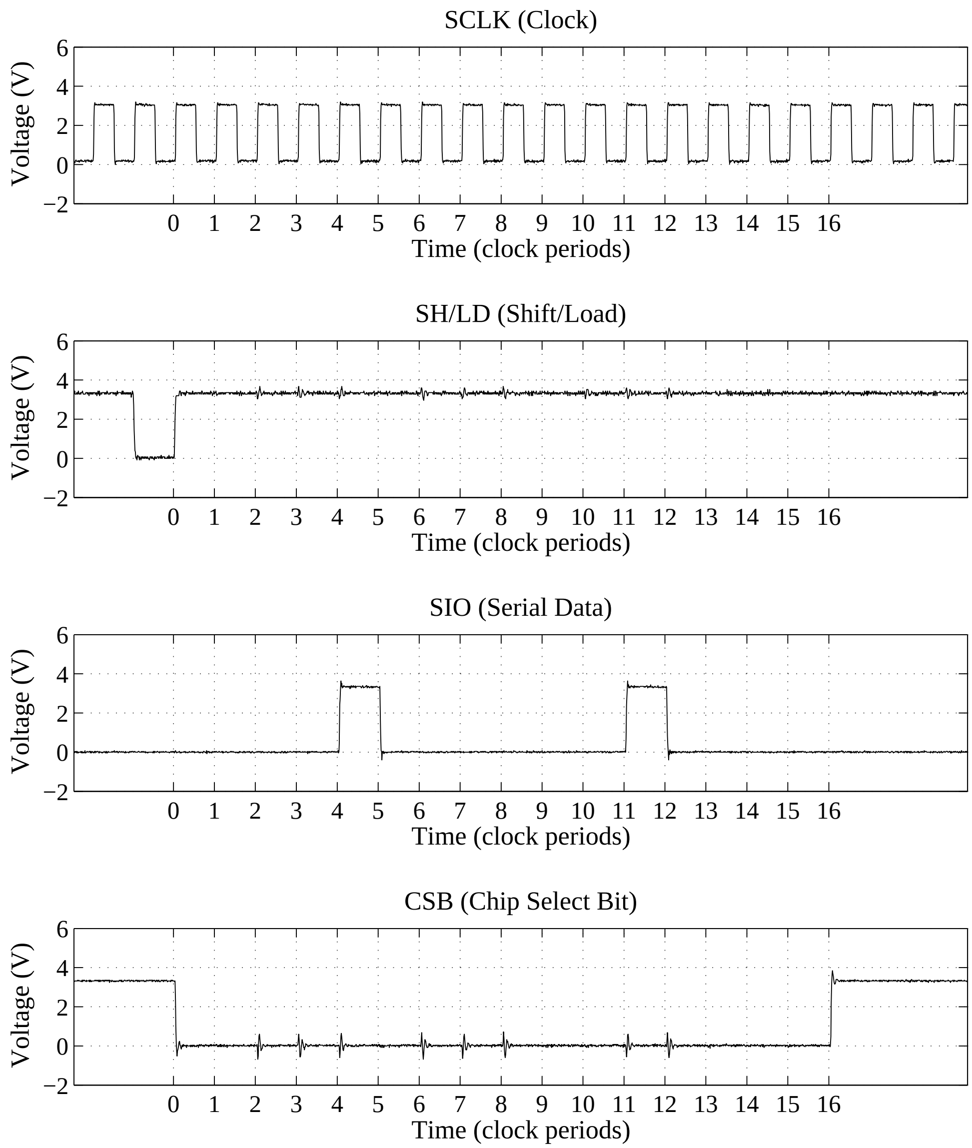

Figure 2.1: 16-bit data transfer (SZ=0). Data on SIO is $0000\ 1000\ 0001\ 0000_2$. Parallel input signals were generated with a breadboard setup and an external SCLK was supplied at 1MHz. The top plot shows the serial clock, which is used to time the rest of the signals. Below SCLK, the SH/LD plot shows the signal that loads the SIO data into the parallel-to-serial converter. Once SH/LD returns high, data is clocked onto the SIO pin, shown second from the bottom. This data consists of one bit per clock cycle, as shown with dotted vertical lines. The bottom plot shows the CSB, which is held low during the entire 16-bit data transfer. Note that this figure uses the CSB data from Figure 2.2 and, as such, the noise on CSB caused by changes in the SIO line do not line up.

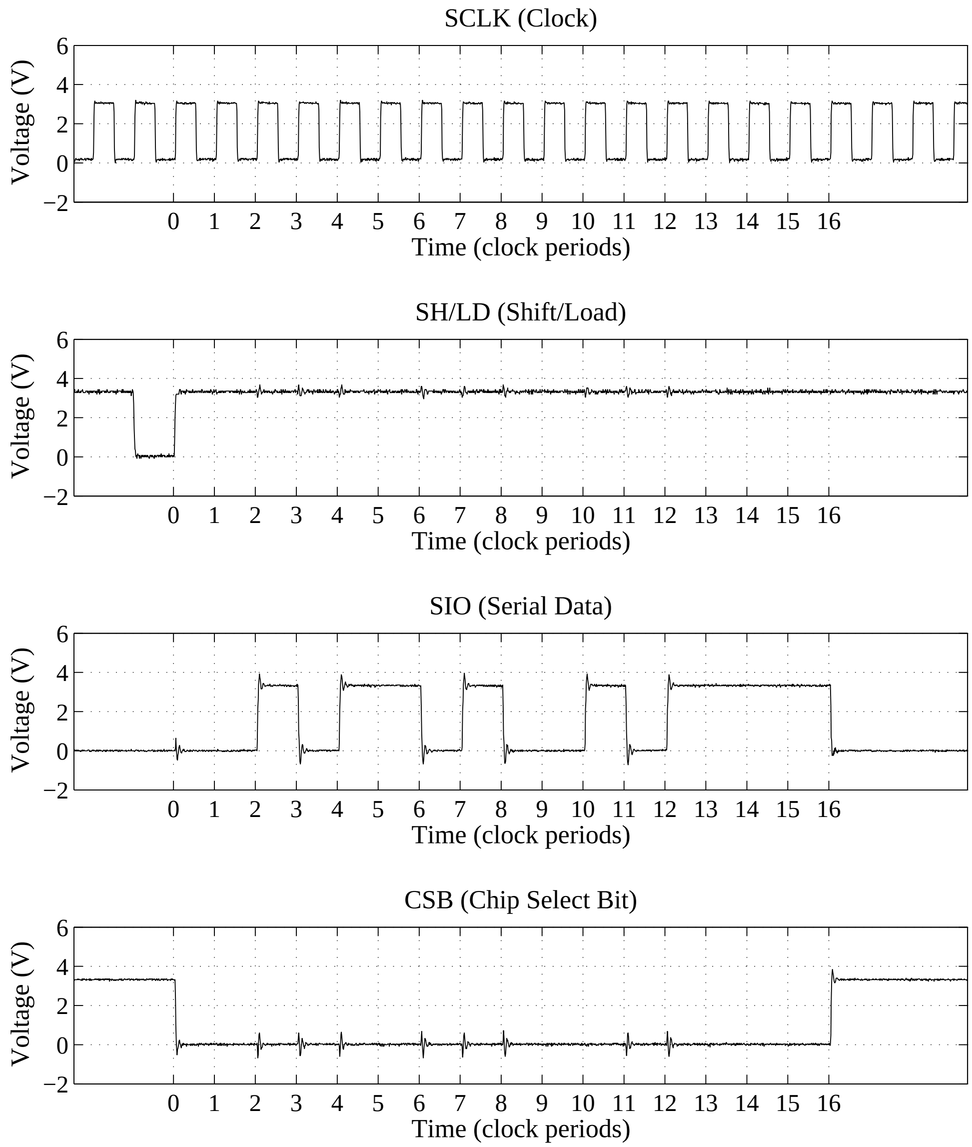

Figure 2.2: 16-bit data transfer (SZ=0). Data on SIO is $0010\ 1101\ 0010\ 1111_2$. Parallel input signals were generated with a breadboard setup and an external SCLK was supplied at 1MHz. The top plot shows the serial clock, which is used to time the rest of the signals. Below SCLK, the SH/LD plot shows the signal that loads the SIO data into the parallel-to-serial converter. Once SH/LD returns high, data is clocked onto the SIO pin, shown second from the bottom. This data consists of one bit per clock cycle, as shown with dotted vertical lines. The bottom plot shows the CSB, which is held low during the entire 16-bit data transfer.

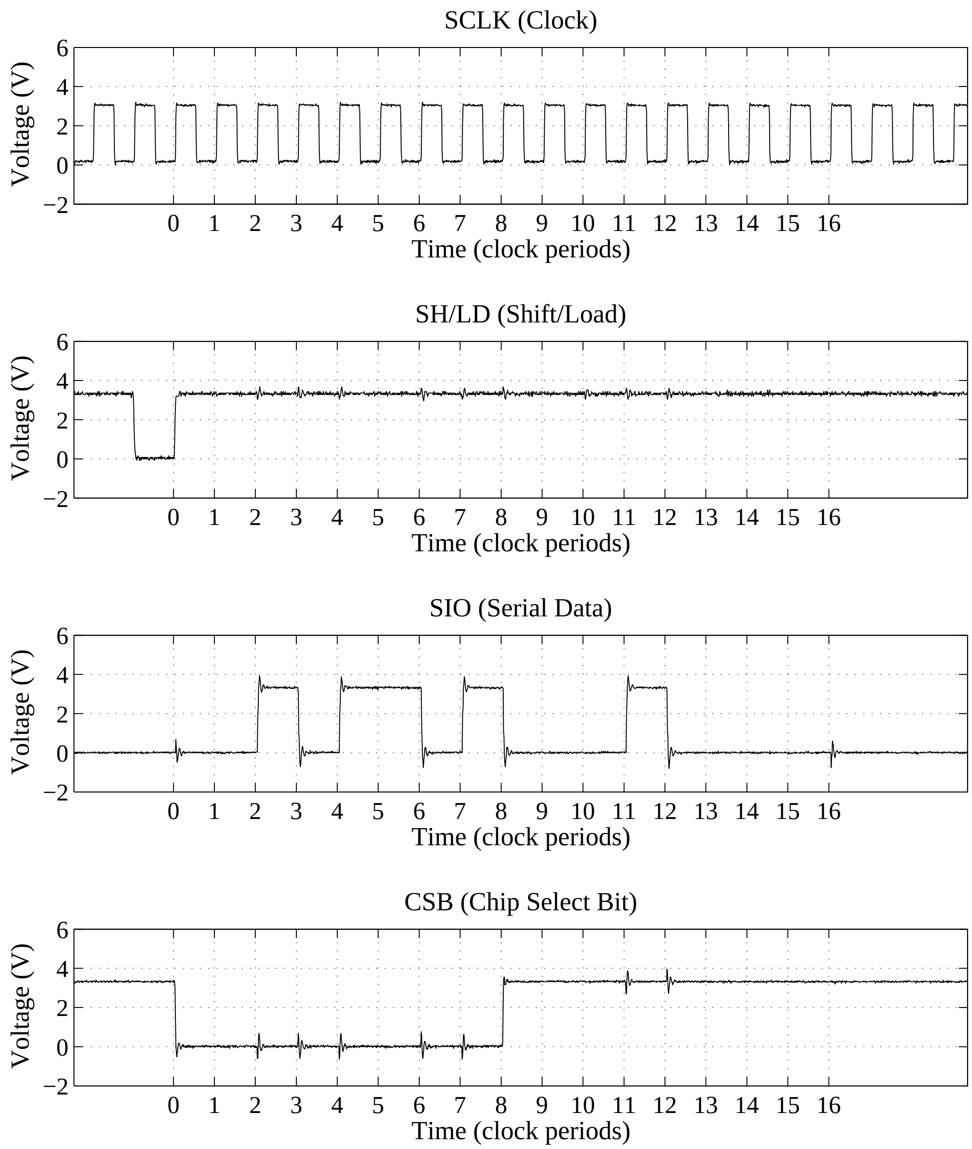

Figure 2.3: 8-bit data transfer (SZ=1). Data on SIO is 0010 1101_2. Parallel input signals were generated with a breadboard setup and an external SCLK was supplied at 1MHz. The top plot shows the serial clock, which is used to time the rest of the signals. Below SCLK, the SH/LD plot shows the signal that loads the SIO data into the parallel-to-serial converter. Once SH/LD returns high, data is clocked onto the SIO pin, shown second from the bottom. This data consists of one bit per clock cycle, as shown with dotted vertical lines. The bottom plot shows the CSB, which is held low during the 8-bit data transfer; CSB returns high for the final 8 bits of data.

2.1.2 NI-DAQ Control of the UTBus

To decrease the time between commands, the DDS was hooked up to the QDG laboratory's existing UTBus installation. This was used to confirm expected operation of the DDS device's digital interface and to attempt to program the DDS. Although the data and address bits were supplied from the lab system, function generators were used to drive the SCLK and SYSCLK at 5MHz and 25Mhz, respectively. As the PLL multiplier is set to 40x, this SYSCLK value provided a 1GHz internal clock to the AD9912.

This round of tests was performed with the address comparator placed. Figures 2.4 and 2.5 show the results. Note that the SCLK frequency has increased to 5MHz. This higher clock speed was used to allow the board to communicate with the existing lab setup. That is, the board was hooked up to the UTBus it was designed to support. This was accomplished by connecting the DDS device alongside previous generation DDS boards in a real lab setup. Through investigation of the software used to control the UTBus, a python program was created to write a file with the data to send to our DDS board. The python program may be seen in Appendix A. This data was passed to a NI-DAQ driver program, which takes byte-level data commands and outputs them on the UTBus.

Therefore these figures demonstrate that the parallel-to-serial converter supports the lab's UTBus setup. To further demonstrate functionality, the strobe bit was included in Figures 2.4 and 2.5. The UTBus provides this strobe bit, which, if enabled, triggers the parallel-to-serial converter. Note that the strobe is not a nice square signal and instead has some ringing and a slow upwards slope. This is due to the high capacitance in the lines used by the QDG lab.In particular, many of the UTBus lines are upwards of a meter in length.

To confirm the internal workings and timings of the parallel-to-serial converter, signals used internally were checked for timing and functionality. These are shown in Figure 2.6. The top plot of this figure shows the serial clock, which is used to time the rest of the signals and is generated from an off-board function generator. Below SCLK, the address comparator output, from the byte comparator in the digital interface, shows the time during which the address pins of the UTBus match the DDS device address. This comparator output must be low in order to enable the parallel-to-serial converter. If this comparator output is low, the UTBus strobe bit will then trigger a signal pulse on the board that loads the digital data from the UTBus into the digital interface. This signal is shown second from the bottom as the SH/LD bit. Finally the CSB, from 2 of the shift registers, should be held low for 16 clock cycles immediately after the SH/LD bit returns high; this enables the serial input of the AD9912 while data is clocked in at the rate of one bit per clock period.

As expected, the address comparator outputs a low signal, as the address on the UTBus matches, some time before and after the strobe bit goes high. Once the strobe bit goes high, the parallel-to-serial converter is triggered and, after two clock cycles (required by the circuit), the SH/$\overline{LD}$ bit goes low for exactly one clock period. Finally, as expected, the CSB is held low for 16 clock cycles.

The full process of a parallel-to-serial conversion is defined as the time from the first rising edge of SCLK while the strobe bit is high until the CSB goes high. In Figure 2.6, this process begins at 1.6μs and ends at 3.4μs for a total time of 1.8μs at a SCLK frequency of 5MHz.

Although the UTBus sent data to the AD9912 with a reasonable time between commands, programming of the AD9912 was still unsuccessful. In particular, attempts to write to a register that would change the output of the AD9912 failed; these expected changes were not observed. For a discussion of why this might be, see Section 3.

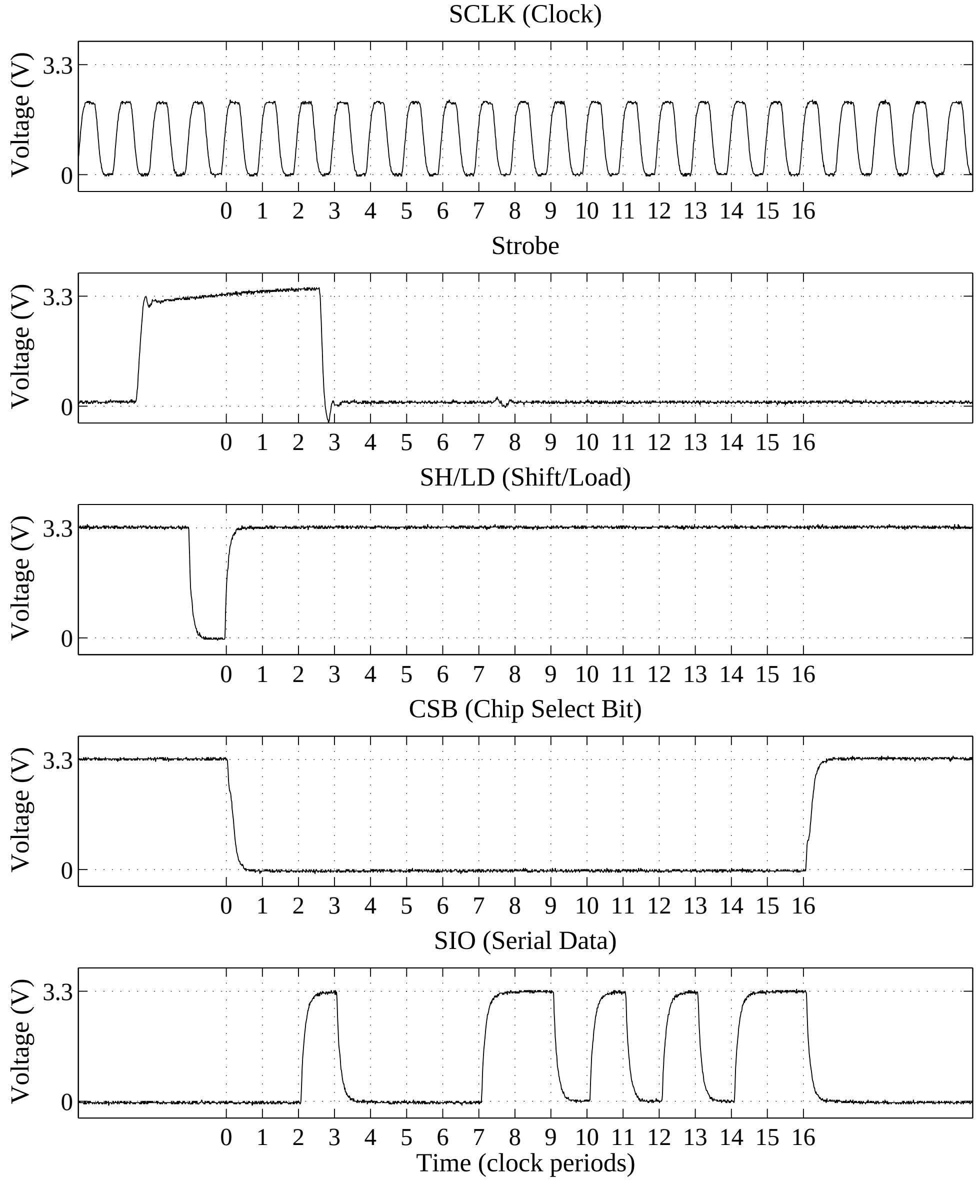

Figure 2.4: 16-bit data transfer (SZ=0). Data on SIO is 0010 0001 1010 1011_2. Parallel input signals were generated from the QDG lab's UTBus system and an external SCLK was supplied at 5MHz. The top plot shows the serial clock, which is used to time the rest of the signals. Below SCLK, the strobe demonstrates the timing of the UTBus strobe pin, which triggers the parallel-to-serial conversion. The SH/LD plot shows the signal that loads the SIO data into the parallel-to-serial converter. Once SH/LD returns high, data is clocked onto the SIO pin, shown second from the bottom. This data consists of one bit per clock cycle, as shown with dotted vertical lines. The bottom plot shows the CSB, which is held low during the entire 16-bit data transfer.

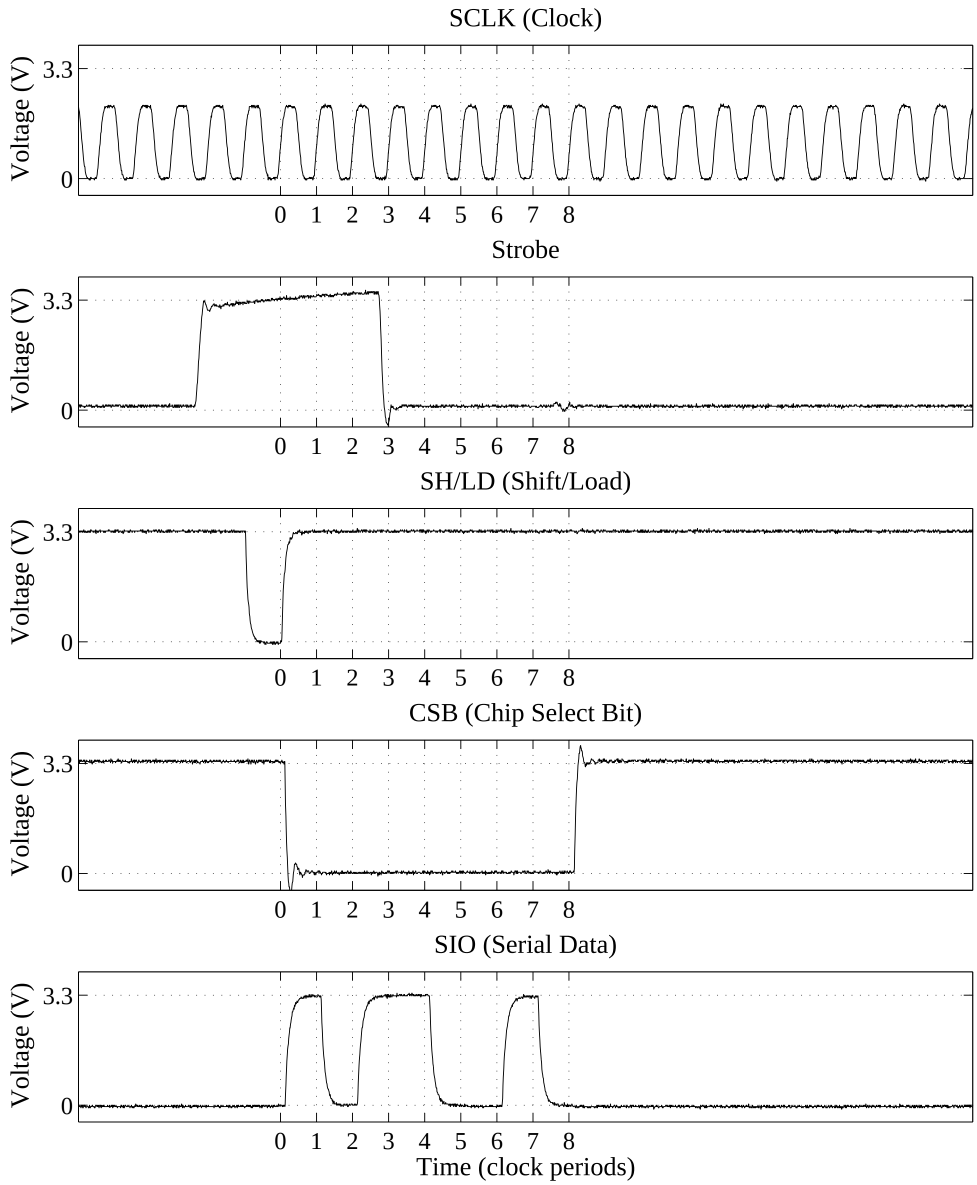

Figure 2.5: 8-bit data transfer (SZ=1). Data on SIO is 1011 0010_2. Parallel input signals were generated from the QDG lab's UTBus system and an external SCLK was supplied at 5MHz. The top plot shows the serial clock, which is used to time the rest of the signals. Below SCLK, the SH/LD plot shows the signal that loads the SIO data into the parallel-to-serial converter. Once SH/LD returns high, data is clocked onto the SIO pin, shown second from the bottom. This data consists of one bit per clock cycle, as shown with dotted vertical lines. The bottom plot shows the CSB, which is held low during the 8-bit data transfer; CSB returns high for the final 8 bits of data.

.

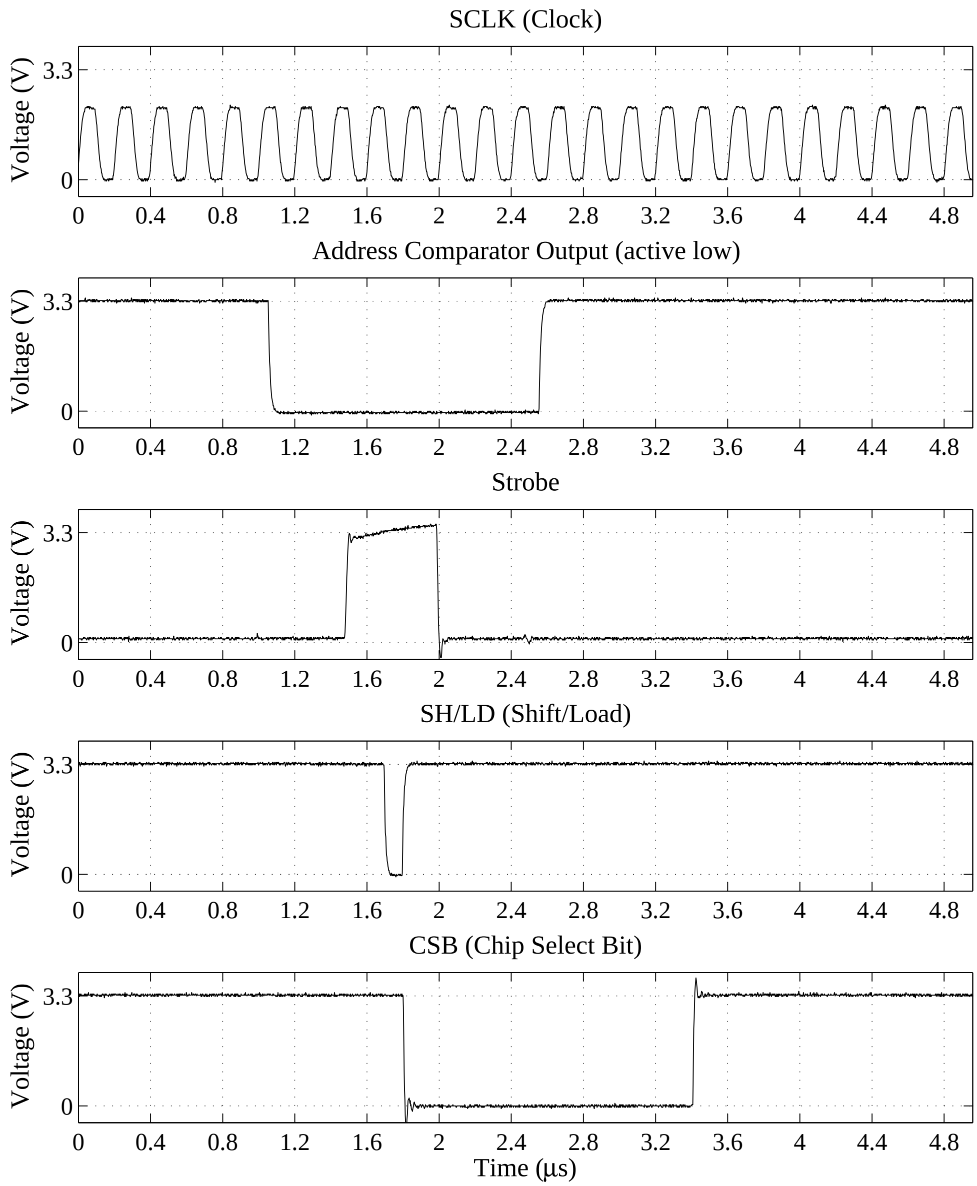

Figure 2.6: Timing of the Internal Parallel-to-Serial Converter Signals. Parallel input signals were generated from the QDG lab's UTBus system and an external SCLK was supplied at 5MHz. The top plot shows the serial clock, which is used to time the rest of the signals. Below SCLK, the address comparator output is an indicator of the time during which the address pins of the UTBus match the DDS device address. The strobe bit tells the DDS device that the serial data is ready to be read and starts the conversion. Second from the bottom, if the UTBus address matches the board address, the SH/LD bit is low for one clock cycle immediately after the strobe bit goes high. Finally, the CSB is held low for 16 clock cycles immediately after the SH/LD bit returns high.

2.2 RF Output

As it was not possible to program arbitrary frequencies, testing was performed only on the preset frequency outputs of the AD9912. These presets are chosen with the use of startup pins on the AD9912. This allowed seven discrete output frequencies, ranging from 38.9MHz to 155.5MHz, on the DDS board's RF output. The results are shown in Figure 2.7.

From the AD9912 datasheet, the possible startup frequencies are: 38.9MHz, 51.8MHz, 61.4MHz, 77.8MHz, 92.1MHz, 122.9MHz and 155.5MHz. By toggling the startup pins, each of these modes was selected and the output was monitored with a 1GS/s oscilloscope. With a AD9912 SYSCLK of 1GHz (clock input from a function generator at 25MHz and PLL multiplier of 40), these frequencies were successfully viewed on the DDS board RF output, post-filter, and are shown in Figure 2.7.

That changing frequencies were visible on the RF output verified the correct functionality of a number of parts on the board. For the DDS to output a signal at all, power regulators and supply for the DDS must be working. To output at the correct frequency, the DDS has to have a working SYSCLK and a functional PLL circuit for the correct multiplier. For all the signals to be connected and correct, the PCB fabrication would have to be correct. Finally, the transformer and reconstruction filter on the DAC output would have to be operational and working as designed. That the expected frequencies of the RF output were observed, all of these areas of the board have been confirmed as correct.

These RF outputs correspond to the frequencies expected but also change in amplitude. As can be seen in Figure 2.7, the signal amplitude decreases with increasing frequency. The cause of this has not been verified. However, it should be noted that the optional resistors R1 and R2 were not placed, which means that the RF output was not driving a significant load[1]. Due to their placement on the differential DDS DAC output pins, however, it is possible that these resistors will stabilize the output. Jumper resistor W8 was also not placed.

[1]The oscilloscope load is on the order of 1MOhm

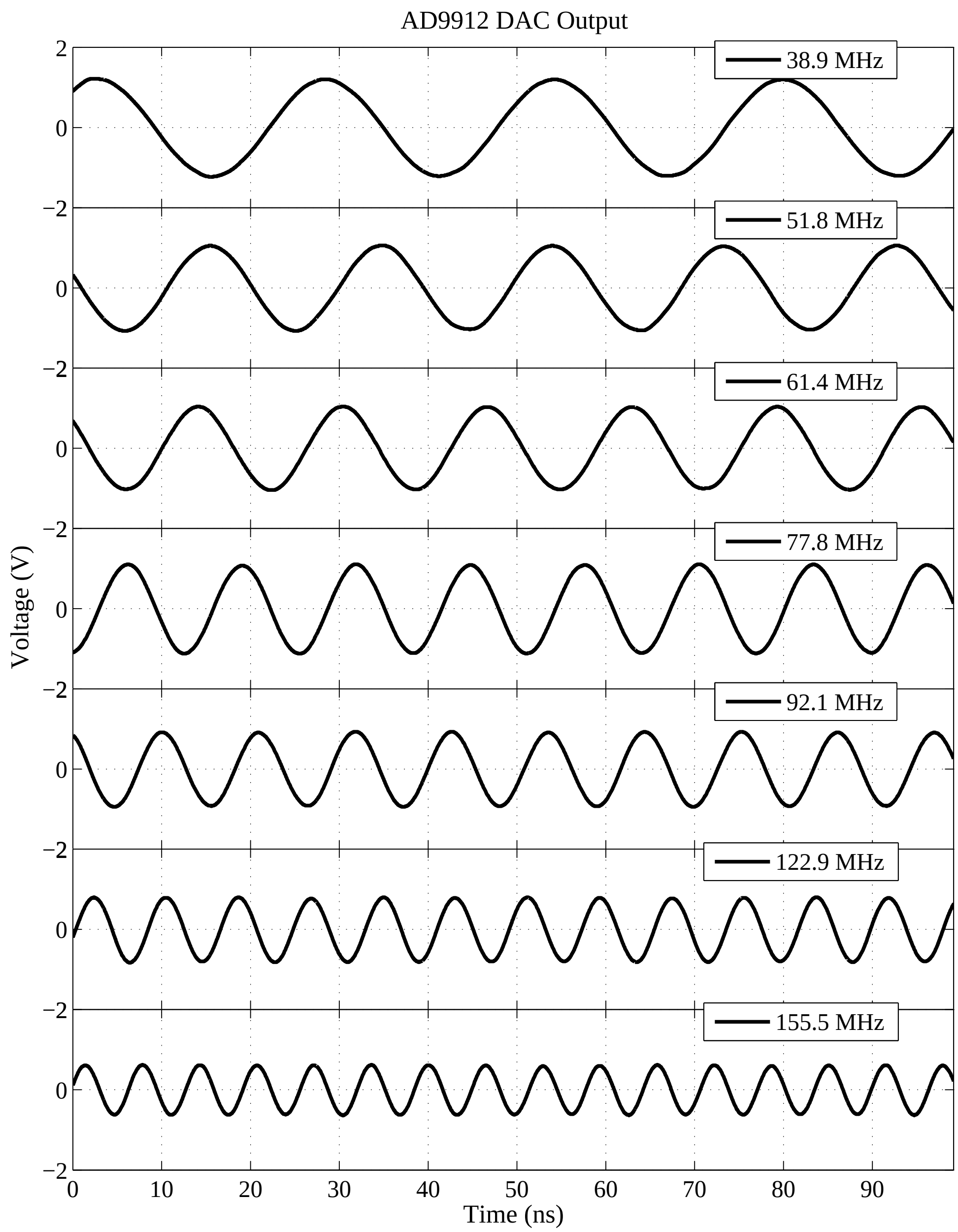

Figure 2.7: Filtered DAC output. Measured with a 60MHz, 1GS/s oscilloscope. The frequency levels were set with the AD9912 startup pins and are preset to one of 7 frequencies as defined in the AD9912 datasheet.

3

Conclusions

Three main areas were tested: the digital interface, programming the AD9912 and the RF output. For the basic functionality tests, both the digital interface and RF output performed as designed. However, programming the device was not achieved.

The digital interface was found to work as expected. Although frequency limits were not tested, it was confirmed that the parallel-to-serial conversion process is performing as designed when used with the QDG lab's UTBus system. Despite this, attempts to program the AD9912 were not successful. Determining the cause of this failure will need to be investigated in the future.

Despite a lack of software control of the AD9912, RF output from the DDS was successfully generated. Frequencies ranging from 38.9MHz to 155.5MHz were tested, with only a slight attenuation at higher frequencies. That this range of output was successfully generated and observed confirms that, once AD9912 programming is successful, the AD9912 DDS device will accomplish the stated project requirements.

4

Recommendations

This section describes the recommendations for future work on the AD9912 DDS board. Suggestions for future testing, board population for a production run, board installation in the lab and device programming are discussed.

4.1 Further Testing

It is desirable to determine the upper (stable) rate of the SCLK, as this allows for a faster UTBus frequency. This can be performed by using the external SCLK input and a function generator, then increasing the SCLK frequency until a maximum is found where the parallel-to-serial converter no longer works.

The CMOS output should also be tested. This can be accomplished by programming the AD9912 to use this mode and enabling the feedback jumper W8.

The reconstruction filter can be tested by using BNC connector J2 as a filter input. See Section 2.4.3.6 of Part I of this report for information on how to enable this mode. This setup would allow the use of a spectrum analyser to determine the transfer function. A similar setup can be used to determine the magnitude of high frequency spurs from the DDS DAC output at high frequencies.

Finally, performance testing of these high performance devices should be performed. In particular, the noise levels, Q-factor of the signal and accuracy of the frequency should be investigated. Additionally, it should be confirmed that the power output of the DDS board is always above the -17dBm required by the pre-amplifiers used in the QDG laboratory.

4.2 AD9912 Programming

The ability to control the AD9912 chipset is a crucial part of the DDS board. Prototype testing was unable to find any evidence of successful writing to an AD9912 register. Although there is no conclusive evidence regarding the reason why programming was unsuccessful, there are a few educated guesses. To prove or disprove these possibilities, there are many different techniques available to debug the serial control port, ranging from analysis of the evaluation board to directly controlling the serial ports.

There are two educated guesses of why the AD9912 chip does not appear to receive register write commands. Firstly, it is possible that the AD9912 chip used on the prototype board is broken; the group has heard of a similar issue with an Analog Devices DDS chip. In this case, switching the chip would be a quick fix. However, the chips are not cheap and the procedure required to remove and replace the DDS chip is both difficult and includes a possibility of breaking other components on the board. For this reason, it is suggested that this hypothesis be proved using other methods before attempting a chip replacement.

A second, and perhaps more likely, possibility for why programming has not worked is that the timing of the digital interface circuitry does not meet the timing requirements of the AD9912's serial control port. The AD9912's datasheet is a useful reference here.

Perhaps the most revealing analysis technique will be to set up the AD9912 evaluation board, which was purchased for this type of testing. The evaluation board allows for USB control of the AD9912. It would be particularly instructive to monitor input signals to the serial control port of the AD9912 on the evaluation board and compare with those produced by our digital interface circuitry. It might also be possible to connect the serial control signals from the evaluation board directly to those on our prototype board. The two boards might then be started in the same state and programmed at the same time using the the evaluation board's USB interface circuitry and software.

Another possible analysis technique is to directly control the serial ports without use of the digital interface. This is perhaps easiest with an Arduino or similar microcontroller, which could be programmed to directly control the serial port.

If either the evaluation board or an external microchip board is used to generate an off-board serial control, the external signal input will need to be connected directly to the SCLK, CSB and SIO input pins of the AD9912. This connectivity may easily be obtained by soldering wires to a grouping of vias near the AD9912. However, as the digital interface chips are always powered on when the entire DDS board has an external power source, as required to power the AD9912, an external input will cause fighting on the serial input pins. As no jumpers were placed to disconnect the serial pins from the digital interface, it is necessary to either remove the shift registers or manually cut the traces. Both are viable options, although cutting the traces is likely both faster to perform and easier to repair.

4.3 DDS Board Programming

The AD9912 uses a different programming scheme and register addresses than the previous generation board. To compensate for this, the QDG lab will need to modify the existing UTBus control programs to facilitate programming of the AD9912 registers. Detailed information on what is required to program the AD9912 using the DDS board can be found in Section 2.1.6 of Part I of this report. A low-level python program was used to program the DDS device during testing; this code is in Appendix A.

4.4 Population

Although one board has been built and assembled, the QDG lab has requested at least eight fully assembled devices. Parts and PCBs are available for at least ten. Once additional testing has been completed, it will be necessary to populate the remainder of the boards. While the PHAS electronics shop was willing to populate one board, they requested that volumes higher than five be performed off site.

No research has been done in determining a suitable company for this work. It is recommended that the PHAS electronics shop be contacted for suggestions.

4.5 Installation

To be used in the lab, the DDS board will need to be rack mounted. This is accomplished with a metal rack-mounting front panel, to which the DDS enclosures are screwed, a platform and a 5V power supply. The front panel and power supply have been acquired but the parts will need to be assembled.

Appendix A

UTBus Python Code

```
## This code uses existing libraries written by the QDG lab
## to convert a series of data commands into a binary
## series of commands. These commands are read by a NI DAQ
## driver program, which is executed using the following command:
##    C:\UTBUS\V1\BusDriver\bin\Release\bus_driver_v1.exe 1 20 0\
##        C:\Documents and Settings\QDG Admin\Desktop\AD9912.utb
## This command reads the AD9912.utb binary file, written by this
## python program, and outputs the commands (set in the list "datas")
## to the UTBus
##
## Author: Brendan Mulholland (firemulholland@gmail.com)

import os.path
import struct

def _get_bytecode_info():
    bytecode_info_fname = os.path.join("C:\\UTBUS\\V1\\UTBus1\\bytecode.info")
    NS = {}
    execfile(bytecode_info_fname,NS)
    return NS["bytecode_map"],NS["bytecode_signature"],NS["max_comment_size"]

(bytecode_command,bytecode_signature,max_comment_size) = _get_bytecode_info()

sig_size = len(bytecode_signature)
bcode = struct.pack("=B",sig_size)
bcode += struct.pack("=%dB" % sig_size,*(ord(c) for c in bytecode_signature))

# Test command
# datas = [[0, "0101010101010101"]]

# Read a register from SDO pin
# datas = [[0, '0000000000000000'], # Write (0) 1 byte (00) to register 0x0000
# [1, '0000000010000001'], # Byte to write = 10000001
# [0, '0000000000000001'], # Read (1) 1 byte (00) from register 0x0000
# [1, '0000000000000000'] # Pull CSB low for 8 bits while the DDS outputs
# ]

# Set FTW
datas = [[0, '1101010110000100']
```

```
# , [0, '0110000000000001']
# , [0, '1010000000000000']
, [0, '0001000001001101']
]

for SZ, data in datas:
    address = int('01110100',2)+(2*SZ)
    bcode += struct.pack("=BBH", bytecode_command["COMMAND"], address, int(data,
         2))
    bcode += struct.pack("=BQ", bytecode_command["WAIT"], 10)

bcode += struct.pack("=B",bytecode_command["STOP"])

f = open("C:\\Documents and Settings\\QDG Admin\\Desktop\\AD9912.utb", "wb")
f.write(bcode)
f.close()
```