AD9910 Log

1. Evaluation Board Test

1.1. Verilog Code Modification

1.2. Python Code

1.3. Implementation

2. New Design

2.1. Modified Architecture

2.2. Modified Function

2.3. Simulation on Verilog

2.4. Test on AD9910 evaluation board

1. Evaluation Board Test

1.1. Verilog Code Modification

To test AD9910 evaluation board, AD9912 verilog code was modified (this code is located in ''). AD9912's block diagram is shown in below figure.

**Figs**

First modification in code is that 'DDS1\_powerdown', 'DDS2\_powerdown' logics in 'AD9912\_DAC8734.sv' were replaced to 'DDS1\_ioupdate', 'DDS2\_ioupdate' logics which would make 60ns(6 clock cycle) pulse of IO UPDATE. 'DDS1\_powerdown', 'DDS2\_powerdown' were discharging only LOW in AD9912 code. Second FSM state 'MAIN\_DDS\_IOUPDATE\_OUT', 'MAIN\_DDS\_IOUPDATE\_END' were added in same file. Third, parameter 'DDS\_MAX\_LENGTH' in same file and 'MAX\_LENGTH' in 'WriteToRegister.v' was changed to 9 from 8 due to profile setting which requires 9bytes MOSI. And some code were added in 'data\_receiver\_v1\_00.v' to initialize logics and replaced "\r", "\n" to ascii code value in 'TERMINATOR\_CURRENT\_CHAR\_POINTER' conditional part.

1.2. Python Code

Python code 'AD9910\_eval.py' was made to control AD9910 evaluation board. Using 'Arty\_S7\_v1\_01' library is same with AD9912 python code('AD9912\_DAC8734\_v1\_01.py') and serial communication protocol is similar between AD9910 and AD9912. However AD9910 uses IO\_UPDATE pulse to update register meanwhile AD9912 writing to specific register, and AD9910 has fixed length in each register and longest one is 9bytes including address since AD9912 has mutable length and longest one is 8bytes.

\* \_\_init\_\_( fpga, min\_freq = 10, max\_freq = 400, sys\_clk = 1000, unit = 'MHz')

AD9910 constructor method has fpga, min\_freq, max\_freq, sys\_clk, unit parameter. fpga is ArtyS7 object, min\_freq, max\_freq are minimum, maximum output frequency and sys\_clk is reference clock for DDS(PLL is not implemented yet). Unit of frequency is set by unit parameter and default value is 'MHz'.

\* make\_header\_string(register\_address, direction = 'W')

This method makes header string in hexadecimal which is compatible with AD9912's method. But in AD9910 this was not used, and header was made directly in write16, write 32, write64 method.

\* write16(register\_addr, register\_data, ch1, ch2)

This method write 16 bit data to DDS register calling 'send\_mode\_BTF\_int\_list', and 'send\_command' method in ArtyS7. register\_addr is address of register, register\_data is register data, and ch1, ch2 is whether channel1, channel2 will be written(Now only channel1 is supported).

\* write32(register\_addr, register\_data, ch1, ch2)

This method is similar to write16, but write 32 bits to register.

\* write64(register\_addr, register\_data, ch1, ch2)

This method is similar to write16, but write 64 bits to register.

\* frequency\_to\_FTW(freq, unit = 'MHz')

This method convert frequency to 32 bits FTW which is respond to output frequency of AD9910. Unit of frequency is equal to parameter unit and default is 'MHz'.

\* phase\_to\_POW(phase, unit = 'FRAC')

This method converts phase to 16 bits POW which is respond to phase of output wave. Unit of phase is equal to parameter unit and default is 'FRAC'.

\* amplitude\_to\_ASF(amplitude\_frac)

This method converts fraction between output amplitude and maximum amplitude to 14 bits ASF. There is no unit parameter and only support fraction.

\* set\_frequency(freq, ch1, ch2, unit = 'MHz')

This method sets value in FTW register. Unit of frequency is equal to parameter unit and default is 'MHz', and ch1, ch2 is whether channel1, channel2 will be written(Now only channel1 is supported). Note that this method does not change single tone profile register. This register will be used in RAM mode, DRG mode, parallel mode.

\* set\_phase(phase, ch1, ch2, unit = 'FRAC')

This method sets value in POW register. Unit of phase is equal to parameter unit and default is 'FRAC', and ch1, ch2 is whether channel1, channel2 will be written(Now only channel1 is supported). Note that this method does not change single tone profile register. This register will be used in RAM mode, DRG mode, parallel mode.

\* set\_amplitude(amplitude\_frac, ch1, ch2)

This method sets value in ASF register. ch1, ch2 is whether channel1, channel2 will be written(Now only channel1 is supported). Note that this method does not change single tone profile register. This register will be used in RAM mode, DRG mode, parallel mode.

\* initialize()

This method initialize CFR to use singletone mode using set\_CFR1(), set\_CFR2(), set\_CFR3() function.

\* set\_CFR(...)

\* set\_profile(freq, phase, amplitude, ch1, ch2, profile = 0, unit\_freq = 'MHz', unit\_phase = 'FRAC')

This method sets frequency, phase, and amplitude of profile register.

1.3. Implementation

1.3.1. Debugging Output of FPGA board

Bit stream file was uploaded to ArtyS7 board, and output of FPGA was checked by MSO7054 oscilloscope. ja7, ja6, ja5, ja3, and GND pin of ArtyS7 was connected to RIGOL cable and this cable was connected to oscilloscope which is well impedance matched. To measure output signal of FPGA using oscilloscope, turn on LA port. And you can see output signal of FPGA on oscilloscope screen. Then, set oscilloscope to decode mode and set SPI mode and source of CSB, MOSI, MISO, SCK properly. Then set horizontal time interval. In this case, 1~5us would be sufficient. Finally put SINGLE button and send signal to FPGA using computer. Then you can see output of FPGA on the oscilloscope screen.

1.3.2. Check Output of DDS

1.3.2.1. Using Analog Program

Running DDS board requires signal generator(SG384), DC supply(2230G-30-6). And control of DDS needs connection of computer between board using USB cable, and proper jumper setting on the board. Also to check output of DDS, oscilloscope is needed(TDS3032B). First set frequency of reference clock to 500MHz, and output power bigger than 10.0dBm. Maximum input reference clock of AD9910 is 1GHz, but in manual mode it is limited to 500MHz. Then connect output of signal generator to DDS board using SMA to SMA cable. Then set DC supply output to 1.8V, and 3.3V each. Then connect each output and ground to evaluation board. Connecting each port separately, or using ferrite bead if more precise measurement, but purpose of this test is just checking functionality of DDS so it was not considered. Then set jumper W1, W2, W4 on the evaluation board to enable, and place jumper at the W5, W6, W3. And set W7 to REF\_CLK(since signal generator is used for reference clock) and W11 pin to middle(since TXE pin was not used). And connect DDS board to PC, then you can see VBUS led being flahsed in red, and USB led being blinking in yellow if you connected, or set jumpers well. Then turn on AD9910 program on the computer. If program is not installed, you can search in Google easily. After program is turned on you can see two green bar left below of program and you have to set External Clock in the program. In this case you can set to 500MHz Then connect FILTERED\_IOUT to oscilloscope using SMA to BNC cable. To see simple waveform set profile0 output frequency to 200MHz, phase offset to 0 DEG and put LOAD button. Notice that Enable Multiplier, Power Down should not be set to high and if you did not set IO UPDATE option to Auto, you have to send IO UPDATE signal manually. Then, you can see output of DDS on the oscilloscope screen.

1.3.2.2. Using FPGA(ArtyS7)

After output of FPGA is proper, then connect FPGA pins to AD9910 evaluation board using jumper cable. Same signal generator, and DC supply setting are used except 1GHz output frequency of signal generator, but W1, W2, W4 jumper should be set to DISABLE, and remove W5, W6, W3 jumpers on the board. W7 should be set to REF\_CLK(since signal generator is used for reference clock) and W11 pin to middle(since TXE pin was not used). You should connect 4 pins(IO\_UPDATE, CSB, SCLK, SDIO) to FPGA(in this case JA7, JA6, JA5, JA3). And other pins in U5(IO\_RESET, SDO, DRHOLD, DR\_CTL, DR\_OVR, OSK, SYNC) should be connected to ground. In this test only single tone mode was implemented, so profile pins in P1 should be connected to ground. And EXT\_PWR\_DWN, and RESET pins should be connected to ground. And connect oscilloscope and DDS board using SMA to BNC cable. To see simple output of DDS, turn on AD9910\_EX.py and press 6, 5, 7, 5, 8, 5, 4, output frequency of DDS, output phase of DDS, output amplitude of DDS, ch1 to 1, ch2 to 1, profile to 0. Then you can see output of DDS on the oscilloscope. Notice that pressing 6, 5, 7, 5, 8, 5 is corresponding to send CFR1, CFR2, CFR3 register to DDS board and sending IO UPDATE signal to update register of DDS. If you do not implement CFR setting, you will see half of frequency that you set due to bypassing option of CFR. And pressing 4 and setting output properties is corresponding to setting register of profile 0. If you set profile 1 in this test, you cannot see output on the oscilloscope due to fixed pin setting on the board. And setting frequency, phase, and amplitude using [1], [2], [3] function will not work properly because these does set FTW, POW, ASF register which are not profile register and supply each parameter when using RAM, DRG, Parallel mode.

2. New Design

2.1. Modified Architecture

New hardware architecture was made applying new design of sequencer and maintaining previous AD9912 FSM. Main differences of this design are FIFO which discharge instruction at stated timestamp, gpo core and rto core which are based on new design of sequencer, and removal of DAC modules and WRT modules. DAC modules and pins were removed since DAC pins were replaced to AD9910 parallel output. jb. jc pins were also activated to supply signal of profile and rest of parallel output. WRT module was replaced to new spi module which can set CPHA, CPOL, CSPOL, slave enable, length of signal, cs, whether to end SPI, LSB\_FIRST, and clock divide. So any SPI mode can be implemented. And, depending on slave enable DDS register can be readed. 32 bit, 64 bit wirte, read function is implemented on AD9910.py. Additionally, override function is implemented so driving FPGA without FIFO is possible. Notice that when python code is modified, notice that WRT is removed and length of SPI data is set in SPI configuration, so length of data is not included to BTF\_list(so 64 bits needs list of length 8, and 128 bits needs list of length ). Moreover, BTF\_list should be composed in little endian. For example, 0x0123456789ABCDEF should be converted to [EF, CD, AB, 89, 67, 45, 23, 01].

There are some difference with new sequencer. First sequencer is implemented based on memory mapped IO, while this is not. However, memory mapped IO can be easily implemented adding some register outside of gpo\_core, rto\_core, rti\_core. In addition, sequencer use 32 bit data transfer while this uses 128 bit and 64 bit. Second, there are error signals which discharge pulse and error dat. Error signals are overflow error, timestamp error, overrided, busy error, underflow error. Overflow error indicate that fifo in rto\_core or rti\_core is in overflow. And error data corresponds to data which is not written to fifo due to full of fifo. Timestamp error indicates that timestamp of data in the top of fifo is less than counter. Error data corresponds to this top data. Overrided indicates that override value and fifo output came from input coincidently. Error data corresponds to fifo data, and override value would be discharged from gpo\_core. Busy error indicates that new income arrived while output port is in busy state. In this case error data corresponds to new income, and output of gpo\_core would not be changed and selected pulse would not be made.

2.2. Modified Function

Python code was modified due to new design of hardware. Previous python code use fixed length (9 times 8 bits) when using send\_mod\_BTF\_list, however new python code use 8 bytes in override mode, and 16 bytes in auto mode. These mode can be set using function ‘auto\_mode()’ and ‘override\_enable()’ each. In the same way each can be disabled using ‘auto\_mode\_disable()’ and ‘override\_disable()’. In auto\_mode, timestamp will be added to instruction according to time value in AD9910 object. Time value can be shifted using ‘delay\_cycle(shift\_time)’ and ‘delay(shift\_time)’ function. Notice that ‘delay\_cycle’ function makes shift of unit in machine unit while ‘delay’ function makes shift of unit in second. For example, delay\_cycle(10) is equal to delay(100\*ns) since one cycle is equal to 10ns in ArtyS7. And, using function of AD9910 would not change time value, so you have to consider set proper time delay not to make busy error or timestamp error.

Previous AD9910 code uses parameter unit to indicate which unit is used in frequency, phase etc. However in new code, you don’t have to consider about this, and write proper unit and asterisk(‘\*’) next to number like in ARTIQ. For example, 100 MHz can be written as ‘ 100 \* MHz’. Available unit is written in unit\_set.py so importing unit\_set.py is necessary.

2.3. Simulation on Verilog

In Arty\_S7\_v1\_01.py, there is TEST variable which indicate whether program is used for simulation not implementing serial communication or real experiment. If TEST is set to 1, value which would be discharged from PC would be printed on screen, so simulation using Verilog module is possible. Notice that UartRx is set to high before any data sent, and become low for 17361 cycles (according to baudrate set to FPGA) and data sent to FPGA. After 8 bits of data is sent UartRx should be set to high for 17361. Notice that proper time to see available output is almost 50ms, and it would takes some time(2~3 minutes).