Implementing FIR filter Using Vivado HLS

Alex Paek, Jim Wu
Aug 2015

Overview

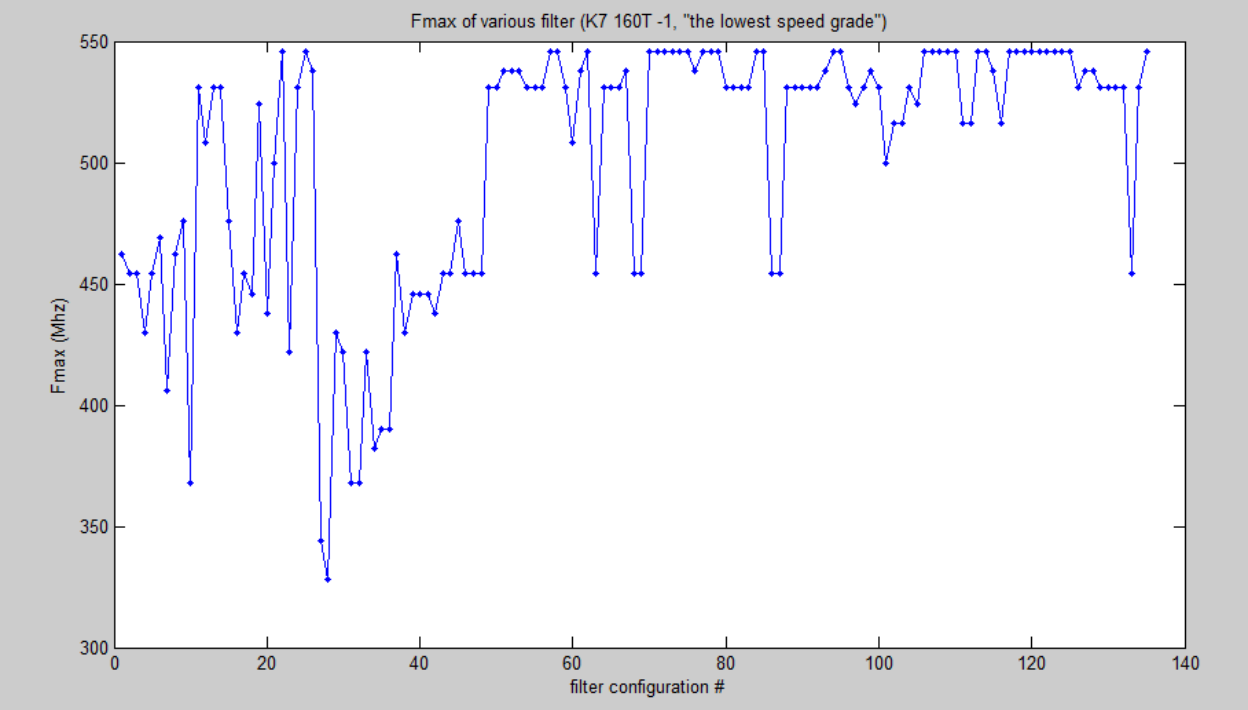
- **Goal:**
  - Build various single stage FIR and multistage FIR filters using Vivado HLS tool
  - Easily parameterizable design, such as filter coefficients, bit width of data/coefficients, data rate
  - Easily maintainable code structure using C++ template class
  - Demonstrate consistent QOR for various filter configuration – FPGA resource such as DSP48 close to theoretical numbers, clock rate > 300 MHz
  - Automatically optimized for a user specified data sample rate. Show data rate vs resource trade off.
  - Show recommendation on coding style and HLS optimization techniques
- **Tools:**
  - All the modules are designed using HLS (2015.1), and using C/C++ testbench
  - Matlab is used to design filter coefficients, and Simulink/Sysgen is used for verification of the HLS designs

FIR Compiler Performance

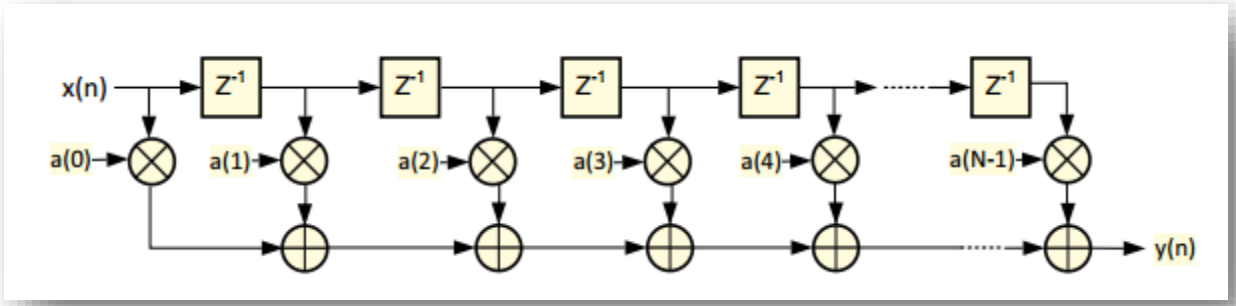
- The Fmax, the maximum clock rate, of various filter designs generated by FIR Compiler are shown on the graph below (for 135 filters, targeting K7 160T -1, the lowest speed grade). These numbers are from the spreadsheet downloaded from:

http://www.xilinx.com/products/intellectual-property/FIR_Compiler.htm

- These Fmax numbers can potentially increase by 10-20 percent for -2 and -3 speed grade.



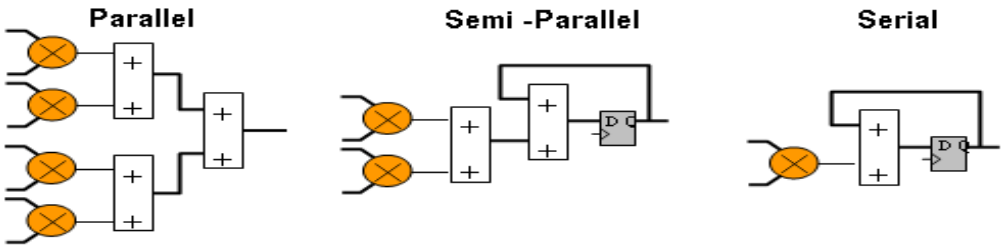
Filter type



- **The following filter types are implemented**
- **Single rate type**
  - There is no rate change between the input and the output
  - Coefficient type: symmetric, non-symmetric, halfband
- **Rate changing type**
  - Increase (interpolate) or decrease (decimate) the data rate by a integer ratio = 2

Resource Sharing

- **FIR Compiler produces resource & speed optimum implementation**
- **Depending on the data rate, clock rate, a FIR can be implemented in serial, parallel or semi-parallel fashion**



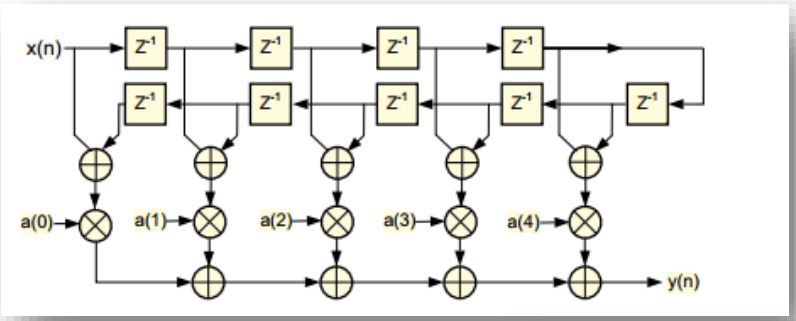
- **Number of MAC (multiplier-accumulator):**

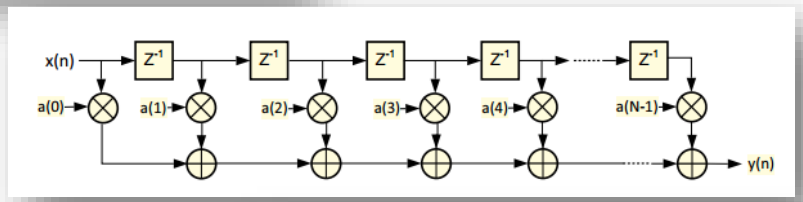
$$\text{Number of MACs required} = \frac{OutputDataRate * NumberOfTaps * NumberOfChannels}{ClockRate}$$

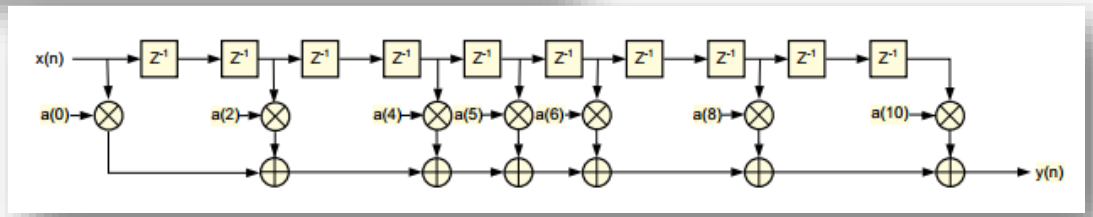
- **This number can vary depending on:**
  - **Exploiting coefficient properties, such as even/odd symmetry, zero coefficients (in halfband filter)**
  - **Coefficient bit width (max 49 bit)**
  - **Output rounding selection**
  - **Target device (number of DSP48 in each column)**

fir.h (FIR class, single rate)

- **These are filters with input rate = output rate**
  - sym_class: single rate symmetric FIR class. Can be even or odd length.
  - nosym_class: single rate non symmetric FIR class
  - hb_class: single rate halfband class. Every other coefficients are zero except the main tap. Always odd length.







fir.h (FIR class, rate changing)

- **Interpolation filter class: these are filters with the output rate = 2* input rate**
  - interp2_class: “interpolate by 2” class. Number of filter must be odd, so two sub-filters becomes symmetric
  - hb_interp2_class: “halfband FIR with interpolate by 2” class
- **Decimation filter class: these are filters with the output rate = ½* input rate**
  - decim2_class: “symmetric FIR with decimate by 2” class
  - hb_decim2_class: “halfband FIR with decimate by 2” class

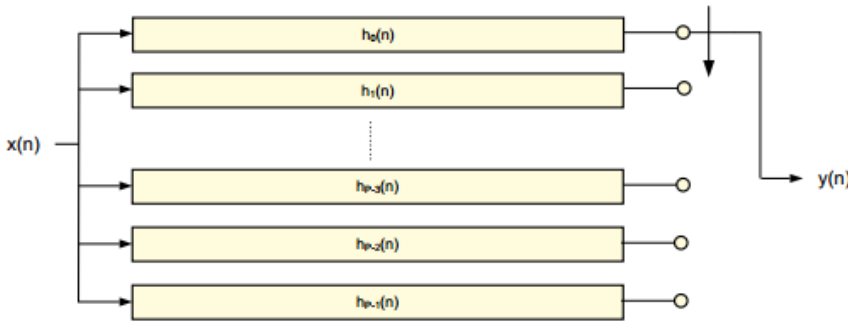


Figure 3-27: **1-to-*P* Polyphase Interpolator**

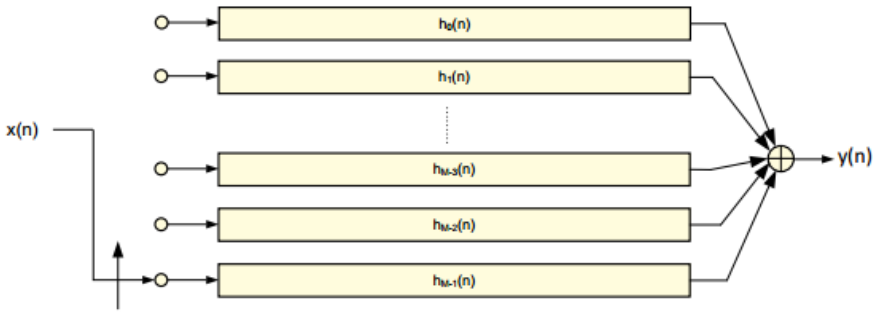
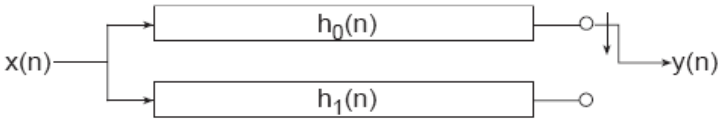


Figure 3-26: ***M*-to-1 Polyphase Decimating Filter**

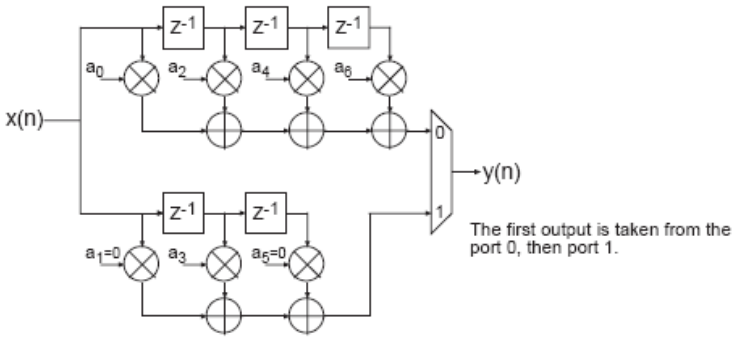
Halfband Interpolation filter

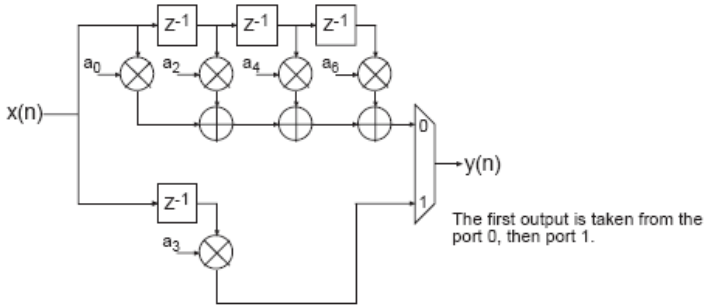
- **Halfband interpolator is a special case of a polyphase intepolator**



- **In halfband filter, every other coefficients are zero except 3 main taps**

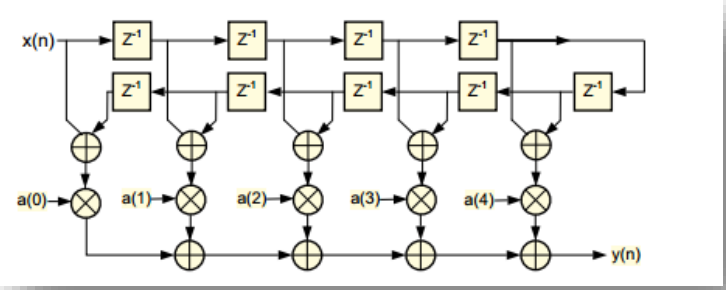
- **The figure below shows partitioning after a normal polyphase decomposition, and the right one takes advantage of zero coefficients**





fir.h – sym_class (1)

- " "II_GOAL", which is a desired initiation interval can be passed thru template parameter
- It is important to have inline the "process"
- Both coefficients and tap delay line (coeff and sr) are reshaped completely to make all the members available each clock cycle
- "MAC_preadd" takes advantage of symmetric coefficients by pre-adding the outputs of tap delay line




```
131 //_____________________________________________________________________
132 // single rate, symmetric fir classs
133 //
134 // - parameter:
135 //    -l_WHOLE:  number of taps
136 //    -l_SAMPLE: number of data sample
137 //    -II_GOAL:  initiation interval goal
138 //_____________________________________________________________________
139
140 template<int l_WHOLE, int l_SAMPLE, int II_GOAL>
141 class sym_class {
142
143 // use assert to check the template parameter
144
145
146 static const int odd    = l_WHOLE % 2;
147 static const int l_HALF = l_WHOLE/2;
148 static const int l_COEF = l_WHOLE/2 + odd;  // number of unique coefficients
149
150 DATA_T sr[l_WHOLE];
151 ACC_T  acc;
152 COEF_T coeff[l_COEF];
153
154 public:
155
156 //_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
157 // MAC engine
158 ACC_T MAC_preadd( DATA_T din0, DATA_T din1, COEF_T coef, ACC_T acc ) {
159
160     DATA2_T preadd = din0 + din1;
161     PROD_T  prod   = preadd * coef;
162     ACC_T   sum    = prod + acc;
163
164     return sum;
165 };
166
167
168 //_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
169 // filter
170 void process ( DATA_T din, DATA_T* dout) {
171
172 #pragma HLS INLINE
173
174 // using 'factor' instead of 'complete' uses BRAM instead of FF
175 #pragma HLS array_reshape variable=sr  complete
176 #pragma HLS array_reshape variable=coeff complete dim=0
177
178 acc = 0;
179
180 LOOP_MAC:
181 for (int i=0; i<l_HALF; i++) {
182     acc = MAC_preadd (sr[i], sr[l_WHOLE-1-i], coeff[i], acc);
183 }
184
185 if (odd)
186     acc = MAC_preadd (sr[l_HALF], 0, coeff[l_HALF], acc);
187
188 LOOP_SR:  for (int i=l_WHOLE-1; i>0 ; i--) {
189             sr[i] = sr[i-1];
190 }
191
192 sr[0] = din;
193
194 *dout = acc;
195
196 }
197
```

fir.h – sym_class (2)

- "process_frame" process a packet of data, by calling "process" member
- "II_GOAL" is used in this pipeline directive
- A constructor is used to initialize the coefficients

```
199 //_____________________________________________________
200 // filter frame
201 void process_frame(DATA_T din[l_SAMPLE], DATA_T dout[l_SAMPLE]) {
202
203    DATA_T tout;
204
205    L_FRAME: for (int i=0; i<l_SAMPLE; i++ ) {
206    #pragma HLS pipeline II=II_GOAL rewind
207             process (din[i], &tout);
208             dout[i] = tout;
209         }
210
211 }
212
213
214 //_____________________________________________________
215 // initialize coeff
216 void init(const COEF_T cin[l_COEF]) {
217 L_COEFF_INIT : for (int i=0; i<l_COEF;i++)
218                  coeff[i] = cin[i];
219 }
220
221 //constructor
222 sym_class(const COEF_T cin[l_COEF]) {
223     init(cin);
224 }
225
226 //destructor
227 ~sym_class(void) {
228 }
229
230
231
232 }; // sym_class
233
```

Single rate symmetric FIR

- single_rate_sym_v4_noframe: uses the sym_class object and calls “process” member
- 65 symmetric tap, II = 1. DSP48 count of 33 makes sense
- Fmax = 208 MHz targeting Zynq -1 device.

```
1  #include "fir_sym.h"
2
3  void fir_sym ( DATA_T din[L_INPUT], DATA_T dout[L_OUTPUT] ) {
4
5  #pragma HLS interface ap_fifo depth=L_INPUT  port=din
6  #pragma HLS interface ap_fifo depth=L_OUTPUT port=dout
7
8
9  static const COEF_T cin[L_WHOLE] = {
10     #include "coef_65tap.inc"
11 };
12
13 static sym_class<L_WHOLE, L_INPUT, II_GOAL> f0(cin);
14
15 DATA_T sout;
16
17 L1:
18 for (int i=0;i<L_INPUT;i++) {
19 #pragma HLS pipeline II=II_GOAL rewind
20
21     f0.process(din[i], &sout);
22     dout[i] = sout;
23 }
24
25 }
```

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	5.00	4.28	0.63

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
265	266	100	100	loop rewind

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	0	11
FIFO	-	-	-	-
Instance	-	33	5221	5622
Memory	-	-	-	-
Multiplexer	-	-	-	17
Register	-	-	218	6
Total	0	33	5439	5656
Available	280	220	106400	53200
Utilization (%)	0	15	5	10

Device target: xc7z020clg484-1

Implementation tool: Xilinx Vivado v.2015.2

Resource Usage

	Verilog
SLICE	2044
LUT	4310
FF	4889
DSP	33
BRAM	0
SRL	3449

Final Timing

	Verilog
CP required	5.000
CP achieved	4.827

Single rate non-symmetric FIR

- Single_rate_nosym_v4: using the “nosym_class” and “process”
- Fmax=331 MHz targeting K7 -1
- There is 8 pipeline stages going thru each MAC (4 is due to multiplication), thus, making the total latency = 8*number of taps. Can we do better?

	Operation\Control Step	C0	C1	C2	C3	C4	C5	C6	C7
1	acc_V_read(read)								
2	coef_V_read(read)								
3	din_V_read(read)								
4	p_Val2_s(*)								
5	r(icmp)								
6	r_i_i(|)								
7	qb_assign_1(&)								
8	p_Val2_1(+)								
9	sum_V(+)								

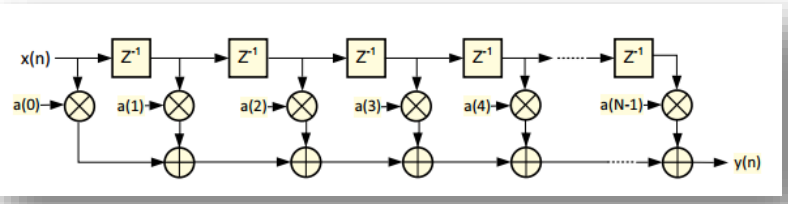
Performance | Resource

Properties | C Source

File: D:\design_QAM\FIR\single_rate_nosym_v4\fir.h

```
45 public:
46
47   //_____________________________
48   // MAC engine
49
50   ACC_T MAC( DATA_T din, COEF_T coef, ACC_T  acc ) {
51
52     PROD_T prod  = din * coef;
53     ACC_T  sum   = prod + acc;
54     return sum;
55   };
```

```
3  //_____________________________
4  void fir_nosym ( DATA_T din[L_INPUT], DATA_T dout[L_OUTPUT] ) {
5
6  #pragma HLS interface ap_fifo depth=L_INPUT  port=din
7  #pragma HLS interface ap_fifo depth=L_OUTPUT port=dout
8
9  static const COEF_T cin[L_WHOLE] = {
10     #include "coef_32tap.inc"
11 };
12 static nosym_class<L_WHOLE, L_INPUT, II_GOAL> ff(cin);
13
14 DATA_T sout;
15
16 L1:
17 for (int i=0;i<L_INPUT;i++) {
18 #pragma HLS pipeline II=II_GOAL rewind
19
20     ff.process(din[i], &sout);
21     dout[i] = sout;
22 }
23
24 }
```



Clock	Target	Estimated	Uncertainty
ap_clk	3.00	2.58	0.38

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
354	355	100	100	loop rewind

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	0	11
FIFO	-	-	-	-
Instance	-	32	3840	2272
Memory	-	-	-	-
Multiplexer	-	-	-	17
Register	-	-	2087	2320
Total	0	32	5927	4620
Available	650	600	202800	101400
Utilization (%)	0	5	2	4

Device target: xc7k160tfbg484-1

Implementation tool: Xilinx Vivado v.2015.2

Resource Usage

	VHDL
SLICE	1585
LUT	3499
FF	5680
DSP	32
BRAM	0
SRL	2647

Final Timing

	VHDL
CP required	3.000
CP achieved	3.025

Single rate halfband FIR

- **single_rate_hb_v3: Using “hb_class”**
- **23 tap total – there are 13 non zero coefficients in HB filter**
- **Fmax = 378 MHz, targeting K7 -1**

```
#include "fir_hb.h"

void fir_hb ( DATA_T din[L_INPUT], DATA_T dout[L_OUTPUT] ) {

#pragma HLS interface ap_fifo depth=L_INPUT  port=din
#pragma HLS interface ap_fifo depth=L_OUTPUT port=dout

static const COEF_T cin[L_NONZERO] = {
    #include "b1_nonzero.inc"
};

static hb_class<L_WHOLE, L_INPUT, II_GOAL> f0(cin);

DATA_T sout;

L1: for (int i=0;i<L_INPUT;i++) {
#pragma HLS pipeline II=II_GOAL rewind

    f0.process(din[i], &sout);
    dout[i] = sout;
}

}
```

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	3.00	3.70	0.38

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
105	106	100	100	loop rewind

Detail

Instance

Loop

```
Implementation tool: Xilinx Vivado v.2015.2
Device target:       xc7k160tfbg484-1
Report date:         Thu Sep 03 16:15:37 -0400 2015

#=== Resource usage ===
SLICE:          179
LUT:            221
FF:             754
DSP:              5
BRAM:             0
SRL:             15
#=== Final timing ===
CP required:    3.000
CP achieved:    2.641
```

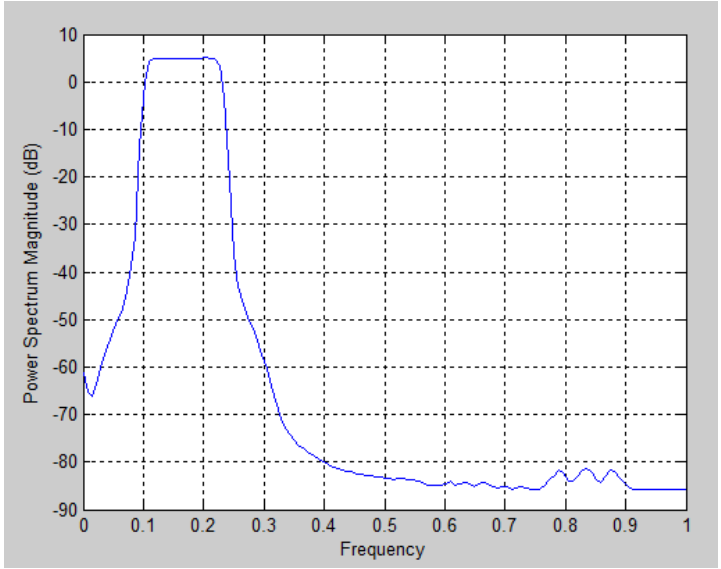
Utilization Estimates

Summary

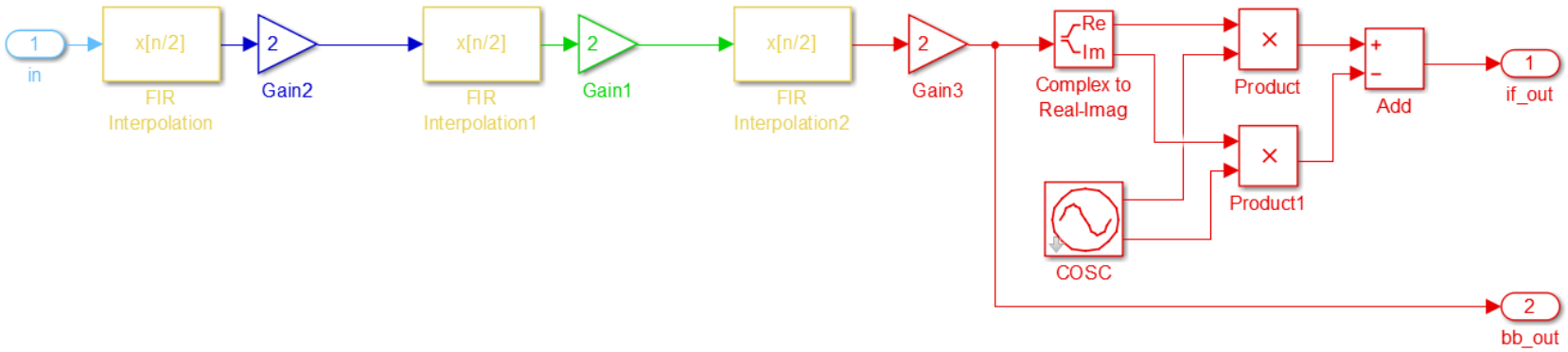
Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	0	239
FIFO	-	-	-	-
Instance	-	5	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	17
Register	-	-	750	27
Total	0	5	750	283
Available	650	600	202800	101400
Utilization (%)	0	~0	~0	~0

Digital Upconverter (DUC)

- **DUC design consist of:**
  - SRRC filter
  - 3 stage of interpolating filter
  - Digital Direct Synthesizer (DDS)
  - mixer
- **Shown on the right is an example of modulated DUC**

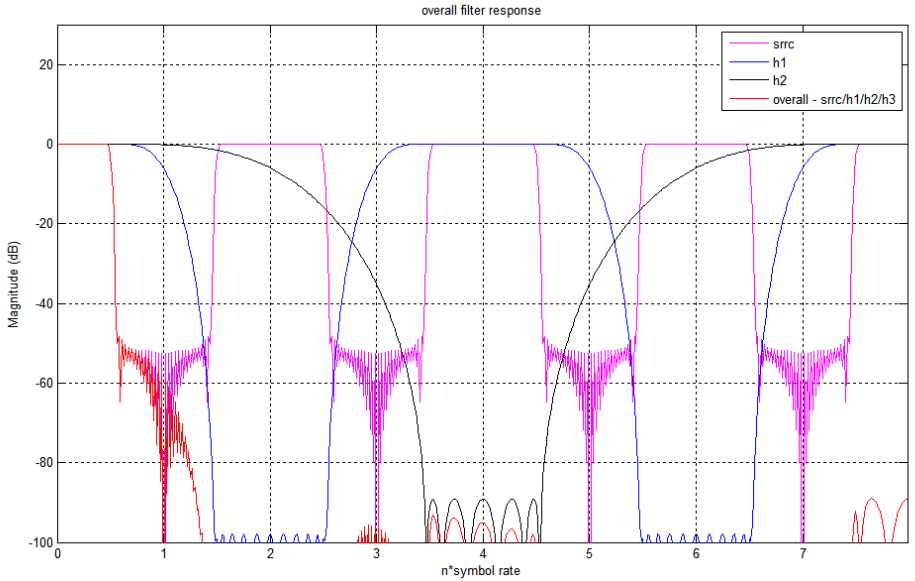


Digial Upconverter



DUC: Overall filter response

- **The overall response of the SRRC and 3 stage filters of the DUC is shown:**
- **The filter specs:**
  - SRRC: 64 taps, interpolate factor=2
  - Filter stages in DUC: consists of 3 stage halfband filters with interpolate factor = 2 each
  - The data rate at the output of the DUC is 16 times the symbol rate (the input rate of the SRRC filter)
- **The interpolated signal at the output of 3 stage HB filter is modulated to the IF frequency by using mixer and DDS (direct digital synthesizer)**



DUC: code structure

- **duc_2ch.cpp: contains the top level module called duc_2ch, which includes the following modules:**
  - duc_2ch_class (calling) -> duc_class -> various FIR classes
  - dds_frame: generates an array of sine/cosine output. It calls dds module
  - mixer: complex multiply duc_2ch output with dds_frame output

```
1  #include "duc_2ch.h"
2
3  #if 1
4  //_____________________________________________________________
5  void duc_2ch ( DATA_T din_i[L_INPUT],
6                 DATA_T din_q[L_INPUT],
7                 DATA_T dout[L_OUTPUT],
8                 incr_t incr ) {
9
10 //#pragma HLS interface ap_fifo depth=L_INPUT  port=din_i
11 //#pragma HLS interface ap_fifo depth=L_INPUT  port=din_q
12 //#pragma HLS interface ap_fifo depth=L_OUTPUT port=dout_i
13 //#pragma HLS interface ap_fifo depth=L_OUTPUT port=dout_q
14
15 //#pragma HLS interface ap_ctrl_none port=return  // to avoid bubble
16
17 //#pragma HLS interface s_axilite port=incr
18 #pragma HLS interface ap_stable port=incr
19
20 #pragma HLS interface axis depth=300  port=din_i
21 #pragma HLS interface axis depth=300  port=din_q
22 #pragma HLS interface axis depth=4800 port=dout
23
24 #pragma HLS dataflow
25
26 static duc_2ch_class<L_INPUT> f0;
27
28 DATA_T dout_i[L_OUTPUT];
29 DATA_T dout_q[L_OUTPUT];
30 dds_t  dds_cos[L_OUTPUT];
31 dds_t  dds_sin[L_OUTPUT];
32
33 f0.process(din_i, dout_i, din_q, dout_q);
34 dds_frame(incr, dds_cos, dds_sin);
35 mixer(dds_cos, dds_sin, dout_i, dout_q, dout);
36
37 }
38
39 #endif
40
```

DUC synthesis results

- **HLS synthesis results is shown**

General Information

Date:	Mon Mar 14 14:55:44 2016
Version:	2016.1 (Build 1483152 on Wed Feb 17 00:03:22 AM 2016)
Project:	proj
Solution:	sol1
Product family:	kintex7
Target device:	xc7k160tfbg484-1

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	3.00	3.70	0.38

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
3376	3377	3200	3200	dataflow

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	-	-
FIFO	0	-	70	328
Instance	2	30	7936	4733
Memory	-	-	-	-
Multiplexer	-	-	-	2
Register	-	-	20	-
Total	2	30	8026	5063
Available	650	600	202800	101400
Utilization (%)	~0	5	3	4

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
din_i_V_TDATA	in	24	axis	din_i_V	pointer
din_i_V_TVALID	in	1	axis	din_i_V	pointer
din_i_V_TREADY	out	1	axis	din_i_V	pointer
din_q_V_TDATA	in	24	axis	din_q_V	pointer
din_q_V_TVALID	in	1	axis	din_q_V	pointer
din_q_V_TREADY	out	1	axis	din_q_V	pointer
dout_V_TDATA	out	24	axis	dout_V	pointer
dout_V_TVALID	out	1	axis	dout_V	pointer
dout_V_TREADY	in	1	axis	dout_V	pointer
incr_V	in	32	ap_stable	incr_V	scalar
ap_clk	in	1	ap_ctrl_hs	duc_2ch	return value
ap_rst_n	in	1	ap_ctrl_hs	duc_2ch	return value
ap_start	in	1	ap_ctrl_hs	duc_2ch	return value
ap_done	out	1	ap_ctrl_hs	duc_2ch	return value
ap_idle	out	1	ap_ctrl_hs	duc_2ch	return value
ap_ready	out	1	ap_ctrl_hs	duc_2ch	return value

DUC implementation results

- **Vivado implementation results is shown**
- **Fmax = 313 MHz targeting K7 -1**

Export Report for 'duc_2ch'

General Information

Report date:	Mon Mar 14 16:55:06 -0400 2016
Project:	proj
Solution:	sol1
Device target:	xc7k160tfbg484-1
Implementation tool:	Xilinx Vivado v.2016.1.0

Resource Usage

	Verilog
SLICE	1716
LUT	3650
FF	7110
DSP	30
BRAM	2
SRL	702

Final Timing

	Verilog
CP required	3.000
CP achieved	3.194

Timing not met

Reference

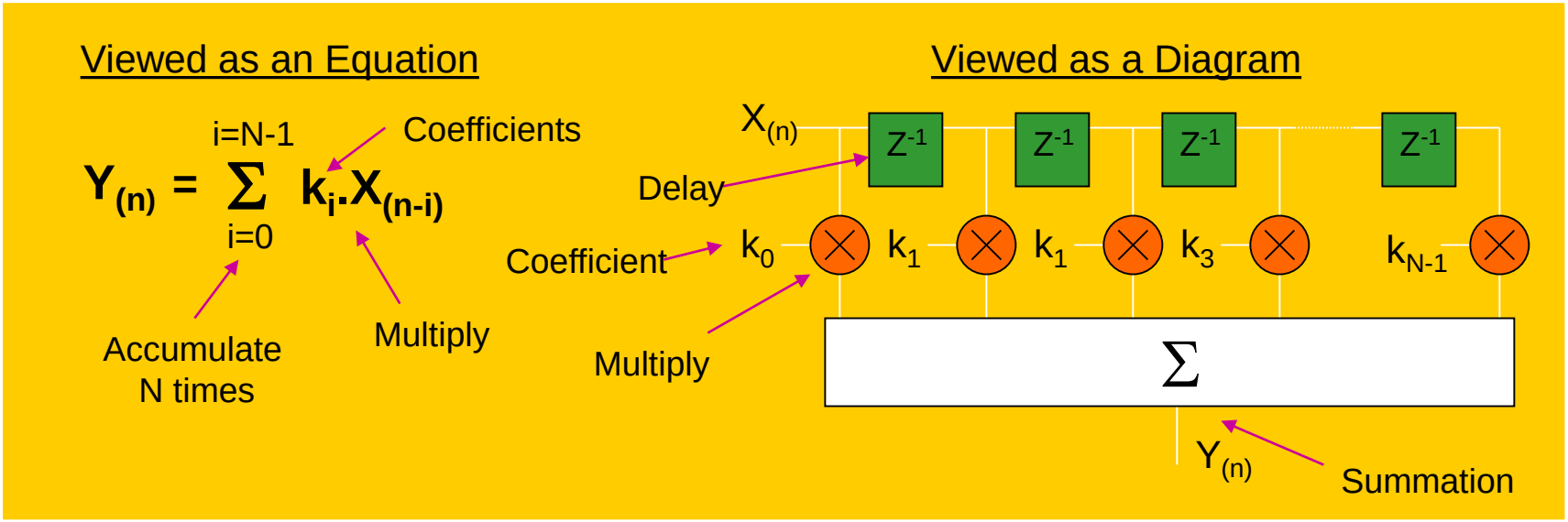
1. UG901, HLS User's Guide

Backup

.

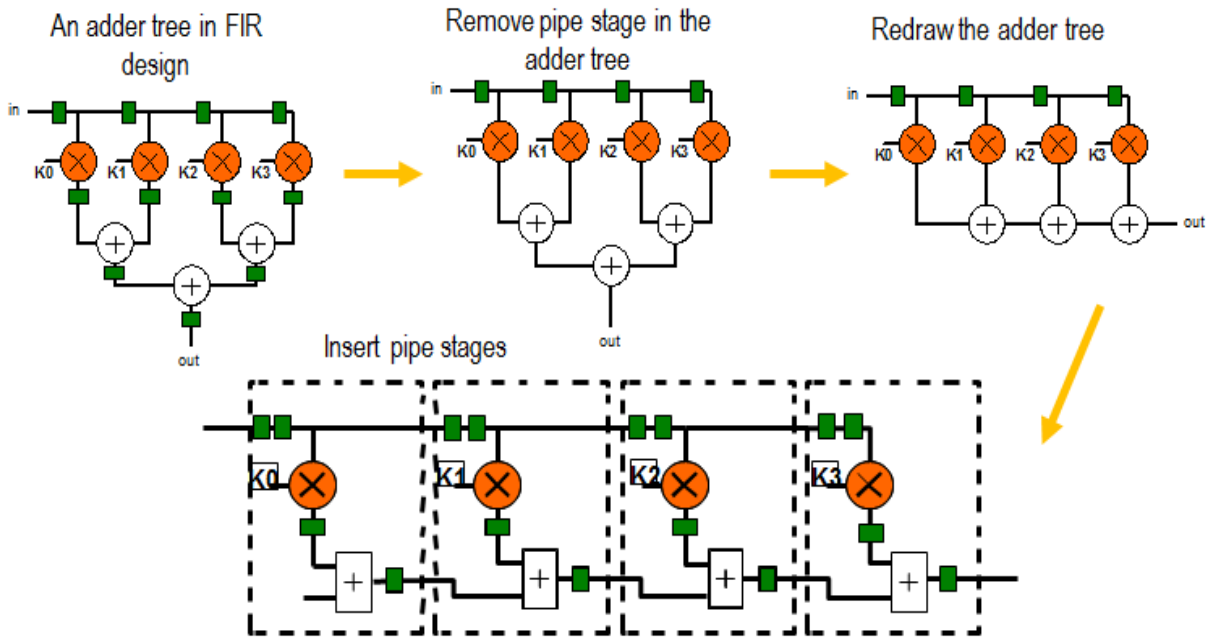
FIR Filter Basic Structure

- **FIR Filter is probably the most common DSP function implemented FPGA:**
- **Key criteria for selecting FIR implementation:**
  - **filter type, order, coefficient type**
  - **data rate, clock rate**
  - **latency**
  - **number of channels**



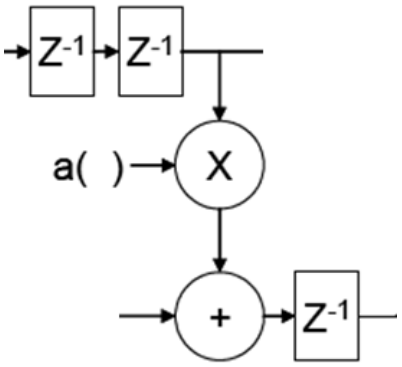
Systolic FIR structure

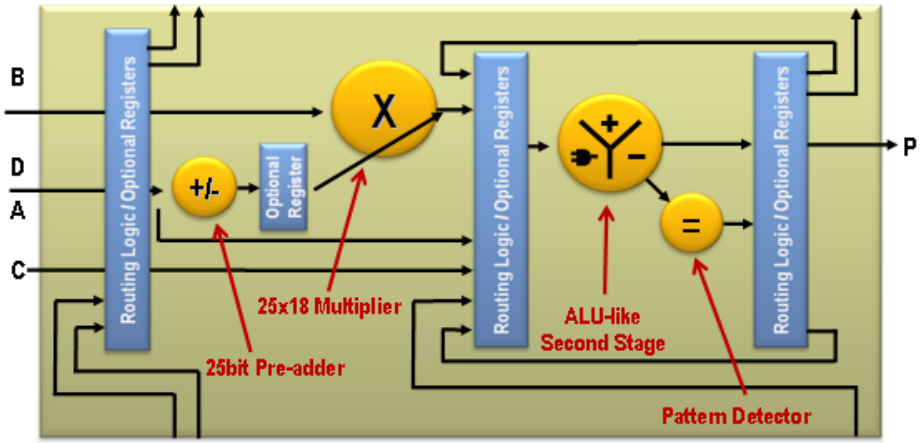
- The adder structure in FIR filter can be converted to systolic structure, and that became the basis of DSP48. The filters implemented this way can run at the Fmax speed of DSP48, which can go up to 741 MHz.



Multiply and Accumulate using DSP48x

- The most essential element in FIR implementation is multiply and accumulate (MAC) function
- The MAC function is mapped into DSP48x hard macro
- The multipliers in DSP48 can be 18x18 (V4, S3DSP) or 18x25 (V5,V6), and the accumulator is 48 bit wide
- A rule of thumb: DSP48x can run about 400-750 MHz in V4/5/6/7 (depending on the speed grade) and about half that speed in S3DSP and S6/A7

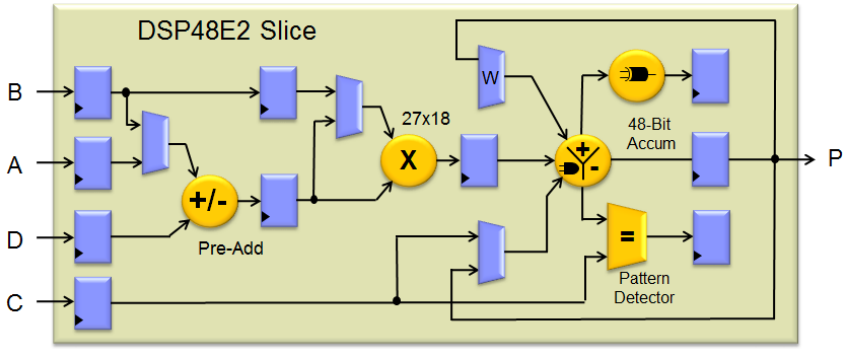




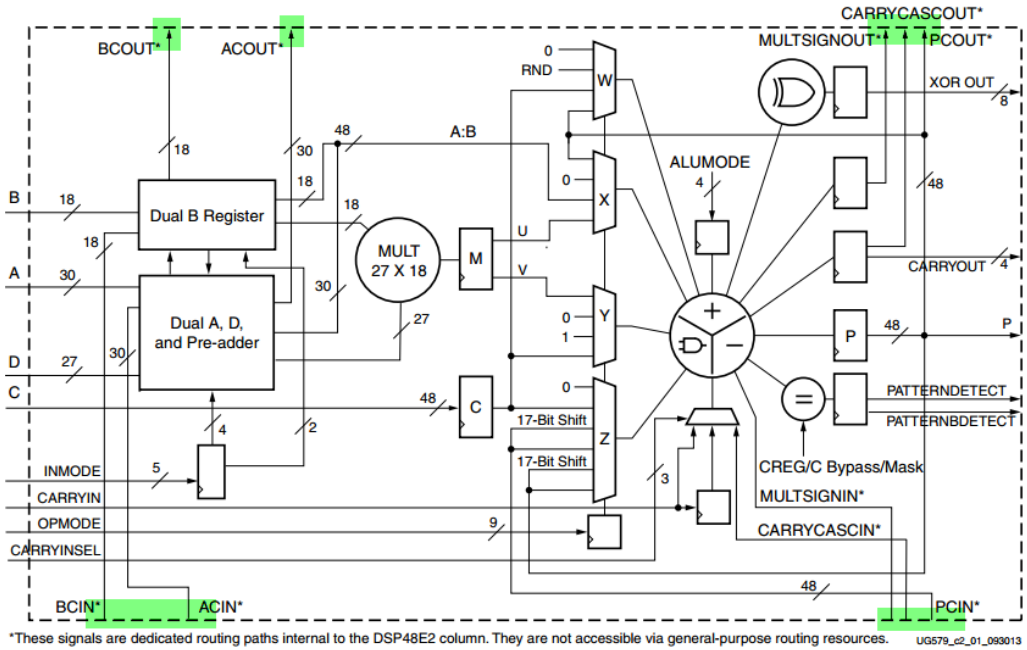
Power Consumption Benefits	Performance Benefits	Cost Benefits
▪ Lowest power operation of any FPGA solution ▪ 1.23mW/100Mz at 38% toggle rate	▪ 600MHz operations for any DSP operation including large filters ▪ ~1.2 TeraMACC in a single device	▪ Hardened pre-adder and adder cascade saves significant logic resources ▪ Logic functions can be mapped into DSP blocks

DSP48E2

- DSP48 is highly integrated multiplier accumulator structure. This solves the inherent problem associated with adder tree structure in FIR filter implementation, which are irregular structure, difficulty in routing, sensitivity to the length of filter.

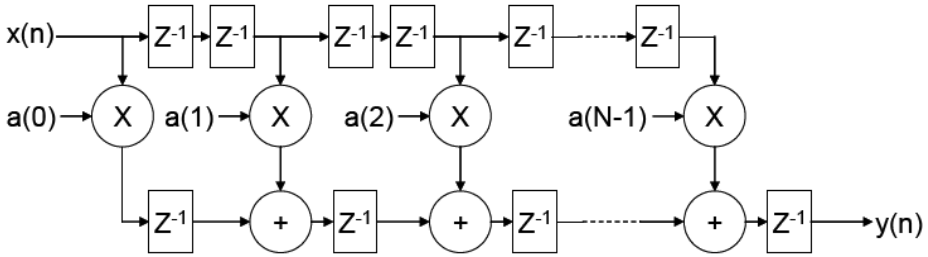


- IOs highlighted in green colored use designated routing columns

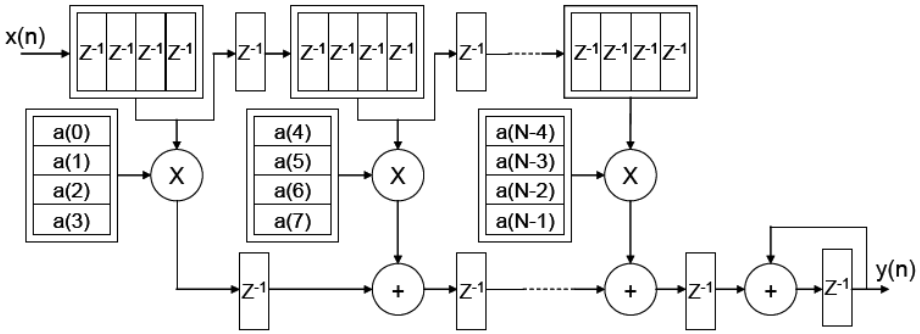


Systolic Multiply-Accumulate

- Most common filter structure
- A pipelined Direct-Form FIR.



- A multi-MAC implementation of Direct-Form FIR

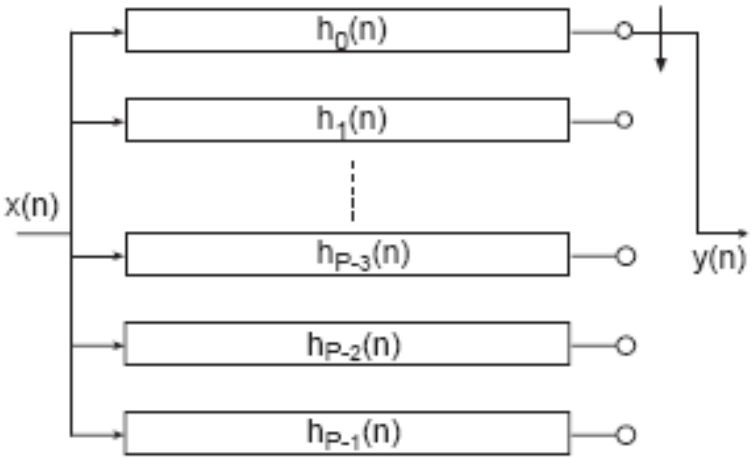


Polyphase Decomposition in Interpolator

- In an interpolation filter, the filter coefficients are partitioned into M polyphase sub-filters (M=interpolation ratio) as follows:

A set of N prototype filter coefficients $a_0, a_1, .., a_{N-1}$ are mapped to the M polyphase sub-filters $h_0(n), h_1(n), .., h_{M-1}(n)$ according to Equation 2.

$$h_i(n) = a(i + Mr) \quad i = 0, 1, .., M-1$$ r = 0,1,...N/M -1



P = M

Keeping Symmetry in Interpolating filter

- **Once symmetric filter is decomposed into polyphase subfilter, it might not be symmetric anymore (there are some exceptions, ex, odd length symmetric filter, and interpolate by 2 case)**
- **There is a simple technique to make the subfilters symmetric**
- **FIRCompiler uses this technique; thus, saving, the MAC resource by factor of 2**
- **The following examples shows this technique for a 15 tap interpolate by 3 filter. Notice h0 become even symmetric, and h2 becomes odd symmetric**

a , b , c , d , e , f, g , h , g , f , e , d , c , b , a

Produce the following subfilters:

h_0 = a , d , g , f , c

h_1 = b , e , h , e , b

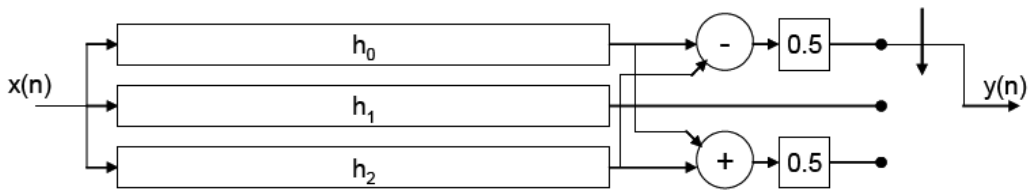
h_2 = c , f , g , d , a

Subfilter h_0 and h_2 are not symmetric. Applying the symmetric pairs technique produces the following subfilters:

h0 = a+c, d+f, 2*g, f+d, c+a

h1 = b,e,h,e,b

h2 = c-a, f-d, 0, d-f, a-c



- **Reference: Mou, Zhi-Jian, Symmetry Exploitation in Digital Interpolators/Decimators, IEEE Transactions on Signal Processing, Vol. 44 No. 10, Oct. 1996**