Interfacing to Si-9986 H-Bridge for Ultrasonic Transducer Control via DE2

ECE 492
Application Note

Brett Griffin
Matthew Johnston

March 15, 2013

This component will produce an N bit pulse train @ 40 kHz(can be changed by modifying factors inside the code, see .vhdl file) as well as enable signals used to interface the DE2 board with the Si9986 buffered H-bridge used to drive transducers. For our application(two transducers communicating) we have two output pulse signals, and two enable signals which are based on the data written; this can be modified for a wide variety of applications. Also, for our application, all outputs are active low; the output pulses will be asserted low when required. This is necessary to drive the H-bridges which will in turn drive our transducers. It was required to write a custom component that would be active low to properly drive our h-bridge. Data is inputted via the NIOS IDE, by writing to registers from the io.h header file that will be generated.
*NOTE: a circuit diagram of the transducer/h-bridge interface will be uploaded at a later time

The function used is:

IOWR(BASE, REGNUM, DATA)

BASE corresponds to the base address of the tx_enable component generated in the SOPC builder and found in the system.h file. *REGNUM* will always be 0. *DATA* will be a 32-bit word whose contents will vary depending on the pulse train length required(for our applications we only use 6 bits of the 32).

31	30	---	---	5	4	3	2	1	0
---	---	---	---	pulse length	pulse length	pulse length	pulse length	A_en	B_en

Most of the bits are unused; bits 5 to 2 are used to specify the pulse length (max of 15, can be modified in the .vhd file), while bits 1 to 0 specify the A and B enable signals. These A and B enable signals are crucial for our h-bridge interface. The following truth table, taken from the Si9986 data sheet shows why we need active low signals from our component.

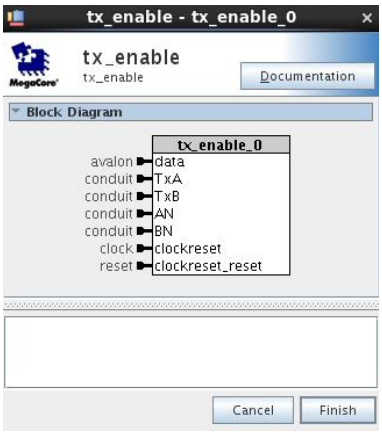
IN_A	IN_B	OUT_A	OUT_B
1	0	1	0
0	1	0	1
0	0	0	0
1	1	HIGH Z	HIGH Z

When both enable signals are 1, the outputs of the H-bridge will be high impedance. Because the enable signals, as well as the Tx pulse signals are active low, we will have high impedance

outputs on our H-bridge when we do not want the transducers operating. In this application, where two transducers are communicating with each other, their modes need to be controlled. When one transducer is transmitting a signal, the other will be in receive mode, and should not be driven by the H-bridge, and vice versa. The modes of the transducers are determined by the A_en and B_en signals of the written data. Both transducers will be connected to an H-bridge, whose inputs are A_en, B_en, TxA, and TxB. When one transducer is transmitting, it is being driven by the h-bridge. When the other transducer is receiving, the h-bridge that it is connected to will have high impedance outputs to ensure that the receiving transducer does not short circuit to ground. Referring again to the truth table above, we can see that the IN_A and IN_B inputs(connected to A_en and B_en from the tx_enable.vhd component), when the B h-bridge enable signal is (receive mode) 1 and Tx signals are 1 the outputs are high impedance. When the A h-bridge enable signal is 0(transmit mode), and Tx signal signal is pulsing, the Tx pulse is passed through the h-bridge to the transducer.

This code is fully functional and very modifiable; a pulse length of 2^{30} is possible. The pulse frequency generated by the file can be modified for transducers of a variety of resonant frequencies. The factors used to generate the pulse frequency depend on the system clock that is behind used(50 MHz for us) and users should note what system clock frequency they are using when determining their specific factors. Once imported into the SOPC builder it is easily integrated into any system, and can be controlled by using the IORW(...) function. Unfortunately, for users requiring precise control over the timing between pulse generation and transducer switching. ie. when IOWR(...)'s are required at small intervals(<10^{-6} seconds), NIOS II is insufficient and custom vhdl components are needed for controlling the tx_enable component.

Below is a screenshot of the component that was generated in the SOPC builder:

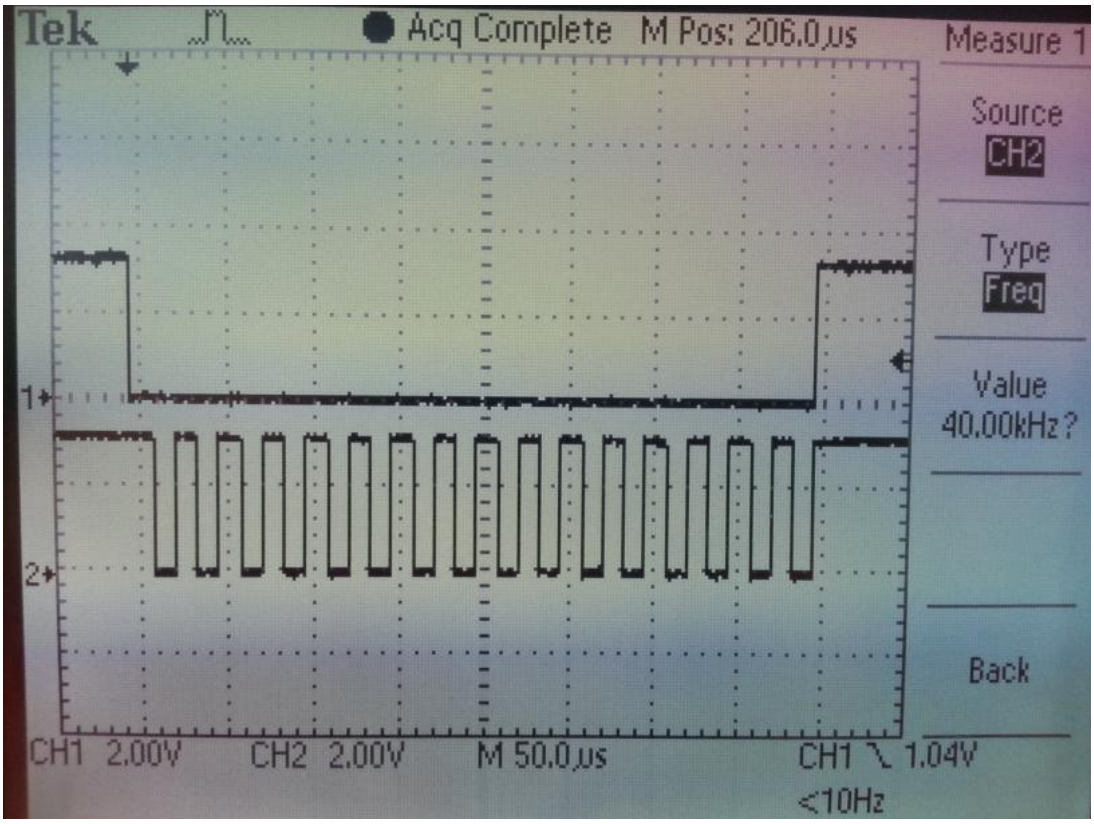


Note that the conduit signals are the signals that will be generated in your *projectname_system.vhd* file. Your top level entity will require port mapping of these four signals to GPIO pins.
A demonstration of the results (measured on the oscilloscope) shows proper operation of the component.

The output is initiated with the following command in NIOS II:

IOWR(tx_enable_0_base, 0, 0x0000003D);

This bit pattern corresponds to one 15-pulse train output on the TxA output, and an assertion of the A_en signal for duration of that pulse train. A shot of the oscilloscope displaying this data is below:



Channel one displays A_en, while channel two display TxA. Notice that the frequency reading from the oscilloscope displays the desired frequency. Probing the B_en, and TxB pins while A_en is being asserted also displays both B_en and TxB staying at logic 1 while A is transmitting, satisfying our needs.