File Name	Hierachy?	What does code do	Modifiable ? (Y/N) Not created by Xilinx?	What to Modify / Notes about file	IP Core? .xci file
Transmitter_TL.vhd	Top-level	Runs the entire transmitter process	Y		
Clk_gen_50Hz.vhd	Top-level component	50Hz clock (Uses a counter for frequency division – every rising edge, the count goes up until it reaches 10000 with a logic level of 1. When it's not a rising edge, it's logic level 0).	Y	N/A	
Clkgen_25.vhd	Top-level component	25Hz clock (This divides the 50Hz clock by half through its toggling behaviour – toggles on each rising edge, so slower)	Y	N/A	
clk_wiz_0.v	**Transmitter**	Clock generation/control	N	N/A	Yes
Clk_wiz_0_clk_wiz.v	Clk_wiz_0.v	Instantiation of clk_wiz	N	N/A	^
DC_FSM.vhd	**Controller (Transmitter)**	FSM Controls BFSK Datapath; feeding in 50Hz, this will reset/pause low for 1024 clock cycles to allow for all bits in desired time frame to be sent. Clock cycle 1025, we reset/pause high for one second (50 clock cycles), and after we return pause and reset to low and return to initial state.	Y	N/A Modify the pause time depending on how many bits are in input sequence Guard interval prevents post processing two frames after each other (clear visual border in readings)	
Bfsk_dc_datapath.vhd	**Phase_gen (Transmitter)**	Describes how everything is connected in datapath and thus, the flow of data and operations to generate a phase signal used in the BFSK modulation. Pn_control(0) and is connected to reset_pn and pn_control(1) is the inverse of reset_pn.	Y	Don't need anything associated with the pn_generator. There are signals defined to reset/control the generator. PN_Sequence_bit_count is a bit count used within the pn_sequence generator; allows user	

				to define length of pseudo-random sequence generated.	
Bit_2_phase.vhd	**Phase_gen**	Selects FSK based on data coming in (pn_data_in in this case) **96Hz** (hardware-limited) and assigns frequency hopping on count Essentially a control signal (0 or 1) determines whether the phase_out will be +96Hz or -96Hz Clock cycles through 13 subcarriers. **Placed between PN generator and DDS block.** Adds signed types of fh and phase_sel (+ or - 96Hz) and fits them into a 32-bit std_logic_vector.	Y	May need to change length of vector depending on how long we want vector Would need to change **FH levels** based on available **bandwidth** and **number of levels** (in case of JANUS = 13 levels) JANUS operates on a bandwidth of 9440-13600Hz For frequency hopping: bandwidth / number of levels = 5000 / 13 = 384.6154 and then divided by 4 --> 384.6154 / 4 = **96.1539** **"4" because the width of two subcarriers is 192**	
Encoder_wrapper.vhd	**Phase_gen**	Encoder calls on convolution component file to perform encoding in conv. Encoding selected bit (bit_selector) decides what data is output (either m_data(0) or m_data(1)), based on rising edge of clk_50 signal.	Y	N/A	
Convolution_0.vhd	**Encoder (phase gen)**	Convolution of data configured	N	N/A Whatever is fed into the input port s_axis_data_tdata needs to be an 8-bit	Yes

				vector, as seen in *encoder_wrapper.vhd* (Similar idea for all other input ports)	
Pn_wrapper.vhd	**Pn_gen (phase gen)**	Calls on pn_generator component; allows user to control behaviour through wrapper interface by connect signals. PN sequence bit count is set using the generic PN_SEQUENCE_BIT_COUNT	Y	**Can eliminate** or modify this code (don't need pseudo-random gen. bit stream)	
Pn_gen_noaxi	**Pn_gen (phase gen)**	Generates PN sequence	Y	**Can eliminate**	
Waveform_mod.vhd	**Wavegen (Transmitter)**	Calls all applicable components to perform waveform modulation	Y	N/A	
Signal_multiplier.vhd	**Wavegen (Transmitter)**	Multiplies data by defined scale factor	Y	Scale factor can be modified during simulation. It is set at an integer of **66** here because it was found to work best.	
Sigma_delta_mod.vhd	**Wavegen (Transmitter)**	Performs sigma-delta modulation algorithm (technique used in DACs for high res. by oversampling and applying feedback). The output is achieved from a digital-to-digital conversion. The output is quantized.	Y	N/A	
BFSK_Stream_combiner.vhd	**Wavegen (Transmitter)**	This combined the two input streams (even and odd) into a single output stream. The input streams are treated as signed values during the combination process.	Y	N/A	
Bb_dds.vhd (baseband)	**Wavegen**	Configures and uses the dds_compiler IP core to general digital sine/cosine wave forms based on the provided settings	N	N/A DDS (Digital Direct Synthesizer) is an IP core that generates complex signals. This may have not been modified, but C-PHASE-INCREMENT = 3 may determine output waveform	Yes

				frequency generated by DDS	
Pb_dds.vhd (passband)	**Wavegen**	Also instantiates the dds_compiler IP core, but smaller output data width and other parameters); depends on use case / what it is needed for	N	N/A Similar to bb_dds	Yes
BFSK_Upconverter.vhd (even up)	**Wavegen**	Upconversion process of the even signal (BFSK mod. by multiplying carrier signal with input data)	Y	Multiplication and additions are performed at **10MHz** and the passband DDS runs at **100MHz** Need to change carrier frequency for different specifications	
BFSK_Upconverter (odd up)	**Wavegen**	Upconversion process of the odd signal (BFSK mod. by multiplying carrier signal with input data)	Y	Need to change carrier frequency for different specifications	

Modules to Implement:
Input Module

- As per the README.md file in the JANUS module, this protocol focuses on defining how the user data is encoded. When **configuring an input module**, the user data is fed into the system and will need to be appended to the JANUS bit stream packet.
- The structure of a JANUS bit stream packet is defined in Table 1 and Figure 2 of the README file.
- A 32-bit preamble needs to precede JANUS header packet (see README in *JANUS module*) before sending the data into the transmitter.

Cyclic Redundancy Check Module (Precedes convolutional encoder and interleaving modules)

- The above will be passed through **a module that performs a Cyclic Redundancy check**.

All components follow this diagram:

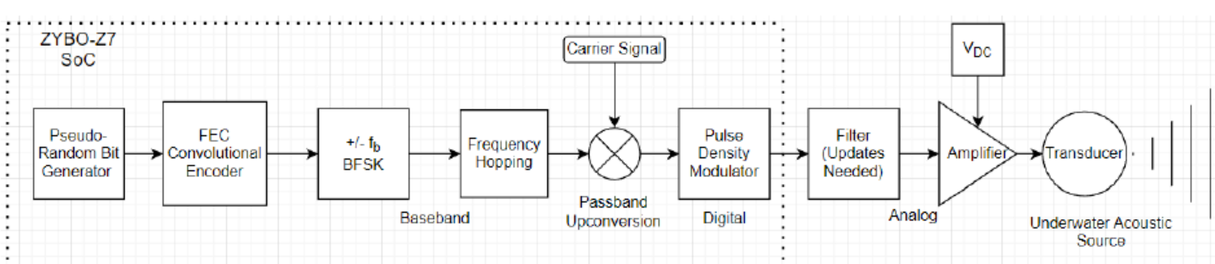

Figure 7: Current ACOM transmitter block diagram of Dr. Bousquet's implementation