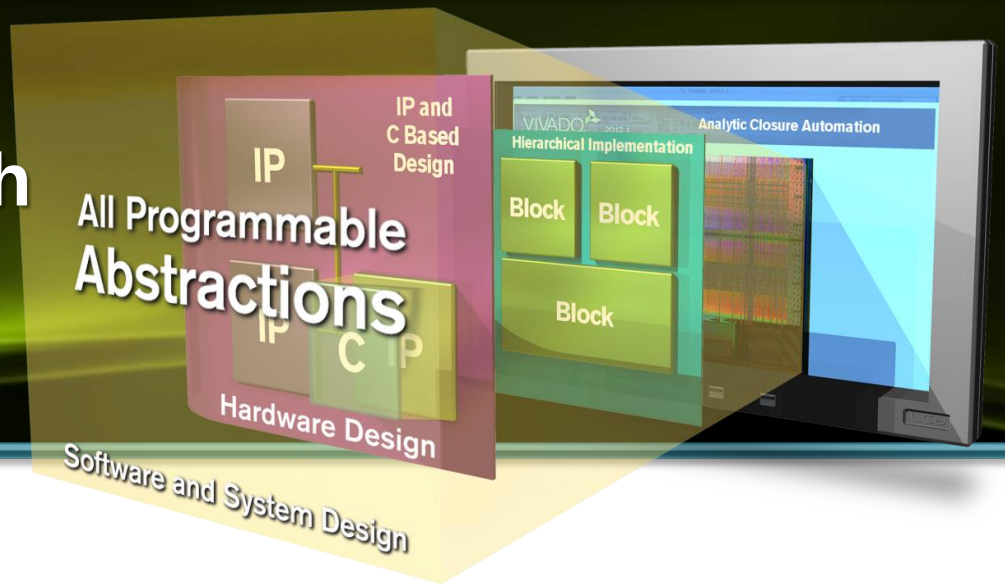
Getting started with with Vivado HLS: FIR filtering



Daniele Bagni
DSP Specialist for EMEA
daniele.bagni@xilinx.com

Outlines

- **HLS concepts: Initialization Interval and Latency**
- **FIR filter design: the C++ code**
- **FIR filter design: optimization directives and related performance**
  - Solution0: baseline
  - Solution1: directives PIPELINE II=x and ALLOCATION (this last one to limit the amount of mult operations)
  - Solution2: directive UNROLL to unroll the loop by x times
  - Solution3: directive LATENCY to limit loop iteration latency between min and max clock cycles
- **FIR filter design: performance summary**

Outlines

- **HLS concepts: Initialization Interval and Latency**
- **FIR filter design: the C++ code**
- **FIR filter design: optimization directives and related performance**
  - Solution0: baseline
  - Solution1: directives PIPELINE II=x and ALLOCATION (this last one to limit the amount of mult operations)
  - Solution2: directive UNROLL to unroll the loop by x times
  - Solution3: directive LATENCY to limit loop iteration latency between min and max clock cycles
- **FIR filter design: performance summary**

Rationale

- Using a simple N=16 taps FIR filter modelled in C++ to illustrate few High Level Synthesis (HLS) directives that affect the effective output data rate by changing the Initialization Interval and therefore the Latency.

- "The **Initiation Interval**, often called the Interval or the **II**, is the number of clock cycles between when the task can start to accept new input data" (from UG902)

- "**Latency** is the number of clock cycles it takes to produce an output value" (from UG902)

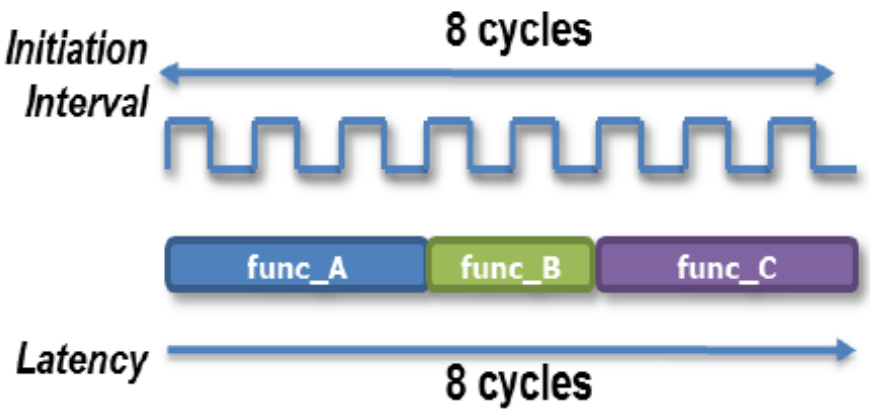
Initialization Interval and Latency 1/6

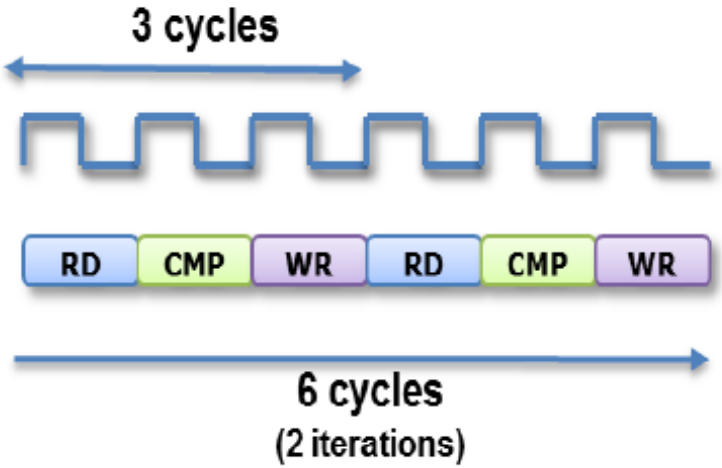
Function Level

```
void top (a,b,c,d) {


  func_A(a,b,i1[N]);
  func_B(c,i1[N],i2[N]);
  func_C(i2[N],d)


}
```

Loop level

```
void sub_func(...) {


 for (i=1;i>=0;i--) {
      op_Read;
      op_Compute;
      op_Write;
  }


}
```





XILINX ALL PROGRAMMABLE.

Initialization Interval and Latency 2/6

- **Latency:**
  - In the function example the latency of the design is 8 clock cycles: it takes 8 clock cycles from the start of function top until and output can be written.
  - In the loop example, it takes 3 clock cycles to execute each iteration of the loop and a total latency of 6 clock cycles to execute all loop iterations.
- **Initialization Interval:**
  - In the function example, func A cannot be executed again until func C is finished. Because it takes 8 clock cycles until func C completes, the II of function top is 8 clock cycles: it runs 8 clock cycles before it can start processing new input data.
  - The loop example takes 6 clock cycles to execute all loop transactions, however it can accept a new input every 3 clock cycle and so the loop II is 3.

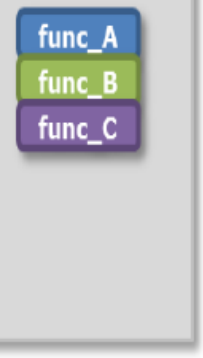
Initialization Interval and Latency 3/6

- In both these examples, everything operates sequentially, since there is no concurrency, the latency and II are the same for both tasks.
- Pipelining optimization enables a sequential C description to be implemented as a concurrent hardware implementation.
- Next Figure shows the result when High-Level Synthesis is used to pipeline the sub-functions inside function top and the operations in the loop.
- With pipelining, both tasks can achieve higher performance. At the function level, dataflow optimization allows the sub-functions (A, B and C) to execute as soon as data is available.

Initialization Interval and Latency 4/6

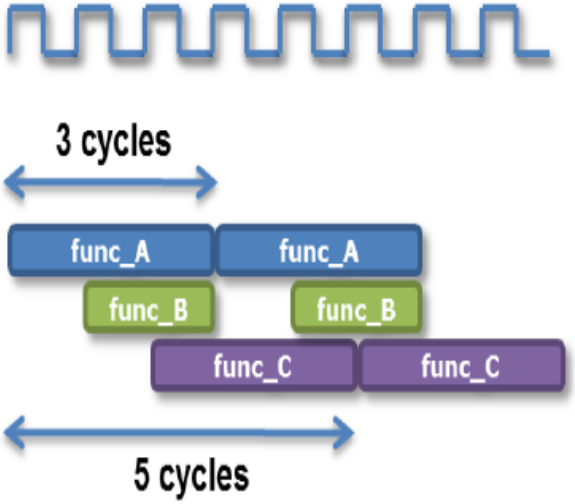
Function Level

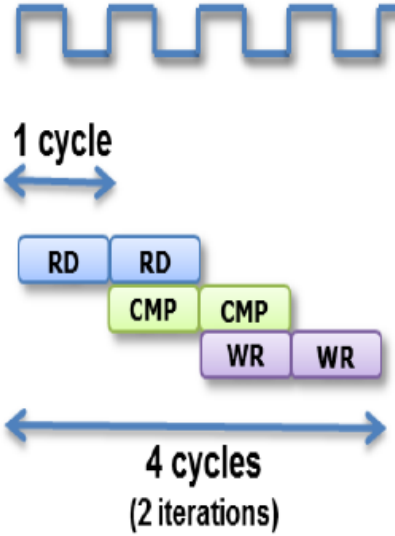
```
void top (a,b,c,d) {

  func_A(a,b,i1[N]);
  func_B(c,i1[N],i2[N]);
  func_C(i2[N],d)

}
```



Loop level

```
void sub_func(...) {

 for (i=1;i>=0;i--) {
      op_Read;
      op_Compute;
      op_Write;
  }

}
```





XILINX ALL PROGRAMMABLE.

Initialization Interval and Latency 5/6

- If function A starts to produce data before it has completely finished, func B can start accepting that data as soon as it is ready. It does not have to wait for func A to complete. Similarly, func C can start to execute as soon as data becomes available from func B. More importantly, func A can start the next transaction before func C has completed the current transaction.
- For the function top original design the latency and II were both 8 clock cycles. In this pipelined version of the same design, the latency is only 5 clock cycles and the II is 3 clock cycles: the pipelined design outputs data in less time and can accept data at a faster rate (almost 3X).

Initialization Interval and Latency 6/6

- Performance improvements can be seen also for a pipelined implementation of the loop example. As soon as the read operation has completed, the next read operation can start.
- The performance of the loop can improve from an II=3 to an II=1 and by performing overlapped loop iterations, the overall loop latency can be reduced to 4 (even though the latency of each iteration remains at 3).

Outlines

- HLS concepts: Initialization Interval and Latency
- FIR filter design: the C++ code
- FIR filter design: optimization directives and related performance
  - Solution0: baseline
  - Solution1: directives PIPELINE II=x and ALLOCATION (this last one to limit the amount of mult operations)
  - Solution2: directive UNROLL to unroll the loop by x times
  - Solution3: directive LATENCY to limit loop iteration latency between min and max clock cycles
- FIR filter design: performance summary

FIR filter design: fir.cpp and fir.h design files

fir.cpp | fir_test.cpp

(Global Scope) | fir_filter(inp_data_t x, coef_t c[N])

```cpp
#include "fir.h"

out_data_t fir_filter (  inp_data_t x,   coef_t c[N])
{

  static inp_data_t shift_reg[N];

  acc_t acc = 0;
  acc_t mult;
  out_data_t y;

  signed char i;

  Shift_Accum_Loop: for (i=N-1;i>=0;i--)
  {

     if (i==0)
     {

       shift_reg[0]=x;
     }
     else
     {
            shift_reg[i]=shift_reg[i-1];

     }
     mult = shift_reg[i]*c[i];
     acc  = acc + mult;

  }

  y = (out_data_t) acc;

  return y;

}
```

fir.h

(Global Scope)

```cpp
#ifndef _H_FIR_H_
#define _H_FIR_H_

#define N   16

#define SAMPLES 1024




#define DB_FIXED_POINT


#ifdef DB_FIXED_POINT
#include "ap_fixed.h"

typedef ap_fixed<18,2>  coef_t;
typedef ap_fixed<48,12> out_data_t;
typedef ap_fixed<18,2>  inp_data_t;
typedef ap_fixed<48,12> acc_t;

#else //FULLY ANSI C
typedef           int coef_t;
typedef long long int out_data_t;
typedef           int inp_data_t;
typedef long long int acc_t;

#endif




out_data_t fir_filter ( inp_data_t x, coef_t c[N] );

#endif // _H_FIR_H_
```


XILINX ALL PROGRAMMABLE.

FIR filter design: fir_test.cpp testbench file

```c
#include "fir.h"

int main (void)
{

  inp_data_t signal[SAMPLES];
  out_data_t output[SAMPLES], reference[SAMPLES];
  coef_t taps[N];

  FILE         *fp1, *fp2, *fp3;
  int i, ret_value;
  float val1, val2;
  float diff, tot_diff;
  int val3;

  tot_diff = 0;

  // LOAD FILTER COEFFICIENTS
  fp1=fopen("./data/fir_coeff.dat","r");
  for (i=0;i<N;i++)
  {
      fscanf(fp1,"%f\n", &val1);
      taps[i] = (coef_t) val1;
      fprintf(stdout,"taps[%4d]=%10.5f\n", i, taps[i].to_double());
  }
  fclose(fp1);

  // LOAD INPUT DATA  AND REFERENCE RESULTS
  fp2=fopen("./data/input.dat","r");
  fp3=fopen("./data/ref_res.dat","r");

  for (i=0;i<SAMPLES;i++)
  {
    fscanf(fp2,"%f\n", &val1);
    signal[i]    = (inp_data_t) val1;
    fscanf(fp3,"%f\n", &val2);
    reference[i] = (out_data_t) val2;

  }
  fclose(fp2);
  fclose(fp3);
```

```c
  fclose(fp2);
  fclose(fp3);

// CALL DESIGN UNDER TEST
  for (i=0;i<SAMPLES;i++)
  {
      output[i] = fir_filter(signal[i], taps);
  }

  // WRITE OUTPUT RESULTS
  fp1=fopen("./data/out_res.dat","w");
  for (i=0;i<SAMPLES;i++)
  {
       fprintf(fp1,"%10.5f\n", output[i].to_double());
  }
  fclose(fp1);

  // CHECK RESULTS
  for (i=0;i<SAMPLES;i++)
  {
       diff = output[i].to_double() - reference[i].to_double();
       if (i<64) fprintf(stdout,"output[%4d]=%10.5f \t reference[%4d]=%10.5f\n",
                                i, output[i].to_double(), i, reference[i].to_double() );
       diff = fabs(diff);
       tot_diff += diff;
  }
  fprintf(stdout, "TOTAL ERROR =%f\n",tot_diff);

  if (tot_diff < 1.0)
  {
      fprintf(stdout, "\nTEST PASSED!\n");
      ret_value =0;
  }
  else
  {
      fprintf(stdout, "\nTEST FAILED!\n");
      ret_value =1;
  }

  return ret_value;

}
```

Outlines

- HLS concepts: Initialization Interval and Latency
- FIR filter design: the C++ code
- **FIR filter design: optimization directives and related performance**
  - Solution0: baseline
  - Solution1: directives PIPELINE II=x and ALLOCATION (this last one to limit the amount of mult operations)
  - Solution2: directive UNROLL to unroll the loop by x times
  - Solution3: directive LATENCY to limit loop iteration latency between min and max clock cycles
- FIR filter design: performance summary

HLS solution0: first trials

- solution0_a: baseline
- solution0_b: Loop pipelining
- solution_c: as solution_b + partitioning the shift_reg array
- solution_d: as solution_c + loop unrolling

Performance Estimates

Timing (ns)

Clock		solution0	solution0_b	solution0_c	solution0_d	solution0_e
default	Target	10.00	10.00	10.00	10.00	10.00
	Estimated	6.38	6.38	8.62	8.28	8.28

Latency (clock cycles)

		solution0	solution0_b	solution0_c	solution0_d	solution0_e
Latency	min	97	37	20	11	2
	max	97	37	20	11	2
Interval	min	98	38	21	12	3
	max	98	38	21	12	3

Utilization Estimates

	solution0	solution0_b	solution0_c	solution0_d	solution0_e
BRAM_18K	0	0	0	0	0
DSP48E	1	1	1	16	16
FF	202	171	466	1084	927
LUT	133	142	269	648	627

Outlines

- HLS concepts: Initialization Interval and Latency
- FIR filter design: the C++ code
- **FIR filter design: optimization directives and related performance**
  - Solution0: directive LATENCY to limit loop iteration latency between min and max clock cycles
  - Solution1: directives PIPELINE II=x and ALLOCATION (this last one to limit the amount of mult operations)
  - Solution2: directive UNROLL to unroll the loop by x times
- FIR filter design: performance summary

HLS solution1: II=1 at top level

- **solution1_II1:**
  - II=1 at top level
- **solution1_II2:**
  - II=2 at top level + allocation mul=8
- **solution1_II4:**
  - II=4 at top level + allocation mul=4
- **solution1_II8:**
  - II=8 at top level + allocation mul=2
- **solution1_II16:**
  - II=16 at top level + allocation mul=1

Performance Estimates

Timing (ns)

Clock		solution1_II1	solution1_II2	solution1_II4	solution1_II8	solution1_II16
default	Target	10.00	10.00	-	10.00	10.00
	Estimated	8.28	8.28	-	8.28	8.28
ap_clk	Target	-	-	10.00	-	-
	Estimated	-	-	8.28	-	-

Latency (clock cycles)

		solution1_II1	solution1_II2	solution1_II4	solution1_II8	solution1_II16
Latency	min	2	3	5	9	17
	max	2	3	5	9	17
Interval	min	1	2	4	8	16
	max	1	2	4	8	16

Utilization Estimates

	solution1_II1	solution1_II2	solution1_II4	solution1_II8	solution1_II16
BRAM_18K	0	0	0	0	0
DSP48E	16	8	4	2	1
FF	927	640	610	654	662
LUT	626	916	773	776	823

HLS solution1_II1: directives to achieve II=1

Vivado HLS - hls_sc_fir1ch_prj (C:\BagniD\V_HLS\multi_channel_FIR_filter\fir1ch\sc\hls_sc_fir1ch_prj)

Edit Project Solution Window Help

EN English (Ireland) Help

Debug Synthesis Analysis

Explorer

- hls_sc_fir1ch_prj
  - Includes
  - Source
    - fir.cpp
  - Test Bench
    - fir_test.cpp
    - data
  - solution0
  - solution0_lat10
  - solution0_lat20
  - **solution1_II1**
    - constraints
      - directives.tcl
      - script.tcl
    - csim
      - build
      - report
    - sim
      - autowrap
      - report
      - tv
      - verilog
      - wrapc
      - wrapc_pc
    - syn
      - report
      - systemc
      - verilog
      - vhdl
  - solution1_II16
  - solution1_II2
  - solution1_II4
  - solution1_II8

fir.cpp

```
1 #include "fir.h"
2
3 out_data_t fir_filter (  inp_data_t x,   coef_t c[N])
4 {
5
6   static inp_data_t shift_reg[N];
7
8   acc_t acc = 0;
9   acc_t mult;
10  out_data_t y;
11
12  signed char i;
13
14  Shift_Accum_Loop: for (i=N-1;i>=0;i--)
15  {
16
17     if (i==0)
18     {
19       //acc+=x*c[0];
20       shift_reg[0]=x;
21     }
22     else
23     {
24             shift_reg[i]=shift_reg[i-1];
25             //acc+=shift_reg[i]*c[i];
26     }
27     mult = shift_reg[i]*c[i];
28     acc  = acc + mult;
29
30  }
31
32  y = (out_data_t) acc;
33
34  return y;
35
36 }
37
```

Outline Directive

- fir_filter
  - HLS PIPELINE II=1
  - shift_reg
  - HLS ARRAY_PARTITION partition variable=shift_reg complete dim=1
  - x
  - HLS INTERFACE ap_none register port=x
  - c
  - HLS ARRAY_PARTITION partition variable=c complete dim=1
  - Shift_Accum_Loop

XILINX ALL PROGRAMMABLE.

HLS solution1_II1: RTL/C Cosimulation report

Vivado HLS - hls_sc_fir1ch_prj (C:\BagniD\seminar\HEFR_Fribourg_CH\3_FIR_Example\hls_sc_fir1

File Edit Project Solution Window Help

Explorer

- hls_sc_fir1ch_prj
  - Includes
  - Source
    - fir.cpp
  - Test Bench
  - solution0
  - solution0_b
  - solution0_c
  - solution0_d
  - solution0_e
  - ***solution1_II1***
    - constraints
      - directives.tcl
      - script.tcl
    - sim
    - syn
      - report
      - systemc
      - verilog
      - vhdl

Simulation(solution1_II1)

Cosimulation Report for 'fir_filter'

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	2	2	2	1	1	1

Export the report(.html) using the Export Wizard

XILINX ALL PROGRAMMABLE.

HLS solution1_II1: Export IP

- **Results after Place-And-Route**

Export Report for 'fir_filter'

General Information

Report date: Sun Jul 19 17:31:50 +0200 2015
Device target: xc7z020clg484-1
Implementation tool: Xilinx Vivado v.2015.1

Resource Usage

	Verilog
SLICE	156
LUT	566
FF	98
DSP	16
BRAM	0
SRL	10

Final Timing

	Verilog
CP required	10.000
CP achieved	9.580

HLS solution1_II2: directives to achieve II=2

vado HLS - hls_sc_fir1ch_prj (C:\BagniD\V_HLS\multi_channel_FIR_filter\fir1ch\sc\hls_sc_fir1ch_prj)

Edit Project Solution Window Help

EN English (Ireland) Help

bug Synthesis Analysis

plorer

- hls_sc_fir1ch_prj
  - Includes
  - Source
    - fir.cpp
  - Test Bench
    - fir_test.cpp
    - data
  - solution0
  - solution0_lat10
  - solution0_lat20
  - solution1_II1
  - solution1_II16
  - **solution1_II2**
    - constraints
      - directives.tcl
      - script.tcl
    - syn
      - report
      - systemc
      - verilog
      - vhdl
  - solution1_II4
  - solution1_II8
  - solution1_trial
  - solution2_full_pipeline
  - solution2_unroll16
  - solution2_unroll2
  - solution2_unroll4
  - solution2_unroll8

fir.cpp

```
1 #include "fir.h"
2
3 out_data_t fir_filter (  inp_data_t x,   coef_t c[N])
4 {
5
6   static inp_data_t shift_reg[N];
7
8   acc_t acc = 0;
9   acc_t mult;
10  out_data_t y;
11
12  signed char i;
13
14  Shift_Accum_Loop: for (i=N-1;i>=0;i--)
15  {
16
17     if (i==0)
18     {
19       //acc+=x*c[0];
20       shift_reg[0]=x;
21     }
22     else
23     {
24            shift_reg[i]=shift_reg[i-1];
25            //acc+=shift_reg[i]*c[i];
26     }
27     mult = shift_reg[i]*c[i];
28     acc  = acc + mult;
29
30  }
31
32  y = (out_data_t) acc;
33
34  return y;
35
36 }
```

Outline Directive

- fir_filter
  - % HLS PIPELINE II=2
  - % HLS ALLOCATION instances=mul limit=8 operation
  - shift_reg
  - % HLS ARRAY_PARTITION partition variable=shift_reg complete dim
  - x
  - % HLS INTERFACE ap_none register port=x
  - c
  - % HLS ARRAY_PARTITION partition variable=c complete dim=1
  - Shift_Accum_Loop

XILINX ALL PROGRAMMABLE.

HLS solution1_II4: directives to achieve II=4

Vivado HLS - hls_sc_fir1ch_prj (C:\BagniD\V_HLS\multi_channel_FIR_filter\fir1ch\sc\hls_sc_fir1ch_prj)

Edit Project Solution Window Help

EN English (Ireland) Help

Debug Synthesis Analysis

Explorer

- hls_sc_fir1ch_prj
  - Includes
  - Source
    - fir.cpp
  - Test Bench
    - fir_test.cpp
    - data
  - solution0
  - solution0_lat10
  - solution0_lat20
  - solution1_II1
  - solution1_II16
  - solution1_II2
  - **solution1_II4**
    - constraints
      - directives.tcl
      - script.tcl
    - syn
      - report
      - systemc
      - verilog
      - vhdl
  - solution1_II8
  - solution1_trial
  - solution2_full_pipeline
  - solution2_unroll16
  - solution2_unroll2
  - solution2_unroll4
  - solution2_unroll8

fir.cpp

```
1 #include "fir.h"
2
3 out_data_t fir_filter (  inp_data_t x,   coef_t c[N])
4 {
5
6   static inp_data_t shift_reg[N];
7
8   acc_t acc = 0;
9   acc_t mult;
10  out_data_t y;
11
12  signed char i;
13
14  Shift_Accum_Loop: for (i=N-1;i>=0;i--)
15  {
16
17     if (i==0)
18     {
19       //acc+=x*c[0];
20       shift_reg[0]=x;
21     }
22     else
23     {
24            shift_reg[i]=shift_reg[i-1];
25            //acc+=shift_reg[i]*c[i];
26     }
27     mult = shift_reg[i]*c[i];
28     acc  = acc + mult;
29
30  }
31
32  y = (out_data_t) acc;
33
34  return y;
35
36 }
```

Outline Directive

- fir_filter
  - HLS PIPELINE II=4
  - HLS ALLOCATION instances=mul limit=4 operation
  - shift_reg
  - HLS ARRAY_PARTITION partition variable=shift_reg complete dim=1
  - x
  - HLS INTERFACE ap_none register port=x
  - c
  - HLS ARRAY_PARTITION partition variable=c complete dim=1
  - Shift_Accum_Loop

XILINX ALL PROGRAMMABLE.

HLS solution1_II8: directives to achieve II=8

Vivado HLS - hls_sc_fir1ch_prj (C:\BagniD\V_HLS\multi_channel_FIR_filter\fir1ch\sc\hls_sc_fir1ch_prj)

File Edit Project Solution Window Help

EN English (Ireland) Help

Debug Synthesis Analysis

Explorer

- hls_sc_fir1ch_prj
  - Includes
  - Source
    - fir.cpp
  - Test Bench
    - fir_test.cpp
    - data
  - solution0
  - solution0_lat10
  - solution0_lat20
  - solution1_II1
  - solution1_II16
  - solution1_II2
  - solution1_II4
  - ***solution1_II8***
    - constraints
      - directives.tcl
      - script.tcl
    - syn
  - solution2_full_pipeline
  - solution2_unroll16
  - solution2_unroll2
  - solution2_unroll4
  - solution2_unroll8

fir.cpp

```
1 #include "fir.h"
2
3 out_data_t fir_filter (  inp_data_t x,   coef_t c[N])
4 {
5
6   static inp_data_t shift_reg[N];
7
8   acc_t acc = 0;
9   acc_t mult;
10  out_data_t y;
11
12  signed char i;
13
14  Shift_Accum_Loop: for (i=N-1;i>=0;i--)
15  {
16
17     if (i==0)
18     {
19       //acc+=x*c[0];
20       shift_reg[0]=x;
21     }
22     else
23     {
24             shift_reg[i]=shift_reg[i-1];
25             //acc+=shift_reg[i]*c[i];
26     }
27     mult = shift_reg[i]*c[i];
28     acc  = acc + mult;
29
30  }
31
32  y = (out_data_t) acc;
33
34  return y;
35
36 }
```

Outline Directive

- fir_filter
  - % HLS ALLOCATION instances=mul limit=2 operation
  - % HLS PIPELINE II=8
  - shift_reg
  - % HLS ARRAY_PARTITION partition variable=shift_reg complete dim=1
  - x
  - % HLS INTERFACE ap_none register port=x
  - c
  - % HLS ARRAY_PARTITION partition variable=c complete dim=1
  - Shift_Accum_Loop

XILINX ALL PROGRAMMABLE.

HLS solution1_II16: directives to achieve II=16

Vivado HLS - hls_sc_fir1ch_prj (C:\BagniD\V_HLS\multi_channel_FIR_filter\fir1ch\sc\hls_sc_fir1ch_prj)

File Edit Project Solution Window Help

EN English (Ireland) Help

Debug Synthesis Analysis

Explorer

- hls_sc_fir1ch_prj
  - Includes
  - Source
    - fir.cpp
  - Test Bench
    - fir_test.cpp
    - data
  - solution0
  - solution0_lat10
  - solution0_lat20
  - solution1_II1
  - **solution1_II16**
    - constraints
      - directives.tcl
      - script.tcl
    - syn
      - report
      - systemc
      - verilog
      - vhdl
  - solution1_II2
  - solution1_II4
  - solution1_II8
  - solution2_full_pipeline
  - solution2_unroll16
  - solution2_unroll2
  - solution2_unroll4
  - solution2_unroll8

fir.cpp

```cpp
#include "fir.h"

out_data_t fir_filter (  inp_data_t x,   coef_t c[N])
{

  static inp_data_t shift_reg[N];

  acc_t acc = 0;
  acc_t mult;
  out_data_t y;

  signed char i;

  Shift_Accum_Loop: for (i=N-1;i>=0;i--)
  {

     if (i==0)
     {
       //acc+=x*c[0];
       shift_reg[0]=x;
     }
     else
     {
            shift_reg[i]=shift_reg[i-1];
            //acc+=shift_reg[i]*c[i];
     }
     mult = shift_reg[i]*c[i];
     acc  = acc + mult;

  }

  y = (out_data_t) acc;

  return y;

}
```

Outline Directive

- fir_filter
  - % HLS ALLOCATION instances=mul limit=1 operation
  - % HLS PIPELINE II=16
  - shift_reg
  - % HLS ARRAY_PARTITION partition variable=shift_reg complete dim
  - x
  - % HLS INTERFACE ap_none register port=x
  - c
  - % HLS ARRAY_PARTITION partition variable=c complete dim=1
  - Shift_Accum_Loop

XILINX ALL PROGRAMMABLE.

Outlines

- HLS concepts: Initialization Interval and Latency
- FIR filter design: the C++ code
- **FIR filter design: optimization directives and related performance**
  - Solution0: directive LATENCY to limit loop iteration latency between min and max clock cycles
  - Solution1: directives PIPELINE II=x and ALLOCATION (this last one to limit the amount of mult operations)
  - Solution2: directive UNROLL to unroll the loop by x times
- FIR filter design: performance summary

HLS solution2: loop unrolling

- **solution2_unr2:**
  - Loop unrolling by 2
- **solution2_unr4:**
  - Loop unrolling by 4
- **solution2_unr8:**
  - Loop unrolling by 8
- **solution2_unr16:**
  - Loop unrolling by 16

Performance Estimates

Timing (ns)

Clock		solution2_unroll2	solution2_unroll4	solution2_unroll8	solution2_unroll16	solution2_full_pipe
default	Target	10.00	10.00	10.00	10.00	10.00
	Estimated	8.62	8.62	8.62	8.28	8.28

Latency (clock cycles)

		solution2_unroll2	solution2_unroll4	solution2_unroll8	solution2_unroll16	solution2_full_pipeline
Latency	min	33	25	23	2	2
	max	33	25	23	2	2
Interval	min	34	26	24	3	1
	max	34	26	24	3	1

Utilization Estimates

	solution2_unroll2	solution2_unroll4	solution2_unroll8	solution2_unroll16	solution2_full_pipeline
BRAM_18K	0	0	0	0	0
DSP48E	2	4	8	16	16
FF	537	730	1120	927	927
LUT	983	1443	2652	627	626

- **solution2_unr16_fullpip:**
  - Loop unrolling by 16 plus pipeline at top level

HLS solution2_unr2: loop unrolling by 2

Vivado HLS - hls_sc_fir1ch_prj (C:\BagniD\V_HLS\multi_channel_FIR_filter\fir1ch\sc\hls_sc_fir1ch_prj)

File Edit Project Solution Window Help

EN English (Ireland) Help

Debug Synthesis Analysis

Explorer

- hls_sc_fir1ch_prj
  - Includes
  - Source
    - fir.cpp
  - Test Bench
  - solution0
  - solution0_lat10
  - solution0_lat20
  - solution1_II1
  - solution1_II16
  - solution1_II2
  - solution1_II4
  - solution1_II8
  - solution2_full_pipeline
  - solution2_unroll16
  - **solution2_unroll2**
    - constraints
      - directives.tcl
      - script.tcl
    - syn
      - report
      - systemc
      - verilog
      - vhdl
  - solution2_unroll4
  - solution2_unroll8

fir.cpp

```cpp
#include "fir.h"

out_data_t fir_filter (  inp_data_t x,   coef_t c[N])
{

  static inp_data_t shift_reg[N];

  acc_t acc = 0;
  acc_t mult;
  out_data_t y;

  signed char i;

  Shift_Accum_Loop: for (i=N-1;i>=0;i--)
  {

     if (i==0)
     {
       //acc+=x*c[0];
       shift_reg[0]=x;
     }
     else
     {
            shift_reg[i]=shift_reg[i-1];
            //acc+=shift_reg[i]*c[i];
     }
     mult = shift_reg[i]*c[i];
     acc  = acc + mult;

  }

  y = (out_data_t) acc;

  return y;

}
```

Outline Directive

- fir_filter
  - shift_reg
  - HLS ARRAY_PARTITION partition variable=shift_reg complete dim=1
  - x
  - HLS INTERFACE ap_none register port=x
  - c
  - HLS ARRAY_PARTITION partition variable=c complete dim=1
  - Shift_Accum_Loop
    - HLS UNROLL factor=2

XILINX ALL PROGRAMMABLE.

HLS solution2_unr4: loop unrolling by 4

Vivado HLS - hls_sc_fir1ch_prj (C:\BagniD\V_HLS\multi_channel_FIR_filter\fir1ch\sc\hls_sc_fir1ch_prj)

File Edit Project Solution Window Help

EN English (Ireland) Help

Debug Synthesis Analysis

Explorer

- hls_sc_fir1ch_prj
  - Includes
  - Source
    - fir.cpp
  - Test Bench
  - solution0
  - solution0_lat10
  - solution0_lat20
  - solution1_II1
  - solution1_II16
  - solution1_II2
  - solution1_II4
  - solution1_II8
  - solution2_full_pipeline
  - solution2_unroll16
  - solution2_unroll2
  - solution2_unroll4
    - constraints
      - directives.tcl
      - script.tcl
    - syn
      - report
      - systemc
      - verilog
      - vhdl
  - solution2_unroll8
    - constraints
    - syn

fir.cpp

```cpp
#include "fir.h"

out_data_t fir_filter (  inp_data_t x,   coef_t c[N])
{

  static inp_data_t shift_reg[N];

  acc_t acc = 0;
  acc_t mult;
  out_data_t y;

  signed char i;

  Shift_Accum_Loop: for (i=N-1;i>=0;i--)
  {

     if (i==0)
     {
       //acc+=x*c[0];
       shift_reg[0]=x;
     }
     else
     {
            shift_reg[i]=shift_reg[i-1];
            //acc+=shift_reg[i]*c[i];
     }
     mult = shift_reg[i]*c[i];
     acc  = acc + mult;

  }

  y = (out_data_t) acc;

  return y;

}
```

Outline Directive

- fir_filter
  - shift_reg
  - HLS ARRAY_PARTITION partition variable=shift_reg complete dim=1
  - x
  - HLS INTERFACE ap_none register port=x
  - c
  - HLS ARRAY_PARTITION partition variable=c complete dim=1
  - Shift_Accum_Loop
    - HLS UNROLL factor=4

XILINX ALL PROGRAMMABLE.

HLS solution2_unr8: loop unrolling by 8

Vivado HLS - hls_sc_fir1ch_prj (C:\BagniD\V_HLS\multi_channel_FIR_filter\fir1ch\sc\hls_sc_fir1ch_prj)

File Edit Project Solution Window Help

EN English (Ireland) Help

Debug Synthesis Analysis

Explorer

- hls_sc_fir1ch_prj
  - Includes
  - Source
    - fir.cpp
  - Test Bench
  - solution0
  - solution0_lat10
  - solution0_lat20
  - solution1_II1
  - solution1_II16
  - solution1_II2
  - solution1_II4
  - solution1_II8
  - solution2_full_pipeline
  - solution2_unroll16
  - solution2_unroll2
  - solution2_unroll4
  - ***solution2_unroll8***
    - constraints
      - directives.tcl
      - script.tcl
    - syn
      - report
      - systemc
      - verilog
      - vhdl

fir.cpp

```
1 #include "fir.h"
2
3 out_data_t fir_filter (  inp_data_t x,   coef_t c[N])
4 {
5
6   static inp_data_t shift_reg[N];
7
8   acc_t acc = 0;
9   acc_t mult;
10  out_data_t y;
11
12  signed char i;
13
14  Shift_Accum_Loop: for (i=N-1;i>=0;i--)
15  {
16
17     if (i==0)
18     {
19       //acc+=x*c[0];
20       shift_reg[0]=x;
21     }
22     else
23     {
24            shift_reg[i]=shift_reg[i-1];
25            //acc+=shift_reg[i]*c[i];
26     }
27     mult = shift_reg[i]*c[i];
28     acc  = acc + mult;
29
30  }
31
32  y = (out_data_t) acc;
33
34  return y;
35
36 }
```

Outline Directive

- fir_filter
  - x[] shift_reg
  - % HLS ARRAY_PARTITION partition variable=shift_reg complete dim=1
  - x
  - % HLS INTERFACE ap_none register port=x
  - c
  - % HLS ARRAY_PARTITION partition variable=c complete dim=1
  - Shift_Accum_Loop
    - % HLS UNROLL factor=8

XILINX ALL PROGRAMMABLE.

HLS solution2_unr16: loop unrolling by 16

Vivado HLS - hls_sc_fir1ch_prj (C:\BagniD\V_HLS\multi_channel_FIR_filter\fir1ch\sc\hls_sc_fir1ch_prj)

File Edit Project Solution Window Help

EN English (Ireland) Help

Debug Synthesis Analysis

Explorer

- hls_sc_fir1ch_prj
  - Includes
  - Source
    - fir.cpp
  - Test Bench
    - fir_test.cpp
    - data
  - solution0
  - solution0_lat10
  - solution0_lat20
  - solution1_II1
  - solution1_II16
  - solution1_II2
  - solution1_II4
  - solution1_II8
  - solution2_full_pipeline
  - ***solution2_unroll16***
    - constraints
      - directives.tcl
      - script.tcl
    - syn
      - report
      - systemc
      - verilog
      - vhdl
  - solution2_unroll2
  - solution2_unroll4
  - solution2_unroll8

fir.cpp

```
1 #include "fir.h"
2
3 out_data_t fir_filter (  inp_data_t x,   coef_t c[N])
4 {
5
6   static inp_data_t shift_reg[N];
7
8   acc_t acc = 0;
9   acc_t mult;
10  out_data_t y;
11
12  signed char i;
13
14  Shift_Accum_Loop: for (i=N-1;i>=0;i--)
15  {
16
17     if (i==0)
18     {
19       //acc+=x*c[0];
20       shift_reg[0]=x;
21     }
22     else
23     {
24             shift_reg[i]=shift_reg[i-1];
25             //acc+=shift_reg[i]*c[i];
26     }
27     mult = shift_reg[i]*c[i];
28     acc  = acc + mult;
29
30  }
31
32  y = (out_data_t) acc;
33
34  return y;
35
36 }
37
```

Outline Directive

- fir_filter
  - shift_reg
  - HLS ARRAY_PARTITION partition variable=shift_reg complete dim=1
  - x
  - HLS INTERFACE ap_none register port=x
  - c
  - HLS ARRAY_PARTITION partition variable=c complete dim=1
  - Shift_Accum_Loop
    - HLS UNROLL factor=16

XILINX ALL PROGRAMMABLE.

Outlines

- HLS concepts: Initialization Interval and Latency
- FIR filter design: the C++ code
- **FIR filter design: optimization directives and related performance**
  - Solution0: baseline
  - Solution1: directives PIPELINE II=x and ALLOCATION (this last one to limit the amount of mult operations)
  - Solution2: directive UNROLL to unroll the loop by x times
  - Solution3: directive LATENCY to limit loop iteration latency between min and max clock cycles
- FIR filter design: performance summary

HLS solution3: latency

- **solution3:**
  - Partitioning arrays coeff and shift_reg
- **Solution3_lat10:**
  - as solution3 + latency=10
- **Solution3_lat20:**
  - as solution3 + latency=20

Performance Estimates

Timing (ns)

Clock		solution3	solution3_lat10	solution3_lat20
default	Target	10.00	10.00	10.00
	Estimated	8.62	8.62	8.62

Latency (clock cycles)

		solution3	solution3_lat10	solution3_lat20
Latency	min	49	161	321
	max	49	177	337
Interval	min	50	162	322
	max	50	178	338

Utilization Estimates

	solution3	solution3_lat10	solution3_lat20
BRAM_18K	0	0	0
DSP48E	1	1	1
FF	418	492	502
LUT	373	342	354

Outlines

- HLS concepts: Initialization Interval and Latency
- FIR filter design: the C++ code
- FIR filter design: optimization directives and related performance
  - Solution0: directive LATENCY to limit loop iteration latency between min and max clock cycles
  - Solution1: directives PIPELINE II=x and ALLOCATION (this last one to limit the amount of mult operations)
  - Solution2: directive UNROLL to unroll the loop by x times
- FIR filter design: performance summary

FIR Filter design: performance summary

solution name	clock freq MHZ	Latency	Interval	data rate MSPS	DSP48	FF	LUT
solution0_a: baseline	100	97	98	1.02	1	202	133
solution0_b: Loop pipeline	100	37	38	2.63	1	171	142
solution0_c: +partitioning shiftreg	100	20	21	4.76	1	466	269
solution0_d: +Loop unrolling	100	11	12	8.33	16	1084	648
solution0_e: +partitionin coeff array	100	2	3	33.33	16	927	627
solution1_II1	100	2	1	100.00	16	927	626
solution1_II2	100	3	2	50.00	8	640	916
solution1_II4	100	5	4	25.00	4	610	773
solution1_II8	100	9	8	12.50	2	654	776
solution1_II16	100	17	16	6.25	1	662	823
solution2_unroll2	100	33	34	2.94	2	537	983
solution2_unroll4	100	25	26	3.85	4	730	1443
solution2_unroll8	100	23	24	4.17	8	1120	2652
solution2_unroll16	100	2	3	33.33	16	927	627
solution2_unroll_full_pipelined	100	2	1	100.00	16	927	626
solution3	100	49	50	2.00	1	418	373
solution3 latency10	100	177	178	0.56	1	492	342
solution3_latency20	100	337	338	0.30	1	502	354

FIR Filter design: HLS directives summary

- **So far we have used:**
  - ARRAY_PARTITION
  - ALLOCATION
  - PIPELINE
  - UNROLL
  - LATENCY
  - LOOP TRIP_COUNT

Agenda

- **1) FIR filter case: 1 channel, in fractional fixed-point precision**
  - ARRAY_PARTITION, ALLOCATION, PIPELINE, UNROLL
  - LATENCY, LOOP TRIP_COUNT
- **2) FIR filter case: 1 channel, in Floating-Point**
- **3) Floating-Point Accumulator**
- **4) Dependency**
- **5) image Histogram computation and equalization**
- **6) Siemens' application "Gamma LUT" case study**
- **7) Image Processing: from HLS to IPI, from IPI to ZC702 board**
- **8) cordic arctan2**
- **9) cordic sqrt**