Super Sample Rate FIR (data rate > clock rate) Implementation using Vivado HLS

Alex Paek
July 2014

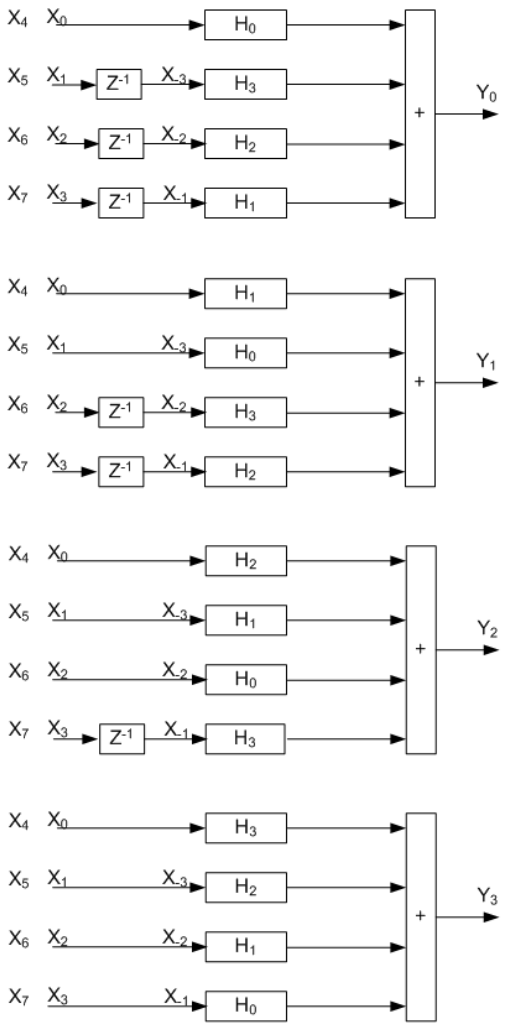
Overview

- **This document describes the implementation of the super sample rate (SSR) filter – where the sample data rate is greater than the clock rate using Vivado HLS tool**
- **The derivation of the parallel filter is described in "Appendix" section**
- **The initial target device is K7 -1 (the lowest speed grade), but the design – described in C++, can be targeted to most of Xilinx FPGAs.**
- **The goal specification for this filter:**
  - The effective data rate = 1.6 Gsps (= 4 * clock rate)
  - 128 tap symmetric coefficients
  - Clock rate = 400 MHz
  - Resource: 240 DSP48, 12994 LUT, 18415 FF, 0 BRAM
- **Tools:**
  - Vivado HLS 2014.2
  - Vivado Tool suite 2014.2

Parallel Filter

- The block diagram and the top level C code for 4th order (L=4) parallel filter are shown.
- Note on the C code:
  - din[L] corresponds to X0/1/2/3 in the right figure
  - dout[L] corresponds to Y0/1/2/3
  - Pipeline/inline/reshape directive at the top level
  - “init_coef” initialize/decompose the filter into subfilter coefficients

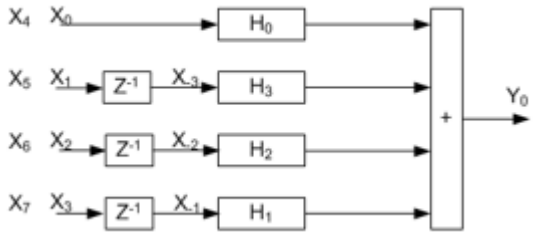
```
194 //_____________________________________________________________________
195 void fir_super( DATA_T din[L], DATA_T dout[L] ) {
196 #pragma HLS INLINE region recursive
197 #pragma HLS PIPELINE II=1
198 #pragma HLS ARRAY_RESHAPE variable=din,dout complete
199 //static COEF_T cin[L][NCOEF];
200 COEF_T cin[L][NCOEF];
201 init_coef(cin);
202
203     dout[0] = fir0( din, cin );
204     dout[1] = fir1( din, cin );
205     dout[2] = fir2( din, cin );
206     dout[3] = fir3( din, cin );
207
208 }
```



- The sampling time of all the instances is 4*T, where T is the data rate
- H_k is the FIR whose coefficients are: h_{k+4*l}, for ex, H_0 has h_0,h_4,h_8..

“fir0”

- The block diagram and the C code of one of 4 parallel filter are shown.
- Note how the register is inferred.
- In 2013.2 version, using constructor to initialize coefficient did not work well (it built logic instead of ROM table. it worked correctly for one instantiation class). Instead, calling the separate function to initialize the coefficients and passing it as argument worked. It should work in 2015.2!




```
4  //_____________________________________________________________________
5  DATA_T fir0( DATA_T din[L], COEF_T cin[L][NCOEF] ) {
6
7  DATA_T d0,d1,d2,d3;
8  static DATA_T r0=0,r1=0,r2=0,r3=0;
9  DATA_T dout;
10
11 static fir_nosym_obj<NCOEF, DATA_T, COEF_T, PROD_T, ACC_T> f0;
12 static fir_nosym_obj<NCOEF, DATA_T, COEF_T, PROD_T, ACC_T> f1;
13 static fir_nosym_obj<NCOEF, DATA_T, COEF_T, PROD_T, ACC_T> f2;
14 static fir_nosym_obj<NCOEF, DATA_T, COEF_T, PROD_T, ACC_T> f3;
15
16 d0 = f0.process(din[0], cin[0]);
17 d1 = f1.process(r1, cin[3]);
18 d2 = f2.process(r2, cin[2]);
19 d3 = f3.process(r3, cin[1]);
20
21 dout = d0 + d1 + d2 + d3;
22
23 r1 = Reg(din[1]);
24 r2 = Reg(din[2]);
25 r3 = Reg(din[3]);
26
27 return dout;
28
29 }
30
```

```
77 //_____________________________________________________________________
78 //  register
79 //
80 template<class T>
81 T Reg(T in) {
82 #pragma HLS PIPELINE
83 #pragma HLS INLINE off
84 #pragma HLS INTERFACE port=return register
85 return in;
86 };
87
```

FIR class

- **"Inline" recursively turned out to be one of the critical pragma to make this work. "inline" flattens the design and optimizes across the hierarchy.**
- **It would be worth checking:**
  - Make sure MAC function is mapped to DSP48 so the timing would be close to the speed of DSP48. There are 512 MAC operation and only 240 DSP48 is used. There are more than 240 non zero coefficients, so look like some non zero coefficients are mapped to LUT.
  - Do we need to code this MAC function in a structural manner to guarantee DSP48 mapping and PCout of one DSP48 is connected to PCin of the adjacent DSP48?
  - It would be useful to create a template class where L, the order of parallel filter, can vary. L does not have to be any number to be practical. It can be just 2,4,8,16.

```
29 //------------------------------------------------------------------------
30 // fir class
31 //
32 template<int ncoef, class data_t, class coef_t, class prod_t, class acc_t>
33 class fir_nosym_obj {
34
35     data_t shift_reg[ncoef];
36     acc_t  acc;
37
38 public:
39 fir_nosym_obj() {}
40
41 // MAC
42 acc_t MAC( data_t din, coef_t coef, acc_t  acc ) {
43
44     prod_t prod = din*coef;
45     acc_t  sum  = prod + acc;
46
47     return sum;
48 };
49
50 // filter
51 data_t process( data_t din, coef_t c[ncoef]) {
52 #pragma HLS INLINE
53 #pragma HLS pipeline
54 #pragma HLS array_reshape variable=c complete dim=1
55 #pragma HLS array_reshape variable=shift_reg complete dim=1
56
57     int i;
58     acc = 0;
59
60     loop_mac:
61         for (i=0; i<ncoef; i++) {
62             acc = MAC(shift_reg[i], c[i], acc);
63         }
64
65     loop_sr:
66         for (i=ncoef-1; i>0 ; i--) {
67             shift_reg[i] = shift_reg[i-1];
68         }
69         shift_reg[0] = din;
70
71     return acc;
72
73 };
74
75 };
```

C Synthesis

- Shown below is the output of C to RTL synthesis
- FF/LUT count seems high

Synthesis Report for 'fir_super'

General Information

Date:	Mon Jul 14 14:24:10 2014
Version:	2014.2 (Build 928826 on Thu Jun 05 17:25:20 PM 2014)
Project:	HLS_proj_2014.2
Solution:	sol1
Product family:	kintex7 kintex7_fpv6
Target device:	xc7k160tfbg484-1

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
default	2.50	1.98	0.31

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
11	11	1	1	function

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	0	11200
FIFO	-	-	-	-
Instance	-	240	48	0
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	21852	940
Total	0	240	21900	12140
Available	650	600	202800	101400
Utilization (%)	0	40	10	11

Interface

Summary

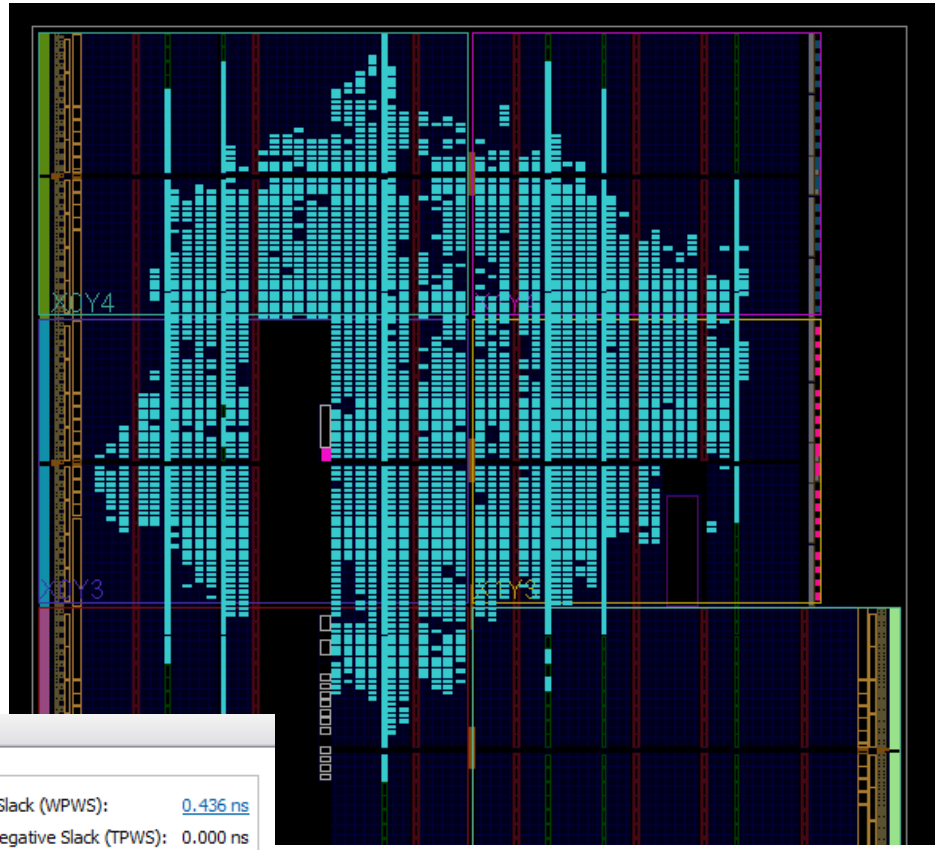
RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	fir_super	return value
ap_rst	in	1	ap_ctrl_hs	fir_super	return value
ap_start	in	1	ap_ctrl_hs	fir_super	return value
ap_done	out	1	ap_ctrl_hs	fir_super	return value
ap_idle	out	1	ap_ctrl_hs	fir_super	return value
ap_ready	out	1	ap_ctrl_hs	fir_super	return value
din_V	in	64	ap_none	din_V	pointer
dout_V	out	64	ap_vld	dout_V	pointer
dout_V_ap_vld	out	1	ap_vld	dout_V	pointer

FPGA Implementation

- Post implementation results shown:
- Fmax = 1/(2.5-.071) = 411 MHz

Utilization - Post-Implementation

Resource	Utilization	Available	Utilization %
FF	18415	202800	9.08
LUT	10470	101400	10.33
Memory LUT	366	35000	1.05
DSP48	240	600	40.00



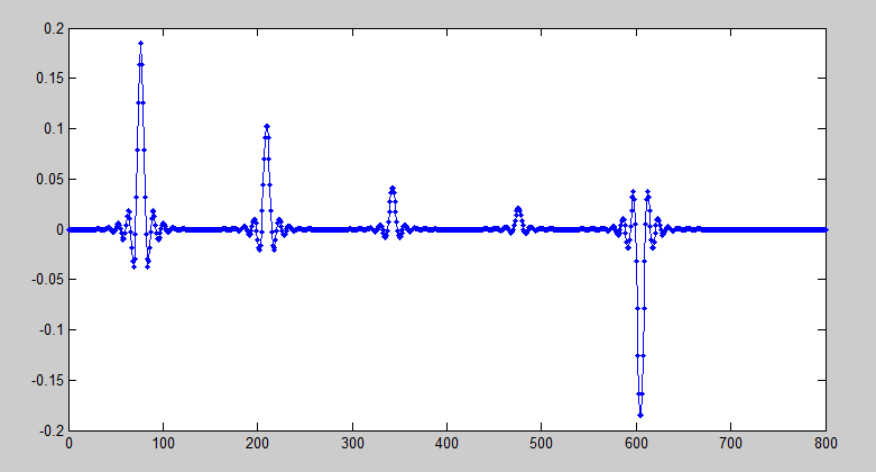
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.071 ns	Worst Hold Slack (WHS):	0.038 ns	Worst Pulse Width Slack (WPWS):	0.436 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	42274	Total Number of Endpoints:	42274	Total Number of Endpoints:	19249

All user specified timing constraints are met.

Design Files

- **fir_super.cpp, fir_super.h: design source codes**
- **fir_super_test.cpp: testbench code. To check the output of the testbench simulation in matlab:**
  - load out.dat; x=out';y=x(:);figure,plot(y,'.-');
  - You should see several impulse responses
- **run_hls.tcl: TCL file to run HLS tool**



Design Review Question

- Is this a sample based design or packet based design?
- Can we better map to DSP48? Better way to code the MAC?
- Can you make this a template function? What would be a good template parameters? How about L (=Data rate/Clock rate)?
- Compare the Fmax and resource with FIR Compiler (2015.2 FIR Compiler has a SSR feature)
- Export this to Sysgen and build a test bench

Reference

1. Vivado HLS User's Guide, UG902
2. "Parallel FIR Design", XKB #54281 article

Appendix

General idea

- The data is assumed coming in a demuxed into L streams where clock rate = data rate/L
- The original filter is decomposed into L subfilters operating at a clock rate and the output of each subfilter can be muxed to achieve the desired data rate
- The overall decomposed structure consists of subfilters and delay elements and adder
- Each subfilter can be implemented using FIRcompiler
- There is a nice pattern to the decomposed structure that this process can be automated for any data/clock rate
- This process can be applied not just a single data rate filter, but to interpolating filter, interpolated filter (IFIR)
- This basic derivation can be optimized even further by the algorithm known as FFA (fast filter algorithm)

Derivation

- Parallel FIR structure is derived using polyphase decomposition commonly used in multi-rate filtering
- N tap FIR is expressed as:

$$y_k = \sum_{i}^{N-1} x_{k-i} * h_i$$

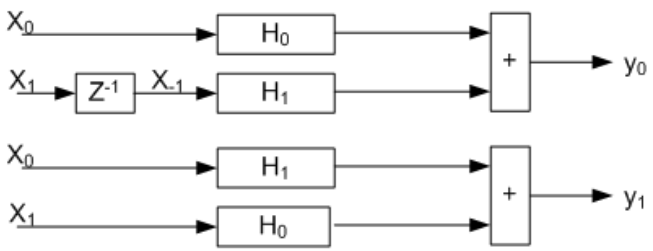
- Which can be written out as:

$$y_0 = x_0 * h_0 + x_{-1} * h_1 + x_{-2} * h_2 + x_{-3} * h_3 + x_{-4} * h_4 + x_{-5} * h_5 + x_{-6} * h_6 + x_{-7} * h_7 + ..$$
$$y_1 = x_1 * h_0 + x_0 * h_1 + x_{-1} * h_2 + x_{-2} * h_3 + x_{-3} * h_4 + x_{-4} * h_5 + x_{-5} * h_6 + x_{-6} * h_7 + ..$$
$$y_2 = x_2 * h_0 + x_1 * h_1 + x_0 * h_2 + x_{-1} * h_3 + x_{-2} * h_4 + x_{-3} * h_5 + x_{-4} * h_6 + x_{-5} * h_7 + ..$$
$$y_3 = x_3 * h_0 + x_2 * h_1 + x_1 * h_2 + x_0 * h_3 + x_{-1} * h_4 + x_{-2} * h_5 + x_{-3} * h_6 + x_{-4} * h_7 + ..$$

- For 2 parallel filter structure, this can be re-written as below and its implementation is shown.

$$y_0 = [x_0 * h_0 + x_{-2} * h_2 + x_{-4} * h_4 + x_{-6} * h_6 + \cdots] + [x_{-1} * h_1 + x_{-3} * h_3 + x_{-5} * h_5 + x_{-7} * h_7 + ..]$$

$$y_1 = [x_1 * h_0 + x_{-1} * h_2 + x_{-3} * h_4 + x_{-5} * h_6 + ..] + [x_0 * h_1 + x_{-2} * h_3 + x_{-4} * h_5 + x_{-6} * h_7 + ..]$$



- The sampling time of all the instances is 2*T, where T is the data rate
- H_k is the FIR whose coefficients are: h_{k+2^*l}, for ex, H_0 has h_0, h_2, h_4..

3 parallel filter

- 3 parallel filter structure is derived:

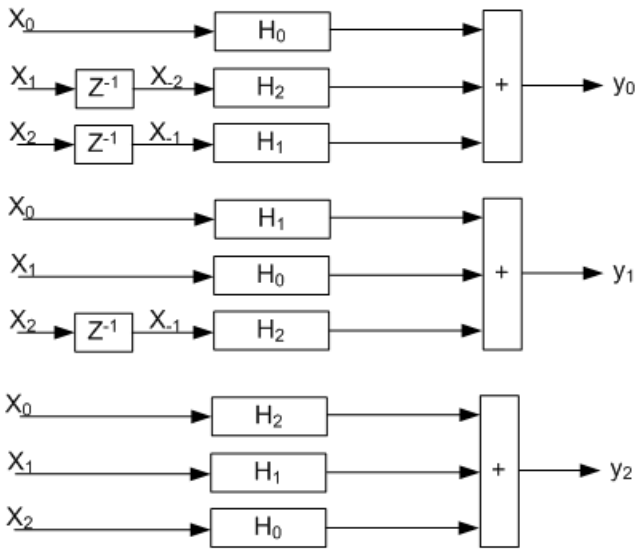
$$y_0 = x_0*h_0 + x_{-1}*h_1 + x_{-2}*h_2 + x_{-3}*h_3 + x_{-4}*h_4 + x_{-5}*h_5 + x_{-6}*h_6 + x_{-7}*h_7 + ..$$
$$y_1 = x_1*h_0 + x_0*h_1 + x_{-1}*h_2 + x_{-2}*h_3 + x_{-3}*h_4 + x_{-4}*h_5 + x_{-5}*h_6 + x_{-6}*h_7 + ..$$
$$y_2 = x_2*h_0 + x_1*h_1 + x_0*h_2 + x_{-1}*h_3 + x_{-2}*h_4 + x_{-3}*h_5 + x_{-4}*h_6 + x_{-5}*h_7 + ..$$
$$y_3 = x_3*h_0 + x_2*h_1 + x_1*h_2 + x_0*h_3 + x_{-1}*h_4 + x_{-2}*h_5 + x_{-3}*h_6 + x_{-4}*h_7 + ..$$

- For 3 parallel filter structure, this can be re-written as below and its implementation is shown.

$$y_0 = [x_0*h_0 + x_{-3}*h_3 + x_{-6}*h_6 + ..] + [x_{-1}*h_1 + x_{-4}*h_4 + x_{-7}*h_7 + ..] + [x_{-2}*h_2 + x_{-5}*h_5 + +x_{-8}*h_8 + ..]$$

$$y_1 = [x_1*h_0 + x_{-2}*h_3 + x_{-5}*h_6 + ..] + [x_0*h_1 + x_{-3}*h_4 + x_{-6}*h_7 + ..] + [x_{-1}*h_2 + x_{-4}*h_5 + +x_{-7}*h_8 + ..]$$

$$y_2 = [x_2*h_0 + x_{-1}*h_3 + x_{-4}*h_6 + ..] + [x_1*h_1 + x_{-2}*h_4 + x_{-5}*h_7 + ..] + [x_0*h_2 + x_{-3}*h_5 + +x_{-6}*h_8 + ..]$$



- The sampling time of all the instances is 3*T, where T is the data rate
- H_k is the FIR whose coefficients are: h_{k+3*l}, for ex, H_0 has h_0, h_3, h_6..

4 parallel filter

- **4 parallel filter structure is derived:**

$$y_0 = x_0 * h_0 + x_{-1} * h_1 + x_{-2} * h_2 + x_{-3} * h_3 + x_{-4} * h_4 + x_{-5} * h_5 + x_{-6} * h_6 + x_{-7} * h_7 + ..$$
$$y_1 = x_1 * h_0 + x_0 * h_1 + x_{-1} * h_2 + x_{-2} * h_3 + x_{-3} * h_4 + x_{-4} * h_5 + x_{-5} * h_6 + x_{-6} * h_7 + ..$$
$$y_2 = x_2 * h_0 + x_1 * h_1 + x_0 * h_2 + x_{-1} * h_3 + x_{-2} * h_4 + x_{-3} * h_5 + x_{-4} * h_6 + x_{-5} * h_7 + ..$$
$$y_3 = x_3 * h_0 + x_2 * h_1 + x_1 * h_2 + x_0 * h_3 + x_{-1} * h_4 + x_{-2} * h_5 + x_{-3} * h_6 + x_{-4} * h_7 + ..$$

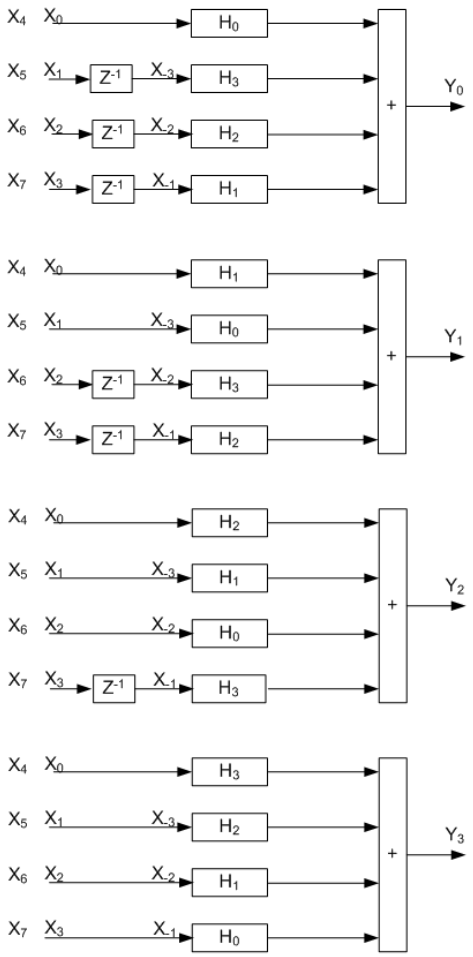
- **These can be re-written as:**

$$y_0 = [x_0 * h_0 + x_{-4} * h_4 + x_{-8} * h_8 + ..] + [x_{-1} * h_1 + x_{-5} * h_5 + x_{-9} * h_9 + ..] + [x_{-2} * h_2 + x_{-6} * h_6 + +x_{-10} * h_{10} + ..] + [x_{-3} * h_3 + x_{-7} * h_7 + +x_{-11} * h_{11} + ..]$$

$$y_1 = [x_1 * h_0 + x_{-3} * h_4 + x_{-7} * h_8 + ..] + [x_0 * h_1 + x_{-4} * h_5 + x_{-8} * h_9 + ..] + [x_{-1} * h_2 + x_{-5} * h_6 + +x_{-9} * h_{10} + ..] + [x_{-2} * h_3 + x_{-6} * h_7 + +x_{-10} * h_{11} + ..]$$

$$y_2 = [x_2 * h_0 + x_{-2} * h_4 + x_{-6} * h_8 + ..] + [x_1 * h_1 + x_{-3} * h_5 + x_{-7} * h_9 + ..] + [x_0 * h_2 + x_{-4} * h_6 + +x_{-8} * h_{10} + ..] + [x_{-1} * h_3 + x_{-5} * h_7 + +x_{-9} * h_{11} + ..]$$

$$y_3 = [x_3 * h_0 + x_{-1} * h_4 + x_{-5} * h_8 + ..] + [x_2 * h_1 + x_{-2} * h_5 + x_{-6} * h_9 + ..] + [x_1 * h_2 + x_{-3} * h_6 + +x_{-7} * h_{10} + ..] + [x_0 * h_3 + x_{-4} * h_7 + +x_{-8} * h_{11} + ..]$$



- The sampling time of all the instances is 4*T, where T is the data rate
- H_k is the FIR whose coefficients are: h_{k+4*i}, for ex, H_0 has h_0, h_4, h_8..

Generalization

- Following the previous derivations, Lth order parallel filter structure can be generalized to the structure on the right
- The data stream, whose rate is L*clock rate, is demuxed into L streams at the clock rate
- The sampling time for this structure is clock rate (=data rate/L)
- H_k is the FIR whose coefficients are h_{k+L*j}, for ex., H_0 has h_0, h_L, h_{2*L}