Comments on the structural of FIR2

1. The most important design in implementation 2 is that we use 2 different clocks: clk and inner\_clk. Since in DFG it has been shown that the latency of implementation 2 is 11, we can choose T\_clk = 11 \* T\_inner\_clk which is very safe and easy. However we think the T\_clk can be shorter since between reg\_ins and reg2 just like a pipeline, which means we can read next data\_in (or to say, start the next clk cycle) in C.S.9 and do the first two multiplications of the next clk cycle. We just need to make sure that reg3 is clear at C.S.2 (we can jump C.S.0 and C.S.1 since the same work have been done in the previous C.S.9 and C.S.10) when the adder1 is dealing with the next clk cycle. Since we have joint the next C.S.0 and C.S.1 with the previous C.S.9 and C.S.10, we can choose T\_clk = 9 \* T\_inner\_clk. It should be noticed that clk is beginning at C.S.9 (meanwhile is C.S.0) however the clear signal should be sent to reg3 in C.S.10 (meanwhile is C.S.1) to make sure reg3 is clear in C.S.2. It's unnessesary to worry about whether we can collect the data\_out in-time, because the final 32-bit result which contains data\_out transfers directly from adder1 to reg\_final in C.S.10 (C.S.2) while reg2 and reg3 still cantains the data of the previous clk cycle.

2. In order to ease the control unit (which is a FSM with counter from 0 to T\_clk/T\_inner\_clk-1) which we will add into account next time, we make some changes which may cost a little more power and space.

a. We make both multiplexers consider all data, which will make the CL signal of mux0 and the read\_address signal of rom0 the same (we use CL0 here). Meanwhile the CL signal of mux1 and the read\_address signal of rom1 will be CL1 <= CL0 + 1.

b. Since in (a) we difined CL1 <= CL0 +1, so for each CL0 there must be a signal CL1 send to mux1 and rom1. We should make sure the mux1 and rom1 can always work without any no\_pointer\_error when receive any possible CL1. In this FIR filter, we have 17 coefficients, so that we only use 17 registers to keep data\_ins. However, CL1="10001" is the situation must happen in each clk. To solve this problem, we add an extra (others=>'0') in rom1 (also rom0) and make mux1 (also mux0) return (others=>'0') when CL1="10001" accours, so that in C.S.8 in DFG, the mul1 returns ('others=>0') which will not disturb data\_out.

c. When we choose T\_clk = 11 \* T\_inner\_clk, we don't care about what mul0 and mul1 do in C.S.9, also what adder0 do in C.S.10. So we just need to make CL0 and CL1 be (others=>'0') in C.S.9 and C.S.10 to avoid no\_pointer\_error. By the way, we need to clear reg0, reg1, reg2 and reg3 before C.S.0 in this case. If we choose T\_clk = 9 \* T\_inner\_clk, we just need to clear all the registers when the FIR filter start running and don't clear registers except reg3 (which we has discussed before) anymore during the filter running.

3. To avoid overflow, we make the inputs and output of adder0 and adder1 to be 32 bits. In the next clk cycle, we get data\_out which is (28 downto 13) bits of the last output of adder1 in the previous clk cycle.

4. To ease the operations done by mul0 and mul1, we use 16 bits to store each coefficient in rom0 and rom1, so that the size of each rom is 18\*16 bits.(Why is 18? See 2.b.)

Comments on the rom

1. We directly write the code which is used to load coefficients in the behavioral architecture of rom.

2. Except the coefficients(0 to 16), we add an extra (others=>'0') which increase the row of rom by 1: rom(17) <= (others=>'0').

3. We add clock into input so that the rom can be synchronous.

Comments on the multiplexer

1. The multiplexer we write accepts array of std\_logic\_vector (which we defined in a package which we wrote in the same vhd file) as input, so that the dataflow architecture of multiplexer can be very easy (just like ROM).