Entities involved: PRBS, Clock Generator, S/P & P/S Converter, RAM or Buff Module, (OOK, PPM, DDPM and DPIM Modulator and Demodulator)

**At transmitter**   
Clock Generation  
PRBS Data -> OOK Modulated Data  
PRBS Data -> S/P Converter -> PPM Modulator -> PPM Modulated Data  
PRBS Data -> S/P Converter-> Buff Module -> DPPM Modulator -> DPPM Modulated Data  
PRBS Data -> S/P Converter-> Buff Module -> DPIM Modulator -> DPIM Modulated Data

**At Receiver**  
Clock Recovery   
Received Data -> OOK Demodulator -> Recovered Data  
Received Data -> PPM Demodulator -> P/S Converter -> Recovered Data  
Received Data -> Buff Module -> DPPM Demodulator -> P/S Converter -> Recovered Data  
Received Data -> Buff Module -> DPIM Demodulator -> P/S Converter -> Recovered Data

1. While generating the hardware module we set the length of sequence by selecting appropriate values of ***N*** in top module entity generation. Length of sequence will be given by L = (2^N - 1) as all zero's will not be included

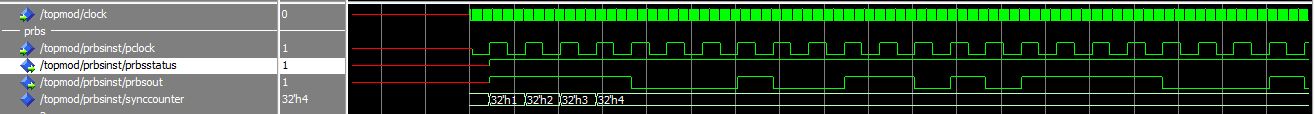
2. Also as the clock frequency that will generated from Virtex2Pro board will be in tens of MHz, but as LASER driver that we have is incapable of being modulated beyond 10kHz so we choose a value of clock divider by selecting appropriate values for ***div*** parameter in top module entity generation

3. Clock generator will generate clock at different frequencies for different modules to make them work synchronously. The ratio of clock divider for different modules would be

Ref : PRBS : P/S & S/P Block : PPM : DPPM : DPIM = 1 : 2^N : N\* 2^N : N : 2N : 2N  
So the ratio of primary clock frequencies of different modules will be

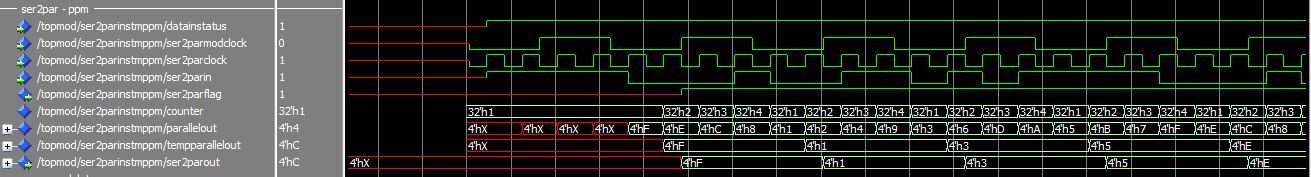
PRBS : P/S & S/P Block : PPM : DPPM : DPIM = ref/ 2^N : ref/N\* 2^N : ref/N : ref/2N : ref/2N

4. **PRBS** will be based on LFSR. It has been programmed to generate sequence of varying length (2^N - 1). Here N can take values {2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,32,64,128}. But it can be altered to include tap points accordingly for other lengths. PRBS will be loaded with an initial vector 0x01. As it is an SSRG implementation so it will shift arbitrarily to other phase sequence. It will loop through all possible sequences before repeating itself as tap points are chosen for maximal length. In PRBS module we also have one flag register to signal the presence of first valid data bit. Since the length of LFSR is N bit so we consider the first valid bit to be present after N clock cycles as we wait for initial N pulses before generating the random data. (This has been done to ensure that a valid data be present in S/P shift register while doing the simulation, this can be done away with while doing the actual implementation since either 1 or 0 will be present in S/P shift register in its hardware implementation). So for the first N clock cycle we are generating a high signal from PRBS output, then our loading vector will come from PRBS output and subsequently other phase vectors depending on tap points. Also one asynchronous ***preset*** signal is provided to reset the LFSR by loading vector 0x01. A counter has been implemented to set the flag ***prbsstatus*** high once it count N pulses.



5. For OOK modulation PRBS data will be taken as output

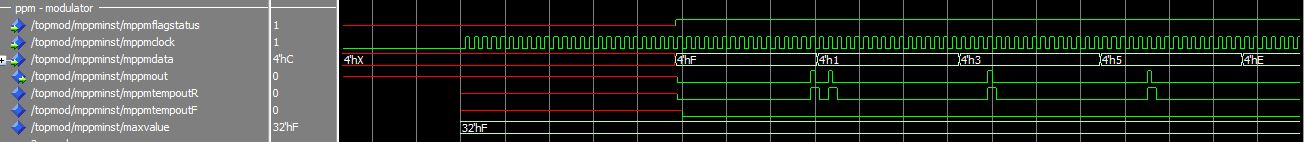
6. **S/P and P/S** block will take the input signal (clock to read the input is at ref/2^N) and will present the output at every Nth pulse (clock to present the output is at ref/N \*2^N). A ***datainstatus*** signal will be present to signal the presence of first “valid” databit (explained earlier) from PRBS so that it start reading it at its ***ser2parin*** pin. A flag ***ser2parflag*** has been included to signal presence of first valid vector output from output ports of S/P module. Here we have used one temporary signals ***parallelout*** which will be used to form a temporary vector output by shifting the content of register and appending ***ser2parin*** on each pulse of ***ser2parclock.***Since while doing the simulation undefined data may be present on signal for first N pulses of prbs clock (after which it will be all 1’s) so we have included one more temporary signal ***tempparallelout*** which will check for undefined data sequence on leftmost or msb of ***parallelout*** (as when shifting is done, msb will have a valid data only when all previous bit contains valid data ). A counter has been implemented for this purpose which will count up to N pulses of ***ser2parmodclock***. Once ***tempparallelout*** contain all valid data point it will be made available to ***ser2parout*** on next rising edge of ***ser2parmodclock.*** (All these complexities can be done away with while doing the hardware implementation since undefined data won’t be present). So first **parallelout** will have data and once it have all valid data it will be given to **tempparallelout**. So there will be a delay of one ***ser2parmodclock*** pulse between these two signals. Once valid data is available on ***tempparallelout*** then it will be given to ***ser2parout*** and **ser2parflag** will be set high on the next rising edge of ***ser2parclock.*** Also notice that nearest rising edge of ***ser2parclock*** and ***ser2parmodclock*** are offset by half pulse width of ***ser2parmodclock***. So there will be delay of half ***ser2parmodclock*** pulse between ***tempparallelout*** and ***ser2parout***Since first all one’s were send from PRBS output so first vector output from S/P module will be 0xF. Next loading vector will come and subsequently other phase vectors.



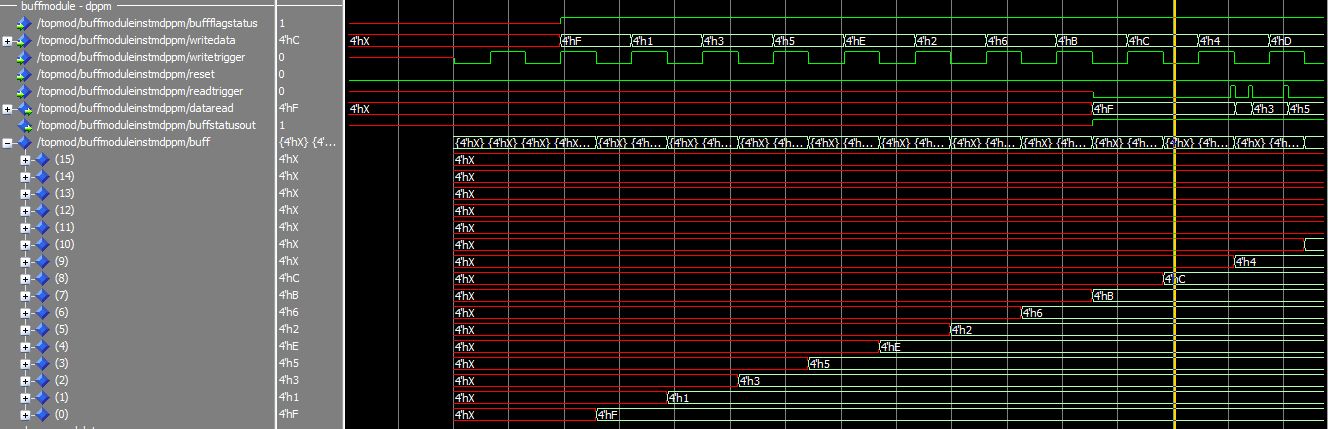
Consider N = 4, So maximum length of sequence will be 15 (since we will not include all zero's sequence). Now let's say last four bit serially output from PRBS be 0001. Serial to parallel will convert it to a vector (whose integer value will be 1) which will be given to modulator blocks.

7. **PPM Modulator**

PPM modulator will generate the pulse at the integer equivalent position of input vector. To generate the half pulse width it is added with ***mppmclock.*** Other positions of ppm modulated output for this cycle will be devoid of pulse. Here counter will go from state (0 to 2^N – 1) in one cycle and once it reaches the max position it will be reset to 0.



8. **Buffer Module**

Buffer element have a memory of 2^N. So read and write address will take values from 0 to 2^N – 1.  
It will wait for the flag bit of ***ser2parflag*** to indicate presence of valid data, after which it will start writing the input vector data to its memory positions on each falling edge of ***ser2parclock***. So the ***ser2parclock*** acts as a write signal for this module. Once the counter reaches 2^(N-1) or half the total number of possible sequence it signals the subsequent circuitry to indicate the presence of enough amount of data to carry out its operation by setting the ***buffstatusout*** bit to high. The buffer module has been implemented as a circular buffer, since when the write address reaches the maximum position of 2^N – 1 then it is being set to 0 again. When read signal is send then it pass the data at its current read address position and increment the read address by 1. Here we have used the falling edge of dppm or dpim modulated data to act as trigger signal for buffer module.  
  


9. **DPPM Modulator**

DPPM modulator will generate the pulse at the integer equivalent position of input vector. To generate the half pulse width it is added with ***mdppmclock.*** Once the position equivalent to integer value of input vector has been reached the counter will get reset to 0. The next input vector is fetched by passing the mdppm modulated data act as read trigger to buffer module. And the cycle repeats. At any given time buffer module would have sufficient number of values stored in it which needs to be modulated and also the number of values will such that it will avoid the writing over unsent data. This happens because of randomness and uniformity of prbs generator.

