Performance Benchmarking Embedded FPGAs

Application Note, v1.3

Contents

1. Purpose of this Application Note 2
2. Setup & Tools 2
3. Introduction 2
   3.1 EFLX® Embedded FPGA Architecture 3
4. Benchmarking Metrics 7
5. Single LUT Stage Logic 7
6. Optimizing Verilog for Performance of Single-Stage Logic 9
   6.1 Virtual Switch 15
   6.2 16-bit Counter 17
   6.3 Wide Comparator 17
   6.4 18x18 Signed Multiply 20
   6.5 FIR Filters 22
      6.5.1 5-tap FIR Filter 24
      6.5.2 21-tap Symmetric FIR Filter 26
      6.5.3 40-tap Non-symmetric FIR Filter 27
7. Multi-Stage Logic 30
   7.1 ARM Cortex-M0 31
   7.2 8051 MCU 34
   7.3 AES-128 35
   7.4 256-Point FFT 36
   7.5 SHA-256 38
   7.6 JPEG 39
8. Summary 41

8.1 Performance Benchmarking on EFLX Cores 41
8.2 Online Resources 41

1. Purpose of this Application Note

This Application Note has the following objectives:

1) Provide performance benchmark achievable on the various EFLX IP Cores currently available;
2) Provide performance variation for different process nodes;
3) Provide performance benchmarks to compare with other embedded FPGAs.

2. Setup & Tools

All the designs used in this document were synthesized using Synopsys SynplifyPro OEM version targeting FlexLogix EFLX eFPGA.

The followng synthesis and compiler tools were used for this Application Note:

- Synopsys Synplify Pro R-2021.03FL-SP1-Beta2.
- EFLX Compiler Version 202211F.2 for different process nodes.

All the EFLX compiler runs were done without I/O files except those which specifically mentions IO constraint. We let the compiler do the I/O placement for optimal performance. Even though this doesn't represent how the embedded FPGA will be used when integrated into an SoC, the performance benchmarks were used to highlight the RTL code performance in the fabric independent of I/O placement.

3. Introduction

When evaluating the performance of an embedded FPGA, one needs to evaluate the performance of each of the individual modules that make up an FPGA. The basic modules (Fig. 1) are:

- Reconfigurable logic building blocks (RBB-Logic),
    - Fine-granularity logic containing LUTs, carry-forwarding adder chain, and flip-flops
- Reconfigurable DSP building blocks (RBB-DSP),
    - Medium-granularity arithmetic containing multiply, adder, and MAC functions
- Interconnect network (switch fabric),
- Reconfigurable building blocks for peripheral I/O (RBB-I/O),
- Optional Block RAM in between reconfigurable logic and DSP blocks.

Copyright 2023 Flex Logix® Technologies, Inc.

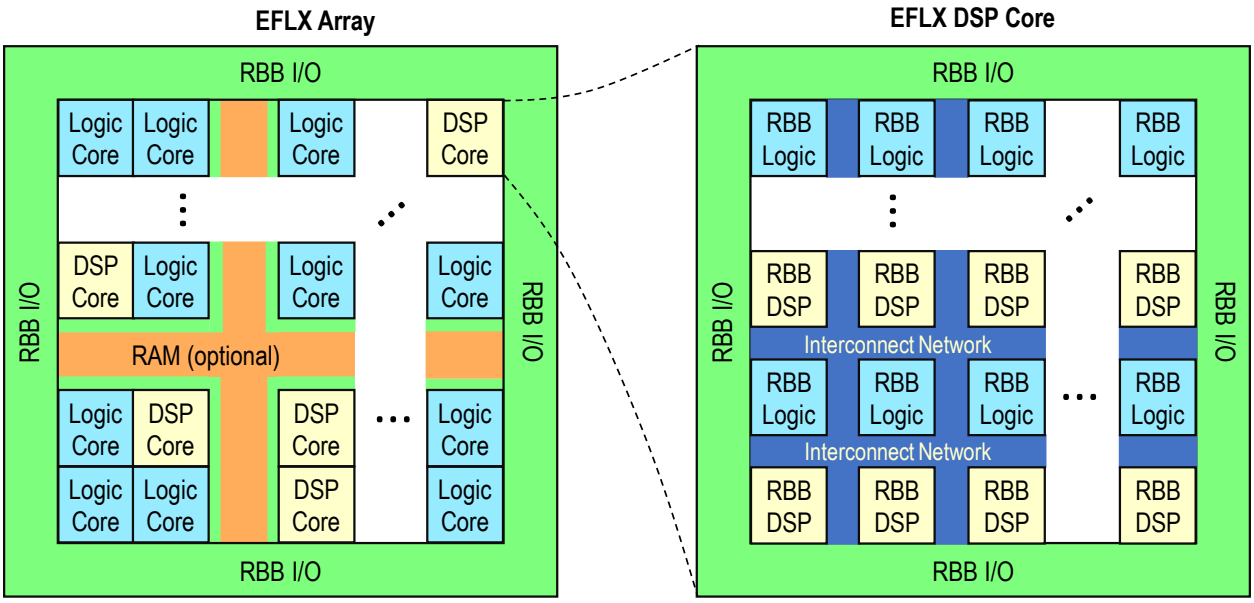


Figure 1 FPGA Compute elements. EFLX Array (left) consists of DSP, Logic and RAM (optional) blocks inclusive of configuration bits and tile-based interconnect network. The array is surrounded with I/O interface for SoC integration. DSP Core (right) consists of DSP Reconfigurable Building Blocks (RBBs), interconnect network and I/O. The I/O is disabled in abutment of Cores, at the intra-Core boundaries, except when optional RAM is inserted between rows/columns of the EFLX Cores.

3.1 EFLX® Embedded FPGA Architecture

The EFLX Cores have 3 Reconfigurable Building Blocks (RBBs): Logic, I/O and DSP. The interconnect network and associated programming bits are contained within RBBs to allow efficient array-style tiling for size customization. The logic RBB includes either four dual 4-input LUTs (EFLX Gen-1 architecture) or four dual 5-input LUTs (EFLX Gen-2 architecture), 8 flip-flops, carry chain and selectable clocks (Fig. 2). The 4-dual input LUTs can also be used as a single 5-input LUT and the 4 dual 5-input LUTs can be used as a single 6-input LUT. The total number of LUTs in an embedded FPGA is counted as the total number of equivalent 4-input LUTs (e.g. 2.5K total 6-input LUTs corresponds to 4K 4-input LUTs in Gen-2 architecture). The Gen-2 architecture has also made significant improvements to the interconnect to improve routing capacity, performance, and RBB utilization. For example, a Gen-2 design on TSMC 28 would be ~40% faster than a Gen-1 design on TSMC 28.

Copyright 2023 Flex Logix® Technologies, Inc.

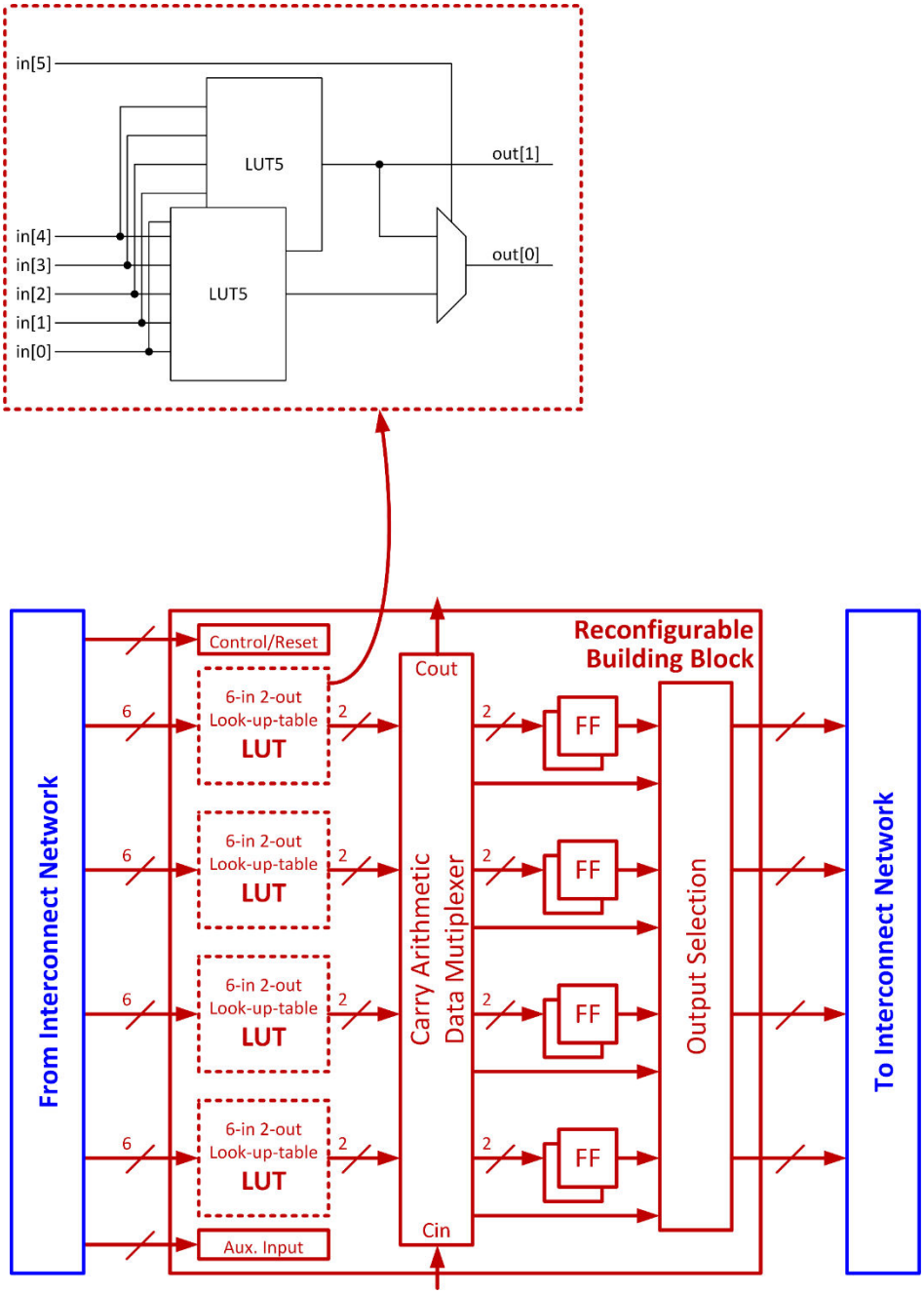


Figure 2 RBB-Logic architecture.

The I/O RBB (Fig. 3) supports flopped and pass-through I/O modes.

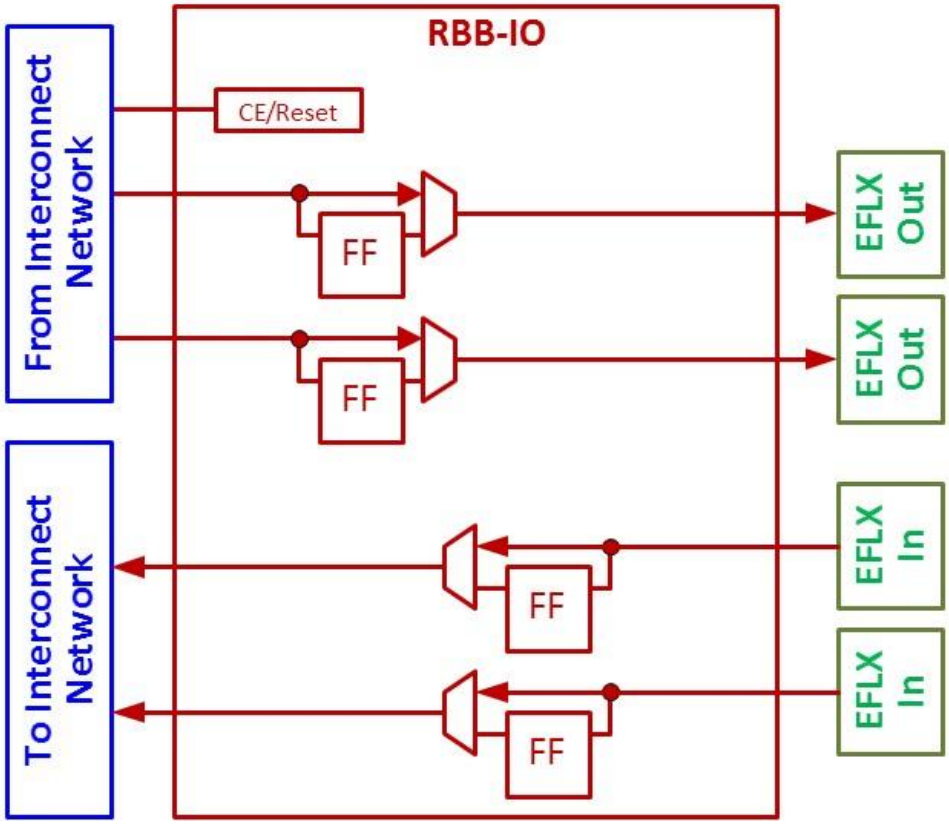


Figure 3 RBB-IO architecture.

The RBB-DSP (Fig. 4) is a MAC with 22-bit pre-adder, 22-bit x 22-bit fixed-point multiplier, and 48-bit post-adder (accumulator). PO_LOC, PI_LOC, AO_LOC, BO_LOC and DO_LOC are used for cascading DSPs and do not go through the interconnect network. PO_NWK output goes to an interconnect network and BI_NWK, AI_NWK and DI_NWK inputs come from an interconnect network.

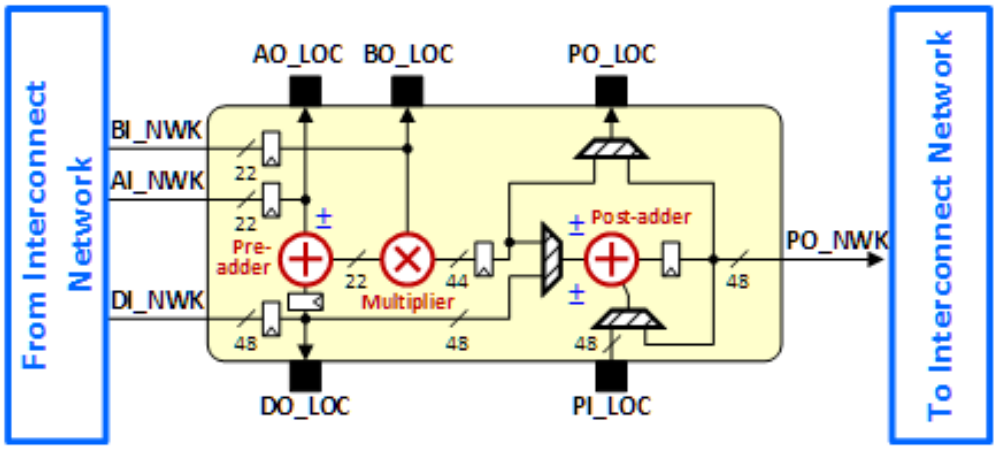


Figure 4 RBB-DSP architecture.

The DSP MACs can be cascaded without going through an interconnect fabric as shown below (Fig. 5):

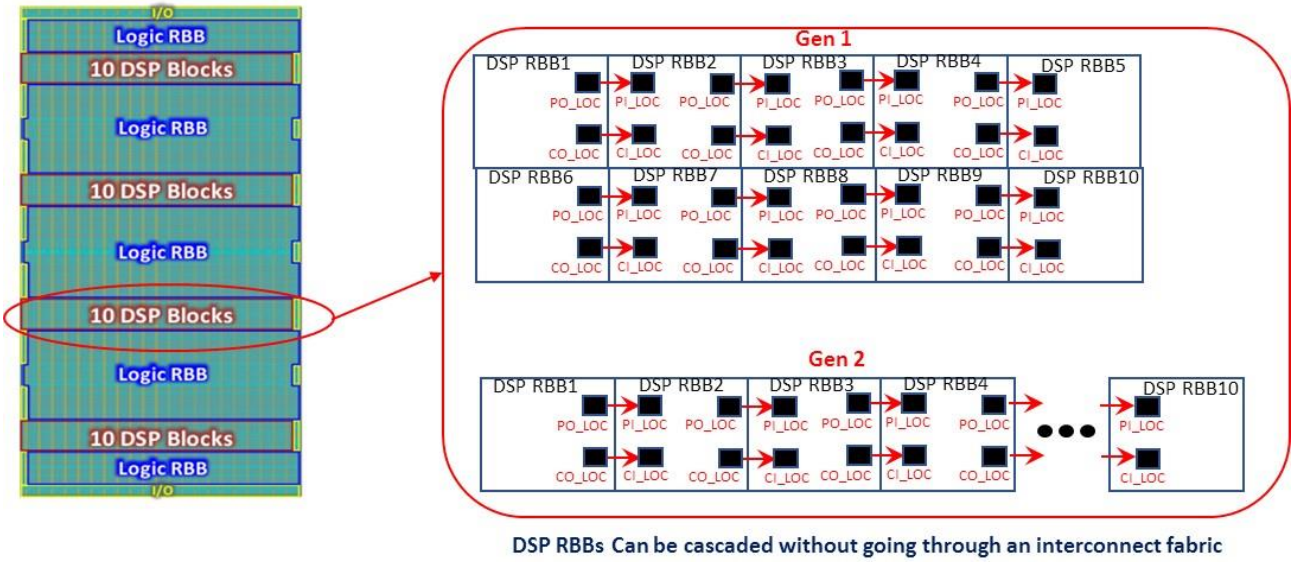


Figure 5 Cascaded DSP RBBs. Gen-2 architecture allows cascading of up to 10 DSP RBBs. Detail of EFLX4K Core is shown.

RBBs mentioned above can be arranged into Cores of varying size. The two most popular Cores are EFLX4K Logic core and EFLX4K DSP core (Fig. 6).

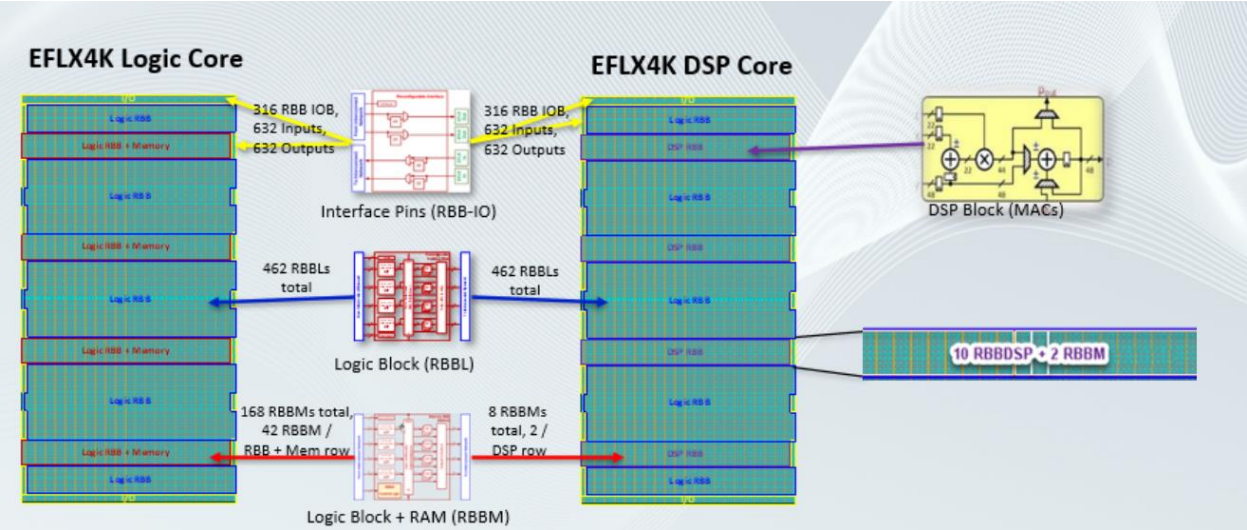


Figure 6 EFLX4K LM Core and EFLX4K DSP Core.

4. Benchmarking Metrics

When benchmarking an embedded FPGA, the following metrics need to be considered:

- Maximum Clock rate,
  - Single LUT stage logic clock rate
  - Multiple LUT stage logic clock rate
- Energy efficiency,
  - Compute energy
  - Configuration energy
- Area (resource) utilization.

5. Single LUT Stage Logic

The metric used to evaluate the performance of the basic compute elements is the single-stage logic operation. The single-stage logic operation is defined in the diagram below.

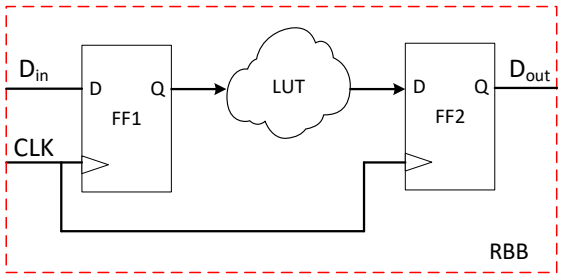


Figure 7 Single-stage logic pipeline.

The maximum clock frequency for the single-stage logic is defined by the following:

$$f_{max} = 1/(FF1_CLK\text{-}to\text{-}Q_Max_Delay + LUT_Max_Delay + FF2_CLK_Setup)$$

When evaluating the single-stage performance of an embedded FPGA, the single-stage operation can take one of the 4 forms shown in Fig. 8, as dictated by logic paritioning:

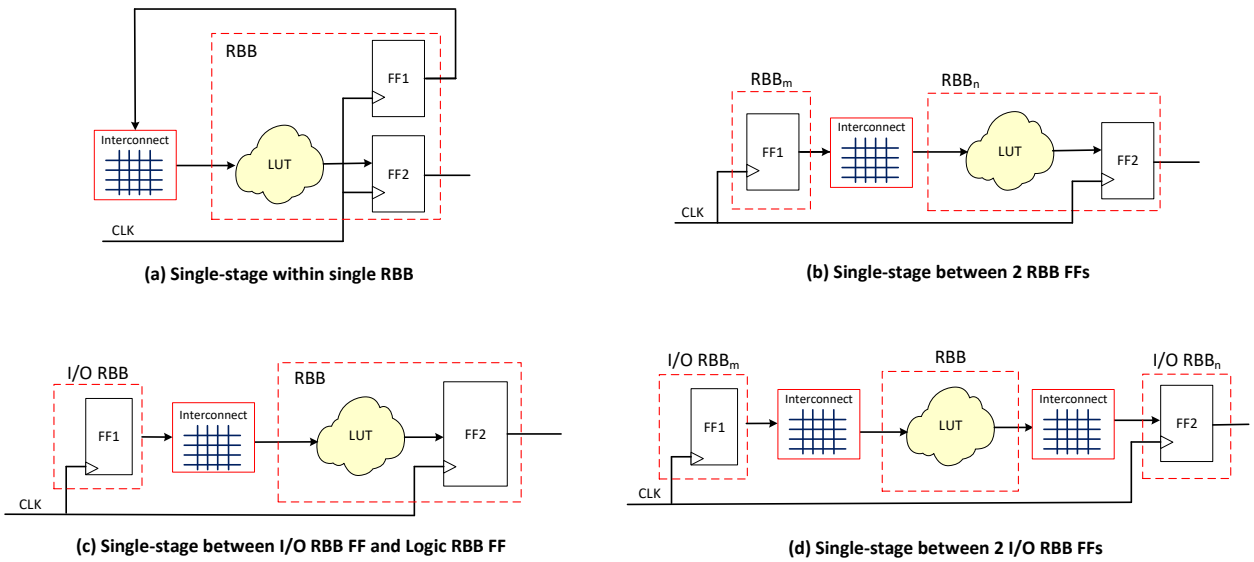
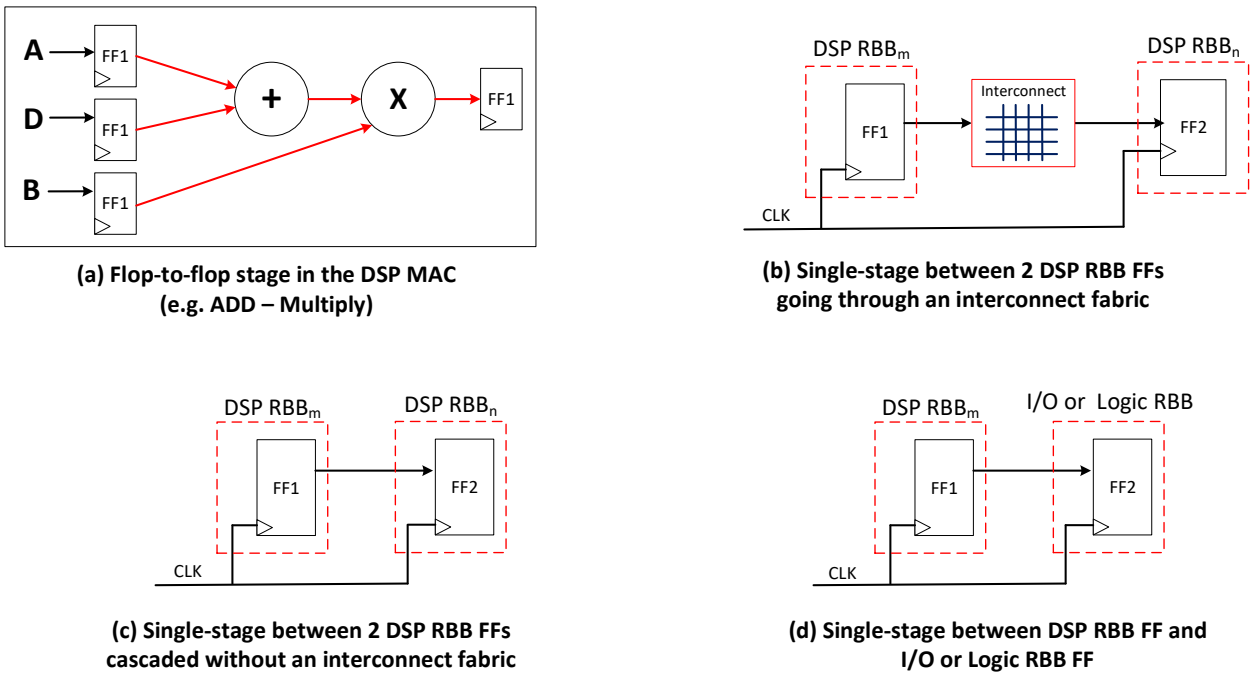


Figure 8 Single-stage logic forms in an embedded FPGA:
(a) within single RBB, (b) between 2 RBB FFs, (c) between I/O RBB FF and Logic RBB FF, and (d) between 2 I/O RBB FFs.

The critical-path scenario, as shown in Fig. 8, is decided by the mapping and routing toolflow. All cases have to be considered to understand critical-path delay and its variation considering the logic and interconnect resources.

The DSP elements also factor into the maximum frequency of the eFPGA since they have a flop-to-flop maximum delay associated with it. There are 4 critical timing paths associated with the DSP elements:

- Flop-to-flop stage inside the DSP (e.g. ADD-Multiply),
- DSP-to-DSP via an interconnect fabric,
- Cascaded DSP without an interconnect fabric,
- DSP to I/O or Logic RBB via an interconnect fabric.



(a) Flop-to-flop stage in the DSP MAC (e.g. ADD – Multiply)

(b) Single-stage between 2 DSP RBB FFs going through an interconnect fabric

(c) Single-stage between 2 DSP RBB FFs cascaded without an interconnect fabric

(d) Single-stage between DSP RBB FF and I/O or Logic RBB FF

Figure 9 Single-stage logic forms for DSP elements:
(a) flop-to-flop in the DSP MAC, (b) between 2 DSP RBB FFs going through an interconnect fabric,
(c) between 2 DSP RBB FFs cascaded without an interconnect fabric, (d) between DSP FF and I/O or Logic RBB FF.

Like in Logic RBB, DSP RBB critical-path timing depends on the mapping and rouding toolflow. The options above need to be considered in critical-path analysis. Figures 8 & 9 cover all crtical-path senarios for Logic and DSP RBBs.

6. Optimizing Verilog for Performance of Single-Stage Logic

The determination of which form the single-stage logic takes place depends on the RTL coded into the embedded FPGA.

Let's take the following RTL example of an inverter LUT surrounded with I/O RBB flip-flops:

```
module single_stage  (in, out, clk);
  input in, clk;
  output out;
  reg out;          // output I/O RBB
  reg in_r;         // input I/O RBB

always @(posedge clk)   // LUT
          begin
          in_r <= in;
          out <= !in_r;
          end

endmodule
```

The above RTL is shown in schematic form in Fig. 10:

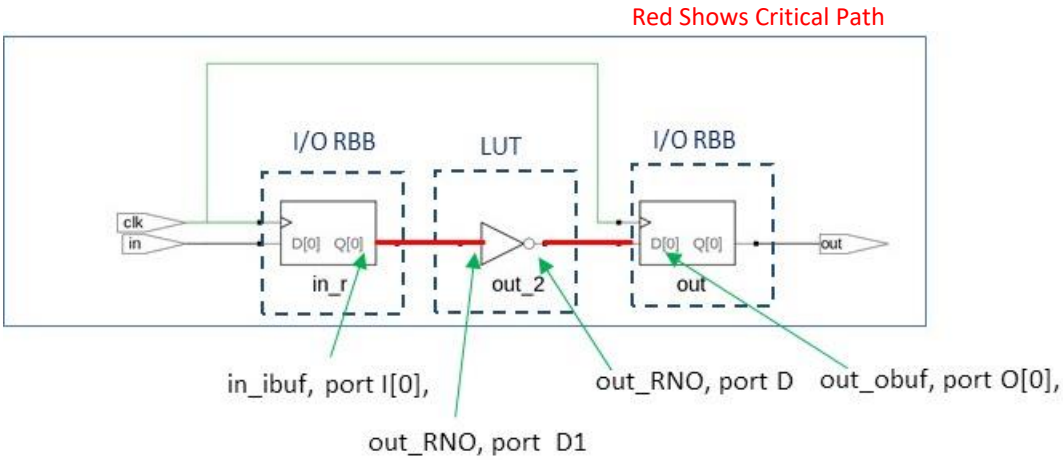


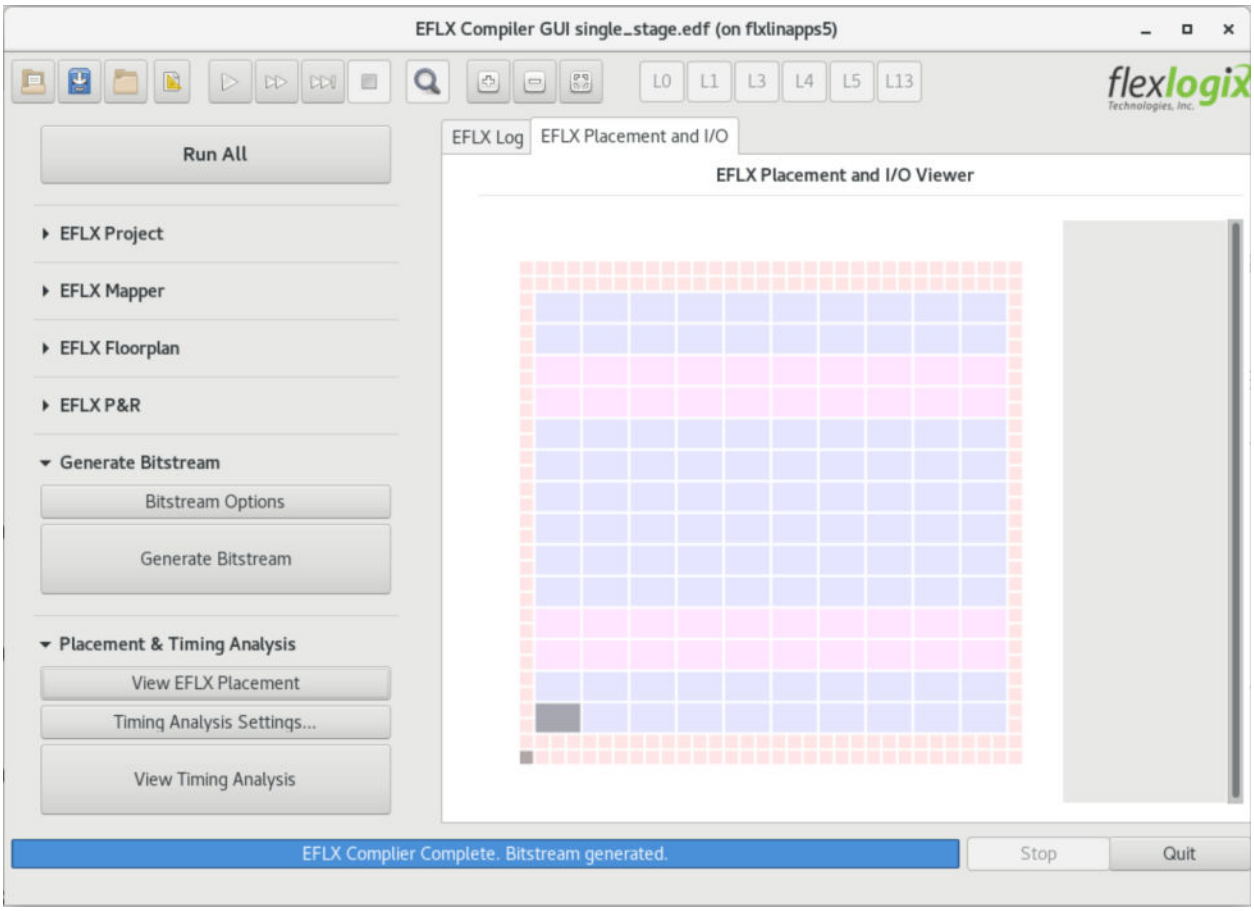
Figure 10 Non-pipelined output single-stage schematic.

When compiling the above code into an embedded FPGA such as the EFLX4K in TSMC's T16FFC process, for example, the maximum frequency achieved is 1.77 GHz (TT, 0.8v, 85c, rctypical). This frequency is being limited by the routing to and from the I/O blocks because both flip-flops are in the I/O blocks. In other words, two routing hops are included in the overall path delay.

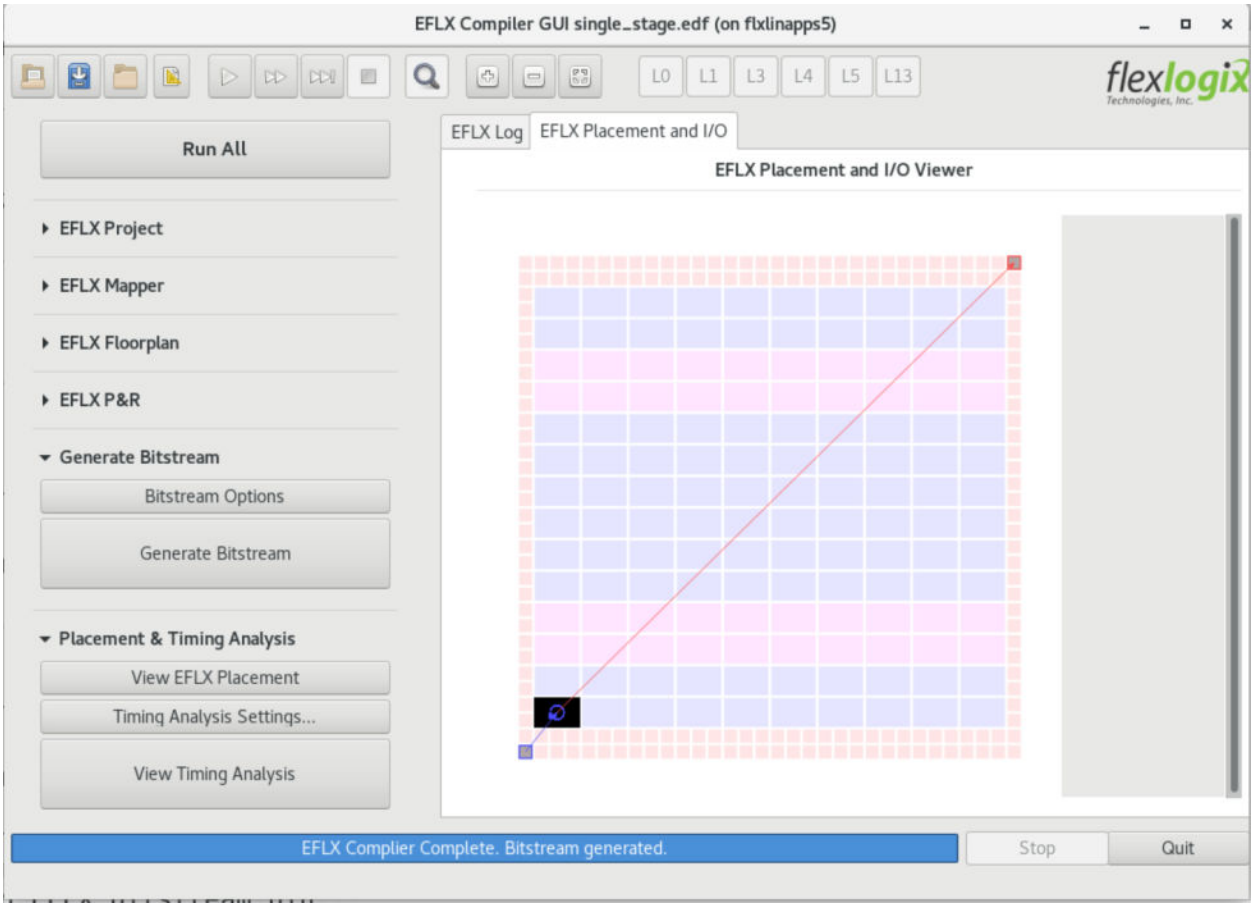
The worst-case timing path is as follows (shown in red in Fig. 10) for TSMC's T16FFC (corner tt0p8v85c_rctypical):

```
Instance/Pin or Net Name   Type         ID   Location or Fanout   Delay(ps)  Edge  Arrival Time(ps)    Instance detail
-------------------------------------------------------------------------------------------------------------------------------------------------
clk_c                      CLOCK-PORT    -                    -     launch     r               0
clk_c                      Clock net     5          (fanout=5)        997                     -
RBB_0/CLK                  IOB           0  [TILE 0 0, RBB 0 0]          -     r             997
-------------------------------------------------------------------------------------------------------------------------------------------------
RBB_0/I[0]                 IOB           0  [TILE 0 0, RBB 0 0]          0     r             997     {(input_port) inst9: in_ibuf (IO)} {inst6: in_r (FD)}
in_r                       Net           0          (fanout=1)        201                     -
RBB_1/D1                   RBB_6L        1  [TILE 0 0, RBB 1 2]          -     r            1198
RBB_1/D                    RBB_6L        1  [TILE 0 0, RBB 1 2]        124     r            1322     {inst10: out_RNO (LUT,site=dlut)}
in_r_i                     Net           9          (fanout=1)        139                     -
RBB_0/O[0]                 IOB           0  [TILE 0 0, RBB 0 0]          -     r            1461     {inst7: out (FD)} {(output_port) inst8: out_obuf (IO)}
-------------------------------------------------------------------------------------------------------------------------------------------------
RBB_0/CLK                  IOB           0  [TILE 0 0, RBB 0 0]         51     r            1512
clk_c                      Clock net     5          (fanout=5)       -997                     -
clk_c                      CLOCK-PORT    -                    -    capture     r             515
-------------------------------------------------------------------------------------------------------------------------------------------------
```

The interconnect delays contribute the bulk of the delay. The first routing hop is 201 ps, the second routing hop is 139 ps, for a total of 340 ps of delay. In this example, the delay due to routing hops contributes to 73% of the overall path delay (not including the output buffer setup delay). Thereby, eliminating interconnect hops can improve the performance of a design. Figure 11 shows the minimum and maximum interconnect delays determined by the I/O and LUT placement. On the left the I/Os are placed close to the LUTs (shortest path). On the right the I/Os are placed far apart with the LUT placed in the center of the Core (longest path). Table 1 summarizes min / max values of the routing-hop delays.



(a) Short Path for EFLX1K at TSMC 40ULP



(b) Long Path for EFLX1K at TSMC40ULP

Figure 11 Routing paths.

A summary of interconnect delays is provided in Table 1.

Table 1 Interconnect delays.

Process Corner	**TSMC 40ULP** tt1p1v25c_Typical		**TSMC 28HPC+** tt0p9v25c_rctypical		**TSMC 16FFC** tt0p8v85c_rctypical		**TSMC 7** tt0p75v25c_typical	
EFLX Core	EFLX1K		EFLX4K		EFLX4K		EFLX4K	
Routing path	Short	Long	Short	Long	Short	Long	Short	Long
Interconnect delay (ps)	1963	4405	429	1690	340	1326	305	1176
Inverter delay (ps)	694		216	216	124	121	120	
Interconnect delay percentage	73%	86%	66%	89%	73%	91%	72%	91%

From Table 1, the interconnect delay for the long path take more than 85% of the total delay.

Note the above table does not include setup time to the destination flop or the clock delta between the launch flop to the destination flop (e.g. 51 ps setup time and -9ps clock delta for 16FFC).

Rewriting the RTL by pipelining the output decreases the interconnect delay and can increase the max frequency by ~21% for the shortest path by removing one interconnect hop and an input mux. Adding

pipeline flops doesn't add to the total static power, it only adds a small percent to the total dynamic power. The critical-path delay for both non-pipelined and pipelined design is the same: FF → inverter → FF. The difference in the pipelined case is that the second FF is in the same RBB as the inverter and reduces 1 routing hop. In general, the designer should always try to merge sharable resources in the same block as shown in Fig. 12, to increase performance and area efficiency.

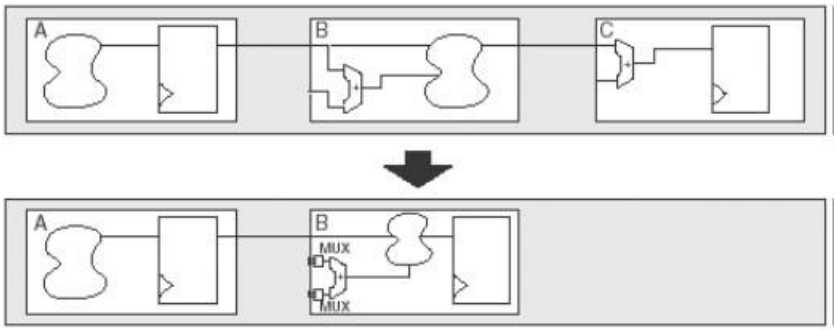


Figure 12 Merging sharable resources into one block.

Single-Stage Pipelined RTL

```
module single_stage  (in, out, clk);
  input in, clk;
  output out;

  reg out;
  reg in_r, out_r;

always @(posedge clk)
          begin
          in_r <= in;
          out_r <= !in_r;
          out<=out_r;  // Pipelining the output
          end
endmodule
```

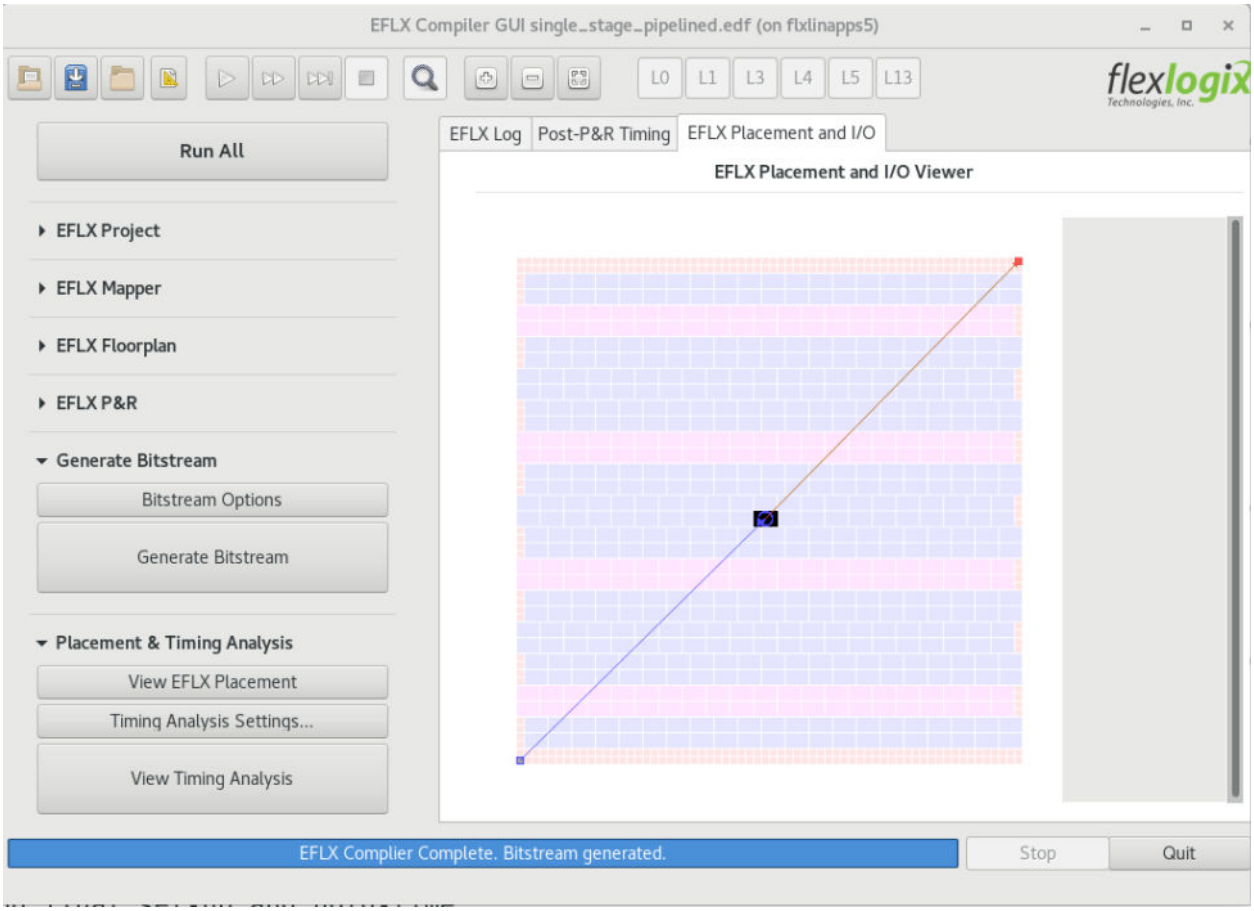


Figure 13 Single-stage Pipelined Critical Path

The critical path for the single stage pipelined logic for the above example is from the flop in the Logic RBB (1 interconnect hop) to the flip-flop in the I/O RBB (1 interconnect hop) as shown below for TSMC 16FFC (corner tt0p8v85c_rctypical):

```
Instance/Pin or Net Name   Type         ID   Location or Fanout   Delay(ps)  Edge  Arrival Time(ps)    Instance detail
-------------------------------------------------------------------------------------------------------------------------------------------
clk_c                      CLOCK-PORT    -                    -     launch     r               0
clk_c                      Clock net     6          (fanout=41)        972                     -
RBB_1/CLK                  RBB_6L        1  [TILE 0 0, RBB 1 2]          -     r             972
-------------------------------------------------------------------------------------------------------------------------------------------
RBB_1/DQ                   RBB_6L        1  [TILE 0 0, RBB 1 2]         80     r            1052     {inst7: out_r (FD,site=dff)}
out_r                      Net           1           (fanout=1)        200                     -
RBB_0/O[0]                 IOB           0  [TILE 0 0, RBB 0 0]          -     r            1252     {inst8: out (FD)} {(output_port) inst9: out_obuf (IO)}
-------------------------------------------------------------------------------------------------------------------------------------------
RBB_0/CLK                  IOB           0  [TILE 0 0, RBB 0 0]         51     r            1303
clk_c                      Clock net     6          (fanout=41)       -990                     -
clk_c                      CLOCK-PORT    -                    -    capture     r             313
-------------------------------------------------------------------------------------------------------------------------------------------
```

Table 2 shows the performance improvements of a single LUT stage logic when the output is pipelined. Without an I/O file specified, the compiler optimizes the placement of the I/Os and the inverter function so that the routing takes the shortest path from input to output for best performance (see Fig. 11 - Shortest Path).

Table 2 Single LUT stage logic performance using pipelined outputs

Technology Corner	**TSMC 40ULP** tt1p1v25c_Typical		**TSMC 28HPC+** tt0p9v25c_rctypical		**TSMC 16FFC** tt0p8v85c_rctypical		**TSMC 7** tt0p75v25c_typical	
EFLX Core	EFLX1K		EFLX4K		EFLX4K		EFLX4K	
Routing	Short	Long	Short	Long	Short	Long	Short	Long
Non-pipelined output stage[1]	278 MHz	166 MHz	1.26 GHz	494 MHz	1.77 GHz	634 MHz	2.01 GHz	741 MHz
Pipelined output stage[2]	338 MHz	238 MHz	2.01 GHz	828 MHz	2.76 GHz	1.12 GHz	2.86 GHz	1.25 GHz

1. Figure 10 plus setup on destination flip flop and clock delta.
2. See Figure 13.

6.1 Virtual Switch

Throughput is another metric to measure performance. Throughput is defined as follows:

Throughput (Bytes/sec) = Number_of_Bytes * (Frequency/Cycles_per_Byte)

An embedded FPGA has the advantage of supporting very wide data paths because of a large number of I/Os available on the embedded FPGA Cores.

- EFLX1K support 368 inputs, 368 outputs per Core
- EFLX4K supports 632 inputs, 632 outputs per Core

Wide data paths allow the embedded FPGA to achieve high throughput when combined with single-stage logic. A virtual switch implemented in an embedded FPGA is an example of high throughput with single-stage logic. The virtual switch has input and output registers without any LUTs between the registers as shown below. The throughput of this design is limited by the flip-flop timing (Clk-Q and setup) and routing-hop delay.

The virtual switch used an io file placing the inputs on the south side of the core and the outputs on the north side of the core. Two RTL cases were written so that the data goes straight through (e.g. in[15:0] routed to out[15:0]). and data crosses (e.g. in[63:48] routed to out[15:0]).

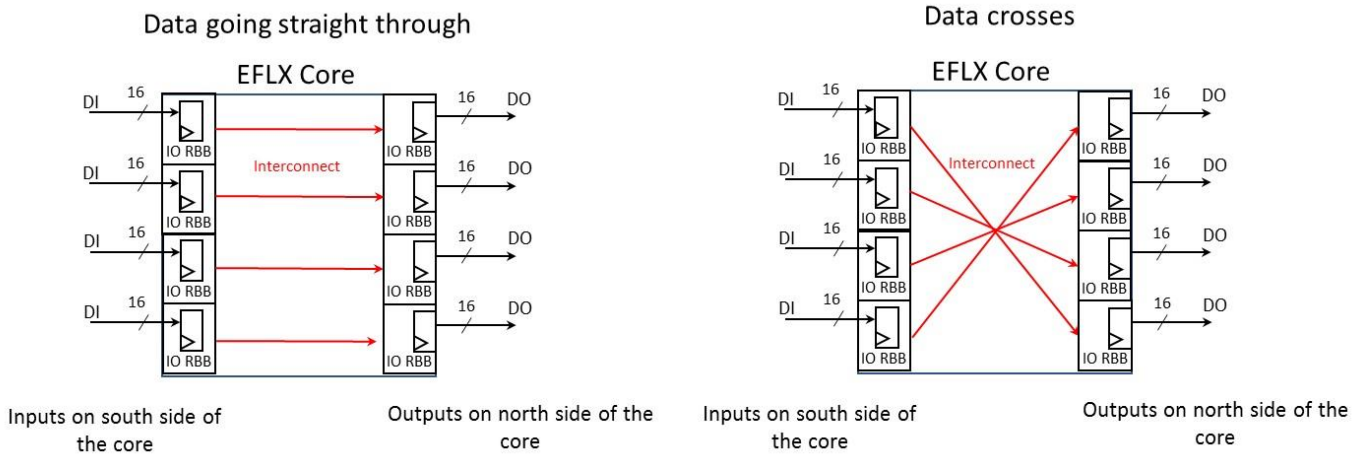


Figure 14 Virtual Switch.

Virtual Switch RTL

Data goes straight through

Data crosses

```
module virtual_switch (in, out,clk);

parameter WIDTH=64;
  input clk;
  input [WIDTH-1:0] in;
  output [WIDTH-1:0] out;

  reg [WIDTH-1:0] out,inr;


always @(posedge clk)
inr<=in;

always @(posedge clk)
        begin
                out[15:0] <= inr[15:0];
                out[31:16] <= inr[31:16];
                out[47:32] <= inr[47:32];
                out[63:48] <= inr[63:48];
        end

endmodule
```

```
module virtual_switch (in, out,clk);

parameter WIDTH=64;
  input clk;
  input [WIDTH-1:0] in;
  output [WIDTH-1:0] out;

  reg [WIDTH-1:0] out,inr;


always @(posedge clk)
inr<=in;

always @(posedge clk)
        begin
                out[15:0] <= inr[63:48];
                out[31:16] <= inr[47:32];
                out[47:32] <= inr[31:16];
                out[63:48] <= inr[15:0];
        end

endmodule
```

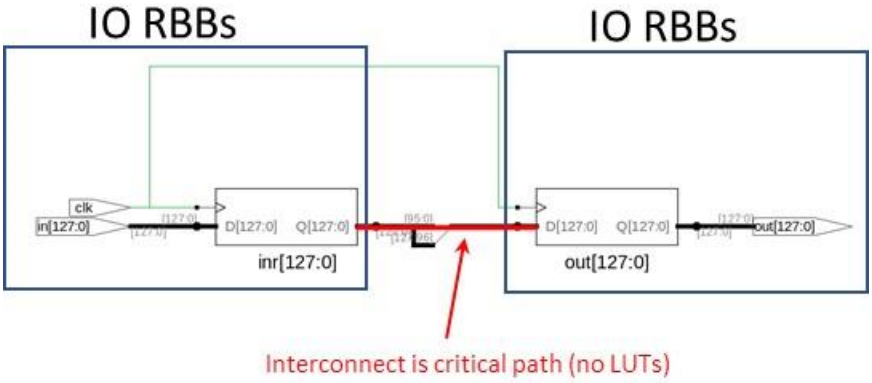


Figure 15 Single-stage virtual switch schematic

Table 3 shows the throughput for 2 technology nodes. Also, the I/Os in this example are packed together using most of the available I/Os by using the flag "`-ENABLE_HIGH_DENSITY_IO_PACKING 1`" when invoking the EFLX Compiler.

Table 3 Throughput for single-stage logic, Virtual Switch design.

Technology Corner	**TSMC 40ULP** tt1p1v25c_Typical		**TSMC 16FFC** tt0p8v85c_rctypical	
EFLX Core	EFLX1K		EFLX4K	
Data input to output mapping	Straight through	Crossing	Straight through	Crossing
Data width	64bits			
Max Frequency	407 MHz	407 MHz	2.92 GHz	2.92 GHz
Throughput	3256 MBytes/s	3256 Mbytes/s	23.36 GBytes/s	23.36 GBytes/s

The performance delta between 40ULP and 16FFC is ~7x. Most of the difference is attributed to the 40ULP process.

6.2 16-bit Counter

The table below shows the performance of the 16-bit counter.

Table 4 16-bit counter performance

Technology Corner	**TSMC 40 ULP** tt1p1v25c_Typical	**TSMC 28 HPC+** tt0p9v25c_rctypical	**TSMC 16 FFC** tt0p8v85c_rctypical	**TSMC 7** tt0p75v25c_typical
EFLX Core	EFLX1K	EFLX4K	EFLX4K	EFLX4K
Max Frequency	173 MHz	780 MHz	1072 MHz	1231 MHz

6.3 Wide Comparator

Another example for single LUT stage logic is a 64-bit comparator shown in Fig. 19.

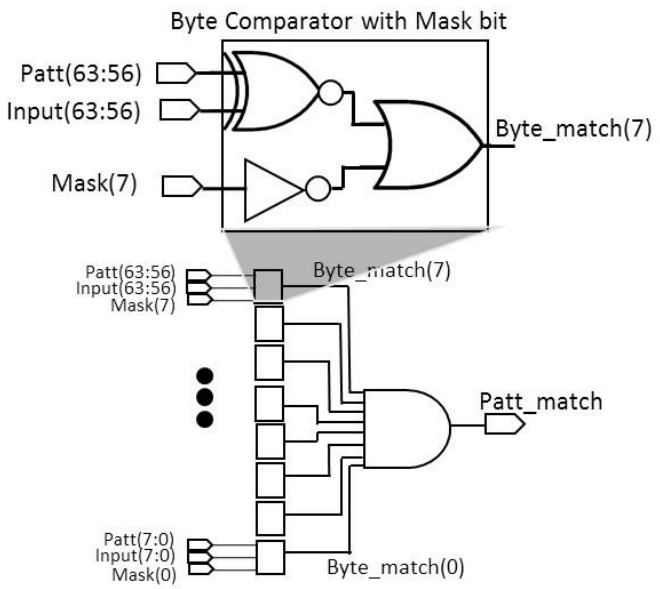


Figure 16 64-bit comparator with mask bits

In order to make this function into a single LUT stage function, the design was pipelined by adding 3 pipelined states as shown in Fig. 20.

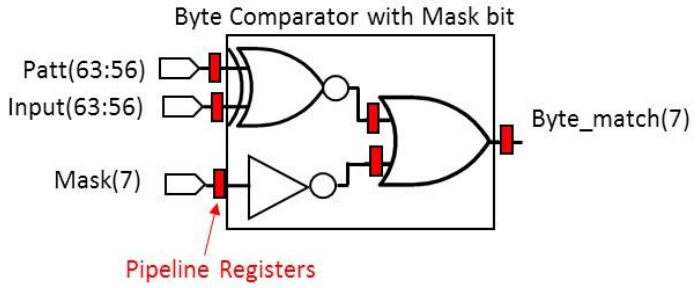


Figure 17 Adding pipelining to Byte Comparator.

Below are the RTL and schematic view of the pipelined 64-bit comparator.

64-bit pipelined comparator with mask bit RTL

```
module comparator #(
  parameter WIDTH = 64
) (
  input clk,
  input rst_n,
  input [WIDTH-1:0] a,
  input [WIDTH-1:0] b,
  input [WIDTH/8-1:0] mask,  // if the mask is set to 1,
do not compare the byte.
  output reg equal
);

  reg [WIDTH-1:0] a_reg, b_reg;
  reg [WIDTH/8-1:0] mask_reg;

  reg [WIDTH/8-1:0] byte_eq;

  always @(posedge clk) begin
    if (!rst_n) begin
      a_reg <= {WIDTH{1'b0}};
      b_reg <= {WIDTH{1'b0}};
      mask_reg <= {WIDTH/8{8'h0}};
      equal    <= 1'b0;
    end
    else begin
      a_reg <= a;
      b_reg <= b;
      mask_reg <= mask;
      equal    <= &byte_eq;
    end
  end
  always @(*) begin
    for (integer i = 0; i < WIDTH/8; i = i + 1) begin
      if (!(mask_reg[i])) begin
        byte_eq[i] = (a_reg[i*8 + 7 -: 8] ==
b_reg[i*8 + 7 -: 8]) ? 1'b1 : 1'b0;
      end
      else begin
        byte_eq[i] = 1'b1; // for the masked bytes,
treat it as equal
      end
    end
  end

endmodule
```

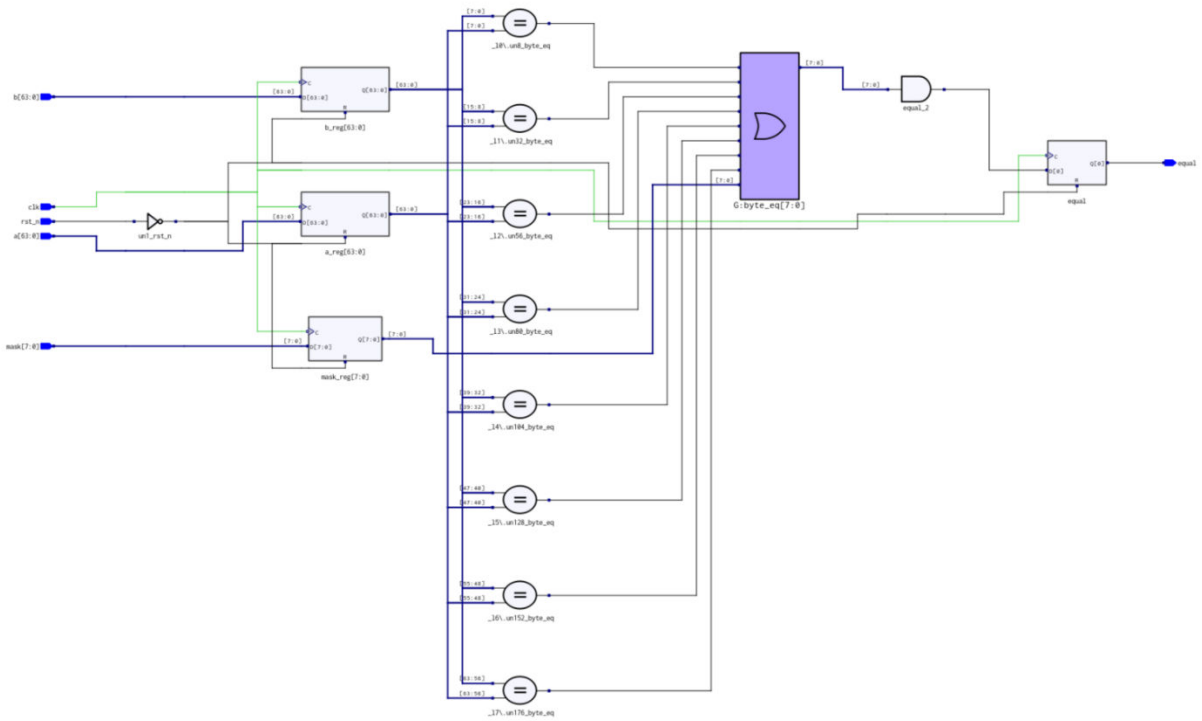


Figure 18 Schematic of a 64-bit Pipelined Comparator with mask bit.

The 64-bit comparator fits into 1 EFLX1K Core.

EFLX1K
Resource Utilization Report #### Top level module name: comparator Resource usage: 10 Logic 6-LUT RBBs 36 RBB LUTs utilized (up to 4 per RBB) 70 I/O RBBs 137 Input pins 136 Input FFs utilized 1 Output pins 1 Output FFs utilized 1 Clock Clock 0: net 171, clk_c, Fanout: 69 RBBs need 0 DSP and 1 Logic-Memory EFLX 1K Tiles. #### Resource Utilization Complete

Table 5 shows the performance of the 64-bit comparator.

Table 5 64-bit pipelined comparator performance.

Technology Corner	**TSMC 40ULP** tt1p1v25c_Typical	**TSMC 28HPC+** tt0p9v25c_rctypical	**TSMC 16FFC** tt0p8v85c_rctypical	**TSMC 7** tt0p75v25c_typical
EFLX Core	1x EFLX1K	1x EFLX4K	1x EFLX4K	1x EFLX4K
Max Frequency	131 MHz	521 MHz	701 MHz	794 MHz
Throughput	1.04 GB/s	4.16 B/s	5.60 GB/s	6.35 GB/s

6.4 18x18 Signed Multiply

The formula used is as follows:

out = a*b

The multiply width was 18bit x 18bit = 36bits. The 18th bit is a sign bit.

The RTL was written to emulate a fully-pipelined MAC with registered inputs and outputs. The reason for dong this is so that the Compiler would time the design correctly. The EFLX Compiler's static timing analysis computes delays between RBB flip-flops. If the I/Os are not registered and only going to 1 DSP MAC, then no timing will be reported.

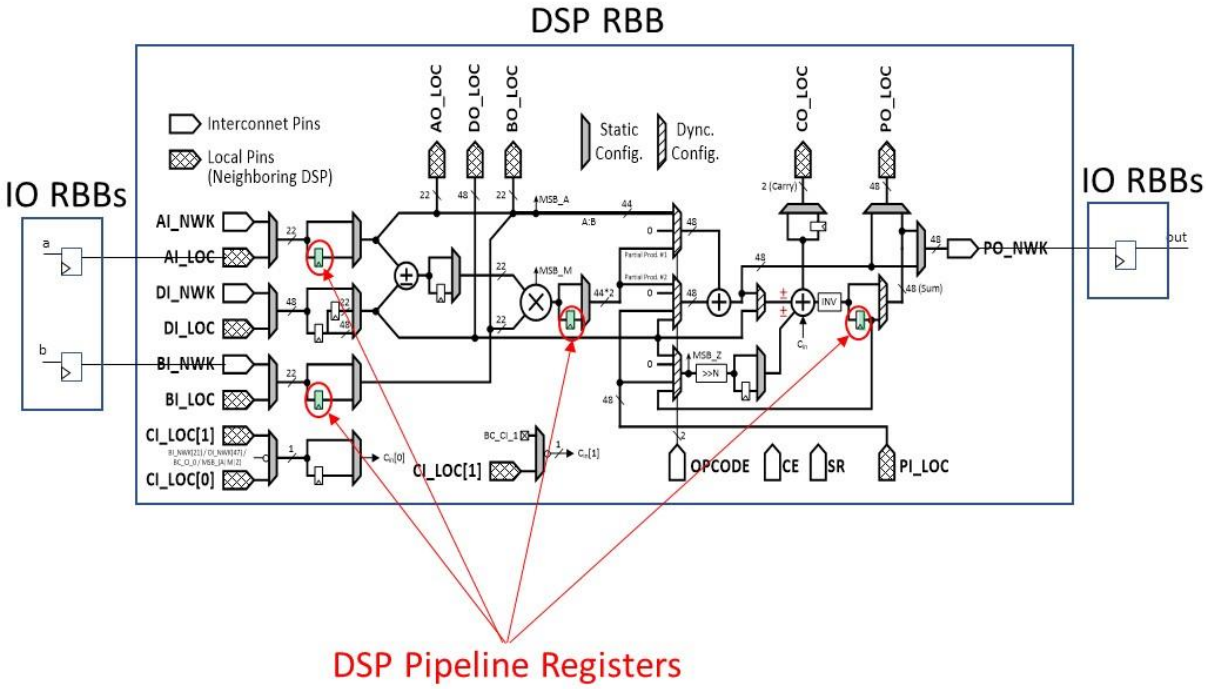


Figure 19 Fully pipelined multiply function.

Multiply function RTL

```
/* 18x18 signed mult reg ios*/
module signed_multiply #(
  parameter DATA_WIDTH = 18
)(
  input signed [DATA_WIDTH-1:0] a,
  input signed [DATA_WIDTH-1:0] b,
  input clk,
  input rst_n,
  output reg signed [2*DATA_WIDTH-1:0] result
);

  reg signed [DATA_WIDTH-1:0] a_reg, b_reg;

  always @(posedge clk) begin
    if (!rst_n) begin
      a_reg <= 'sd0;
      b_reg <= 'sd0;
    end
    else begin
      a_reg <= a;
      b_reg <= b;
    end
  end

  always @(posedge clk) begin
    if (!rst_n) begin
      result <= 'sd0;
    end
    else begin
      result <= a_reg * b_reg;
    end
  end

endmodule
```

The above function fits into 1 DSP MAC with registered inputs and outputs as shown below.

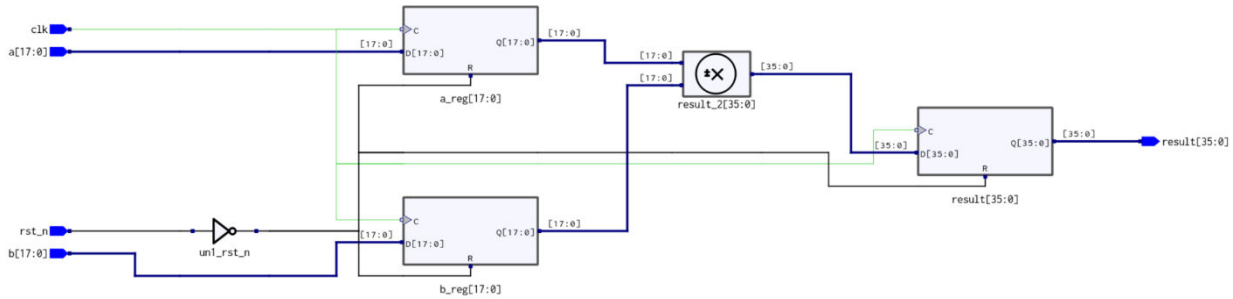


Figure 20 Signed Multiply Schematic

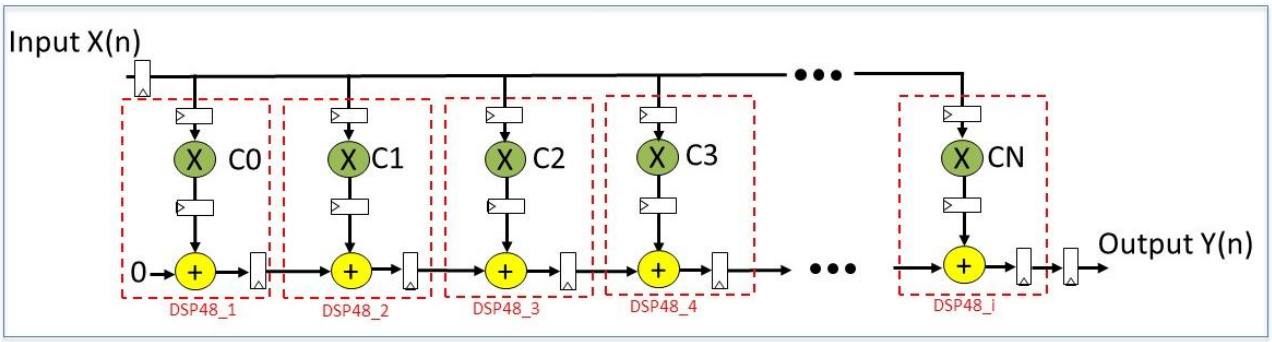
Below is the performance of the 18-bit x 18-bit signed multiply function in different technology nodes.

Table 6 Signed multiply performance.

Technology Corner	**TSMC 28HPC+** tt0p9v25c_rctypical	**TSMC 16FFC** tt0p8v85c_rctypical	**TSMC 7** tt0p75v25c_typical
EFLX Core	1x EFLX4K DSP	1x EFLX4K DSP	1x EFLX4K DSP
Max Frequency from EFLX Compiler	846 MHz	1.31 GHz	1.35 GHz

6.5 FIR Filters

Digital filter algorithms are primarily composed of multipliers, adders, and registers. The basic structure of a Finite Impulse Response (FIR) filter is shown in Fig. 24. There are two FIR filter architectures to chose from: direct form and transposed form. The transposed-form architecture was chosen to best fit the DSP architecture of the embedded FPGA. In a transposed-form FIR, data samples are applied in parallel to all the tap multipliers through pipeline registers. The products are applied to a cascaded chain of registered adders, combining the effect of accumulators and registers. The tap coeficients were programmed into the FPGA fabric directly to minimize FPGA resources (i.e. registers and LUTs), instead of using registers to store the coeficients.



Y[n]=C0*X[n] + C1*X[n-1] + C2*X[n-2] + C3*X[n-3] + C4*X[n-4] … C(N-1)*X[n-N+1]

Figure 21 N-TAP FIR filter transposed form architecture.

Three FIR filter test cases are used to evaluate performance:

- 5-tap asymmetric FIR filter,
- 21-tap symmetric FIR filter,
- 40-tap asymmetric FIR filter.

These three FIR filters were chosen by customer evaluation use cases.

There are three critical timing paths associated with FIR filter test cases:

- DSP-to-DSP via an interconnect fabric,
- Cascaded DSP without an interconnect fabric,
- DSP-to-I/O-Logic FF.

All the FIR filter test cases are based on designs synthesized using Synplify Pro with the pipelining option enabled for DSPs. Each filter is designed so that the inputs and outs are all registered.

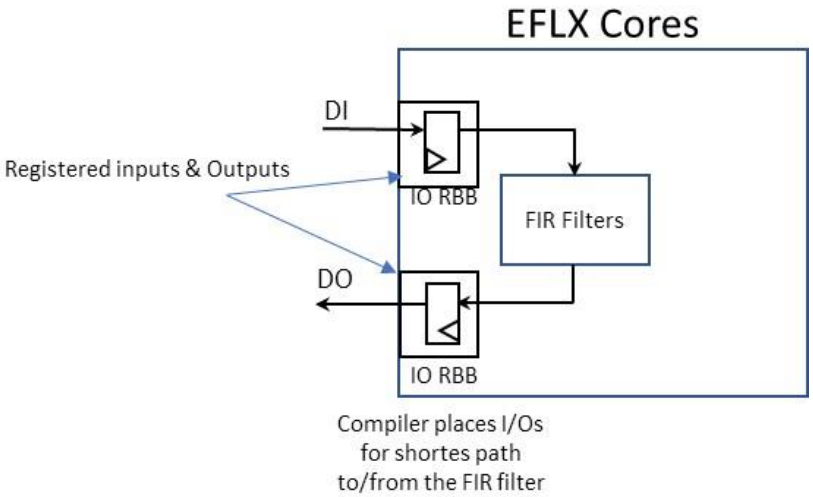


Figure 22 Filter I/Os are registered.

The multiplication widths for all the filters used were 16bits x 16bits = 32bits.

6.5.1 5-tap FIR Filter

The 5-tap FIR filter test case is shown below. The coefficients were programmed into the embedded FPGA fabric thus not requiring separate registers to hold the coefficient values.

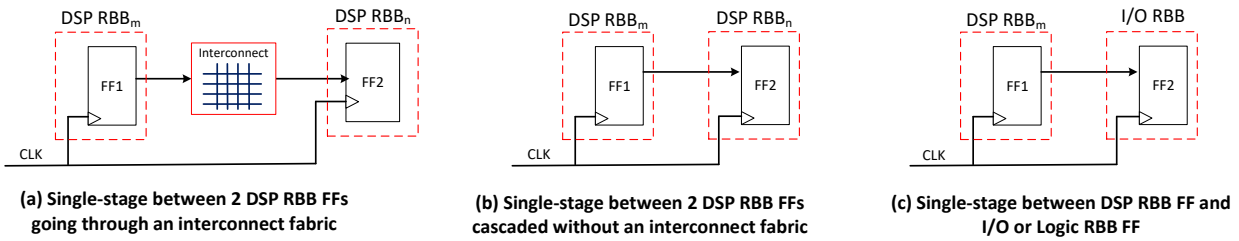
5-tap FIR Filter RTL

```
module FIR_FILTER5 (CLK, DI, DO);

parameter  WL_I  = 16     ; // Input  Wordlength
parameter  WL_O  = 32       ; // Output Wordlength
parameter  TAP   = 5        ; // Number of taps
localparam HTP   = TAP      ; // Number of "symmetric"
taps

input      CLK ;
input signed [WL_I-1:0] DI  ;

wire signed [16:0] CI [0:4];
assign CI[0] = 30611 ;
assign CI[1] = -43810 ;
assign CI[2] = -57075 ;
assign CI[3] = 100010  ;
assign CI[4] =  293097 ;

output reg signed [WL_O-1:0] DO, DOr ;

reg signed [WL_I+17-1:0] Mul_w [0:HTP-1] ;
reg signed [WL_I+17-1:0] Acc_r [0:TAP-1] ;

reg signed [WL_I-1:0] DI_I;
```

```
always@(posedge CLK )
   DI_I <= DI;

genvar i ;
generate for(i=0; i<HTP; i=i+1) begin : HTP_block
   always@(posedge CLK) Mul_w[i] <= DI_I * CI[i] ;
  end
endgenerate

generate for(i=0; i<TAP; i=i+1) begin : TAP_block
   if(i==0) begin : TAP_block_if
    always@(posedge CLK) Acc_r[i] <= Mul_w[i] ;
   end else begin : TAP_block_else
    always@(posedge CLK) Acc_r[i] <= Mul_w[i] +
Acc_r[i-1] ;
   end
  end
endgenerate

always@(posedge CLK)
begin
DOr <= Acc_r[TAP-1][WL_I+17-1:WL_I+17-WL_O] ;
DO <= DOr;
End

endmodule
```

The EFLX Compiler utilization results are as follows:

```
#### Resource Utilization Report #####

  Top level module name: FIR_FILTER5

  Resource usage:
    5 DSP RBBs
   24 I/O RBBs
       16  Input pins
       16  Input FFs utilized
       32 Output pins
       32 Output FFs utilized
    1  Clock domain
      Clock 0: net    34, CLK_c, Fanout: 29 RBBs
```



(a) Single-stage between 2 DSP RBB FFs going through an interconnect fabric

(b) Single-stage between 2 DSP RBB FFs cascaded without an interconnect fabric

(c) Single-stage between DSP RBB FF and I/O or Logic RBB FF

Figure 23 5-tap FIR Filter critical paths.

The following is the performance of the 5-tap FIR filter in different TSMC technology nodes.

Table 7 5-tap FIR Filter performance

Technology Corner	**TSMC 28HPC+** tt0p9v25c_rctypical	**TSMC 16FFC** tt0p8v85c_rctypical	**TSMC 7** tt0p75v25c_typical
EFLX Cores	1x EFLX4K DSP	1x EFLX4K DSP	1x EFLX4K DSP
Max Frequency	755 MHz	1.21 GHz	1.32 GHz

The EFLX4K DSP core has 10 DSPs placed next to each other. So, 5 of the DSPs can be cascaded for the 5-tap FIR filter without going through the interconnect fabric reducing the delay bewteen DSPs which in turn improves performance.

6.5.2 21-tap Symmetric FIR Filter

The 21-tap symmetric FIR filter test case is shown below. The coefficients were programmed into the embedded FPGA Array similar to the 5-tap FIR filter thus not requiring separate registers to hold the coefficient values. Since it is a symmetric FIR filter, only 11 coefficients are required.

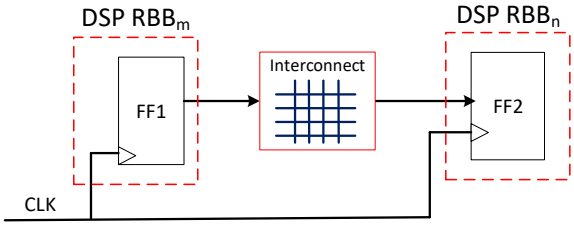
21-tap Symmetric FIR Filter RTL

```
module FIR_FILTER21 (CLK, DI, DO);

par
ameter  WL_I  = 16  ; // Data wordlength IN
parameter  WL_O  = 32 ; // Data wordlength OUT
parameter  CWL   = 16 ; // Coef wordlength
parameter  TAP   = 21 ; // Number of taps
localparam HTP   = (TAP-1)/2 ; // Number of
"symmetric" taps

input  CLK; input      [WL_I-1:0] DI;
output reg [WL_O-1:0] DO;

reg [WL_O-1:0] DO, DO_reg;
reg [WL_O-1:0] Acc_r [0:TAP-1] ;
reg [WL_I-1:0] DI_reg_FIR;

always @ (posedge CLK)
begin
            DO_reg   <= Acc_r [TAP-1] ;
            DI_reg_FIR<=DI;
            DO<=DO_reg;
End

genvar i ;
wire signed [CWL-1:0] Coef_w [0:HTP] ;
//initial begin
//  $readmemb("./FIR_COEF.txt", Coef_w, 0, HTP-1) ;
//end

// 16 bit coefficients

assign Coef_w[0]  =  -1318 ;
assign Coef_w[1]  =  -3829 ;
assign Coef_w[2]  =  -4009 ;
assign Coef_w[3]  =   -717 ;
assign Coef_w[4]  =   3359 ;
assign Coef_w[5]  =   2177 ;
assign Coef_w[6]  =  -3706 ;
assign Coef_w[7]  =  -5613 ;
assign Coef_w[8]  =   4154 ;
assign Coef_w[9]  =  20372 ;
assign Coef_w[10] =  28471 ;
reg [WL_I +CWL-1:0] Mul_w      [0:HTP] ;
reg [WL_O    -1:0] Mul_Trunc_w [0:TAP-1] ;

generate for(i=0; i<HTP; i=i+1) begin
  always@(posedge CLK) Mul_w[i]           <= Coef_w[i]
* DI_reg_FIR ;

  always@(posedge CLK) Mul_Trunc_w[      i] <=
Mul_w[i][WL_I+CWL-1 : WL_I+CWL-WL_O] ;
  always@(posedge CLK) Mul_Trunc_w[TAP-1-i] <=
Mul_w[i][WL_I+CWL-1 : WL_I+CWL-WL_O] ;
 end
endgenerate
            // middle tap
  always@(posedge CLK) Mul_w[HTP]          <=
Coef_w[HTP] * DI_reg_FIR ;
  always@(posedge CLK) Mul_Trunc_w[    HTP] <=
Mul_w[HTP][WL_I+CWL-1 : WL_I+CWL-WL_O] ;

//reg [WL_O-1:0] Acc_r [0:TAP-1] ;

generate for(i=0; i<TAP; i=i+1) begin
   if(i==0) always@(posedge CLK) Acc_r[i] <=
Mul_Trunc_w[i] ;
   else     always@(posedge CLK) Acc_r[i] <=
Mul_Trunc_w[i] + Acc_r[i-1] ;
  end
endgenerate

endmodule
```

The EFLX Compiler utilization results are as follows:

EFLX Gen 2 – TSMC 16 FFC

```
####   Resource Utilization Report   ####

   Top level module name: fir_filter_21tap_sym

   Resource usage:
     21 DSP RBBs
      2 Logic 6-LUT RBBs
          8 RBB LUTs utilized (up to 4 per RBB)
          8 RBB FFs utilized (up to 4 pairs per RBB)
     24 I/O RBBs
         16  Input pins
         16  Input FFs utilized
         32 Output pins
         32 Output FFs utilized
      1 Clock
         Clock 0: net    50, CLK_c, Fanout: 47 RBBs
   need 3 DSP and 0 Logic-Memory EFLX 1K Tiles.
   or...
   need 1 DSP and 0 Logic-Memory EFLX 4K Tiles.


####   Resource Utilization Complete   ####
```



Single-stage between 2 DSP RBB FFs going through an interconnect fabric

Figure 24 21-Tap FIR filter critical path

The following is the performance for the 21-tap symmetric FIR filter in different TSMC technology nodes.

Table 8 21-tap symmetric FIR Filter performance.

Technology Corner	**TSMC 28HPC+** tt0p9v25c_rctypical	**TSMC 16FFC** tt0p8v85c_rctypical	**TSMC 7** tt0p75v25c_typical
EFLX Cores	1x EFLX4K DSP	1x EFLX4K DSP	1x EFLX4K DSP
Max Frequency	462 MHz	651 MHz	691 MHz

6.5.3 40-tap Non-symmetric FIR Filter

The 40-tap non-symmetric FIR filter test case is shown below. The coefficients were programmed into the embedded FPGA Array similar to the 5-tap FIR filter and the 21-tap symmetric FIR filters thus not requiring separate registers to hold the coefficient values.

40-tap Non-symmetric FIR Filter RTL

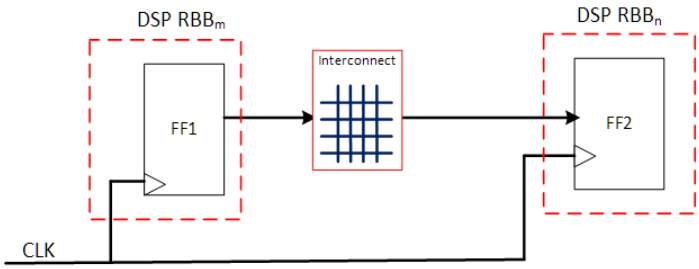
```
module FIR_FILTER40 (DO, DI, CLK);

parameter  DWL_IN  = 16 ; // Data wordlength
parameter  DWL_OUT = 32 ; // Data wordlength OUT
parameter  CWL = 16    ; // Coef wordlength
parameter  TAP = 40    ; // Number of taps
localparam HTP = TAP   ; // Number of "symmetric"
taps

input              CLK ;
input   [DWL_IN-1:0] DI;
output [DWL_OUT-1:0] DO ;            // Final output




//reg  [1       :0] DI_reg, DI_reg2, DI_OUT;
reg  [DWL_OUT-1:0] DO_reg ;
reg  [DWL_OUT-1:0] DO ;

reg  [DWL_IN-1 :0] DI_reg_FIR;

reg  [3:0] pattern_counter;

reg  [DWL_IN +CWL-1:0] Mul_w      [0:HTP-1] ;
wire [DWL_IN +CWL-1:0] Mul_Trunc_w        [0:TAP-1] ;
reg  [DWL_IN +CWL-1:0] Acc_r                      [0:TAP-1] ;



always @ (posedge CLK)
begin
            DI_reg_FIR<=DI;
            DO_reg   <= Acc_r [TAP-1][DWL_IN+CWL-1 :
DWL_IN+CWL-DWL_OUT] ;
            DO       <= DO_reg ;
            end
```

```
genvar i ;
wire signed [CWL-1:0] Coef_w [0:HTP-1] ;
assign Coef_w[0] = -295 ;
assign Coef_w[1] = -91 ;
assign Coef_w[2] = 295 ;
assign Coef_w[3] = 82 ;
assign Coef_w[4] = -358 ;
assign Coef_w[5] = -133 ;
assign Coef_w[6] = 369 ;
assign Coef_w[7] = 134 ;
assign Coef_w[8] = -435 ;
assign Coef_w[9] = -190 ;
assign Coef_w[10] = 457 ;
assign Coef_w[11] = 204 ;
assign Coef_w[12] = -529 ;
assign Coef_w[13] = -267 ;
assign Coef_w[14] = 565 ;
assign Coef_w[15] = 297 ;
assign Coef_w[16] = -648 ;
assign Coef_w[17] = -373 ;
assign Coef_w[18] = 703 ;
assign Coef_w[19] = 426 ;
assign Coef_w[20] = -805 ;
assign Coef_w[21] = -526 ;
assign Coef_w[22] = 891 ;
assign Coef_w[23] = 615 ;
assign Coef_w[24] = -1030 ;
assign Coef_w[25] = -761 ;
assign Coef_w[26] = 1172 ;
assign Coef_w[27] = 919 ;
assign Coef_w[28] = -1391 ;
assign Coef_w[29] = -1167 ;
assign Coef_w[30] = 1663 ;
assign Coef_w[31] = 1494 ;
assign Coef_w[32] = -2104 ;
assign Coef_w[33] = -2052 ;
assign Coef_w[34] = 2821 ;
assign Coef_w[35] = 3060 ;
assign Coef_w[36] = -4380 ;
assign Coef_w[37] = -5707 ;
assign Coef_w[38] = 10001 ;
assign Coef_w[39] = 29309 ;

generate for(i=0; i<HTP; i=i+1) begin : HTP_block
  always@(posedge CLK) Mul_w[i]          <= Coef_w[i]
* DI_reg_FIR;
  assign Mul_Trunc_w[i] = Mul_w[i];
 end
endgenerate
generate for(i=0; i<TAP; i=i+1) begin : TAP_block
```

	if(i==0) begin : TAP_block_if always@(posedge CLK) Acc_r[i] <= Mul_Trunc_w[i] ; end else begin : TAP_block_else always@(posedge CLK) Acc_r[i] <= Mul_Trunc_w[i] + Acc_r[i-1] ; end end endgenerate endmodule

The EFLX Compiler utilization results are as follows:

```
#### Resource Utilization Report ####

  Top level module name: FIR_FILTER40

  Resource usage:
   40 DSP RBBs
   24 I/O RBBs
      16 Input pins
      16 Input FFs utilized
      32 Output pins
      32 Output FFs utilized
    1 Clock domain
     Clock 0: net   66, CLK_c, Fanout: 64 RBBs
```



Single stage between 2 DSP RBB FFs going through an interconnect fabric

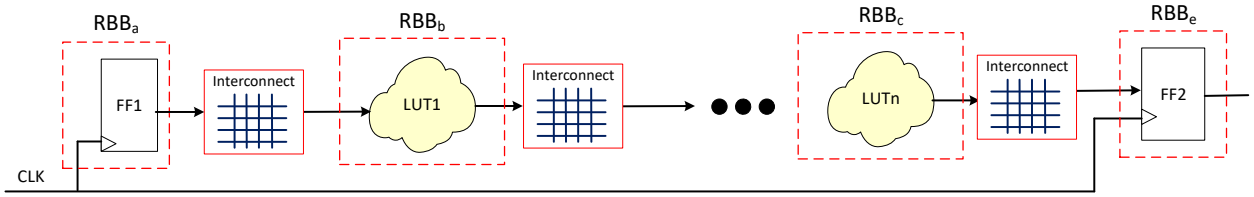
Figure 25 40-tap FIR filter critical path

The following is the performance for the 40-tap non-symmetric FIR filter in different TSMC technology nodes.

Table 9 40-tap non-symmetric FIR Filter performance.

Technology Corner	**TSMC 28 HPC+** tt0p9v25c_rctypical	**TSMC 16 FFC** tt0p8v85c_rctypical	**TSMC 7** tt0p75v25c_typical
EFLX Cores	1x EFLX4K DSP	1x EFLX4K DSP	1x EFLX4K DSP
Max Frequency	460 MHz	635 MHz	695 MHz

7. Multi-Stage Logic



n-LUT stages between FF

Figure 26 n-LUT stages between FF.

Designs that are not pipelined and not optimized for embedded FPGAs usually have multiple LUT stages between flip-flops.

Multi-stage logic RTL:

```
module multi_stage #(parameter STAGES = 5) (
  input wire clk,
  input wire reset_n,
  input wire [7:0] data_in,
  output reg [7:0] data_out
);

  reg [7:0] in_data_reg;
  wire [7:0] out_data_reg[STAGES:0]
/*synthesis syn_keep = 1*/;

  // Register the input data
  always @(posedge clk) begin
    if (reset_n == 1'b0) begin
      in_data_reg <= 8'h0;
    end
    else begin
      in_data_reg <= data_in;
    end
  end

  // Inverter chain
  genvar i;
  generate
    assign out_data_reg[0] = ~in_data_reg;
    for (i = 1; i < STAGES; i = i + 1) begin : inv
      assign out_data_reg[i] = ~out_data_reg[i-1];
    end
  endgenerate

  // Register the output data
  always @(posedge clk) begin
    if (reset_n == 1'b0) begin
      data_out <= 8'h0;
    end
    else begin
      data_out <= out_data_reg[STAGES-1];
    end
  end

endmodule
```

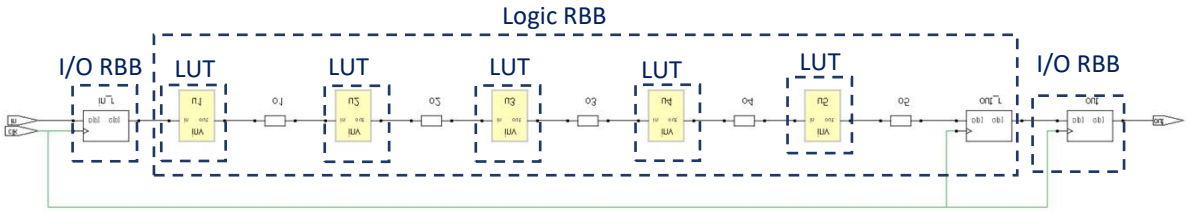


Figure 27 Schematic view of multi-stage logic (5 stages in this example).

In the above example for 5 LUT stages, all the LUT stages reside in the same RBB. Each RBB can have up to 8 of these simple LUT stages. The graph below, Fig. 31, shows the frequency impact per LUT stage for

a simple inverter LUT stage. The slope of the curves below are a function of the length of the interconnect hop and the type of logic being used.

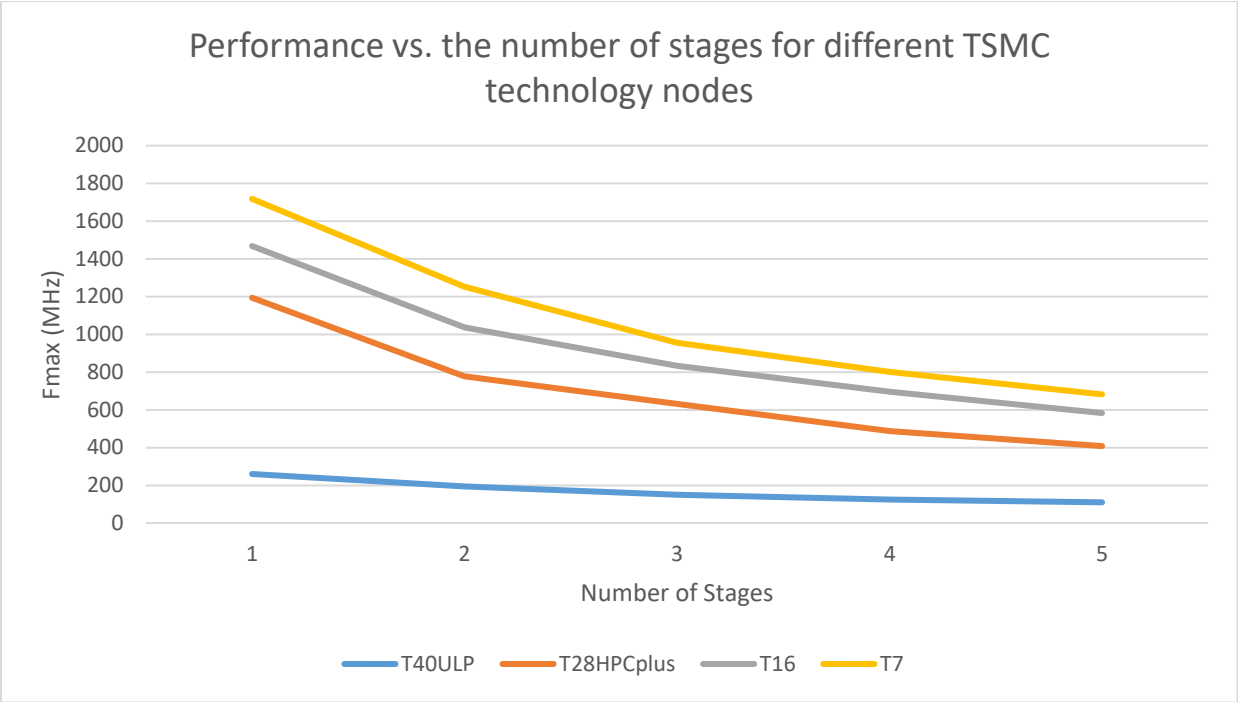


Figure 28 Frequency vs number of LUT stages.

The frequency above is based on TT corner. Performance will vary with different nominal voltage ranges and other corners.

7.1 ARM Cortex-M0

One example of a multi-LUT stage logic design is an ARM Cortex-M0.

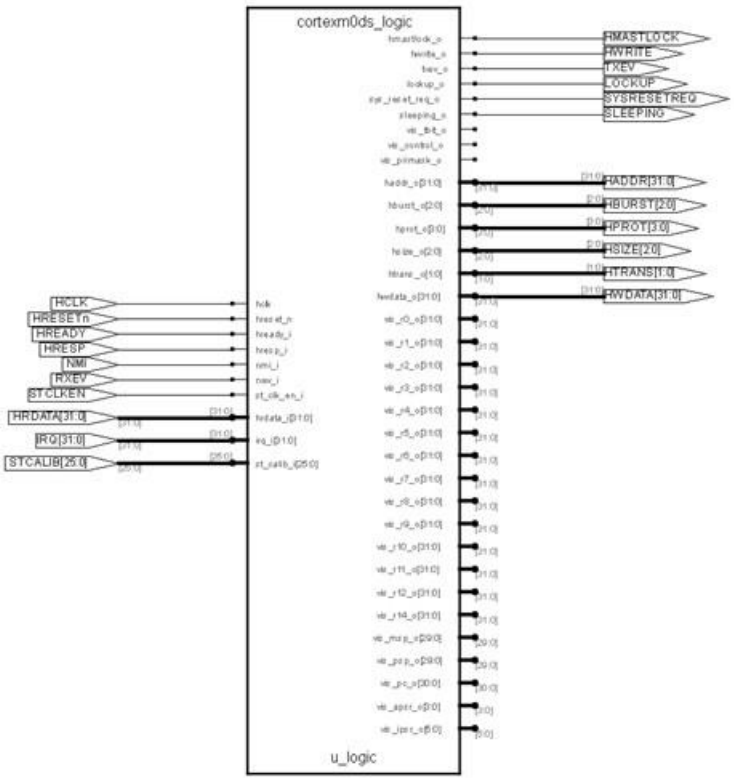


Figure 29 ARM Cortex M0.

The RTL used for the ARM Cortex was the “Cortex-M0 DesignStart Design Kit r1p0”. This design supports fast multiply using 3 DSP RBBs.

The ARM Cortex-M0 was synthesized with pipelining and retiming enabled. The resources required based on the EFLX compiler are shown below.

EFLX Gen 2 – TSMC 16FFC
Resource Utilization Report
Top level module name: CORTEXM0DS
Resource usage: 3 DSP RBBs 833 Logic 6-LUT RBBs 3264 RBB LUTs utilized (up to 4 per RBB) 1235 RBB FFs utilized (up to 4 pairs per RBB) 89 I/O RBBs 96 Input pins 82 Output pins

```
        1 Output FFs utilized
    1 Clock
        Clock 0: net  5134, HCLK_c, Fanout: 411 RBBs
  need 1 DSP and 6 Logic-Memory EFLX 1K Tiles.
  or...
  need 1 DSP and 1 Logic-Memory EFLX 4K Tiles.


####   Resource Utilization Complete   ####
```

Table 10 is the performance for the ARM Cortex-M0 on different TSMC technology nodes.

Table 10 ARM Cortex-M0 performance.

Technology Corner	**TSMC 28HPC+** tt0p9v25c_rctypical	**TSMC 16FFC** tt0p8v85c_rctypical	**TSMC 7** tt0p75v25c_typical
EFLX Cores	1x EFLX4K DSP, 1x EFLX4K LM		
Max Frequency	101 MHz	133 MHz	150 MHz
LUT stages for worst-case timing path	13	11	11

7.2 8051 MCU

The 8051 was synthesized with pipelining enabled. The resources required based on the EFLX Compiler are shown below.

EFLX Gen 2 – TSMC 16 FFC

```
####   Resource Utilization Report   ####

   Top level module name: oc8051_top

   Resource usage:
     1 Block RAMs mapped as 1 external RAMs
         1 of which are 256x8b two-port (1R1W) RAM, mapped to 2048x18b dual-port external RAM
     1 External RAMs
         1 of which are 2048x18b dual-port RAM
           4 dual-port RAM available in RAM-definition file
    577 Logic 6-LUT RBBs
         6 of which require RAM and/or shift-register functions
       1990 RBB LUTs utilized (up to 4 per RBB)
        524 RBB FFs utilized (up to 4 pairs per RBB)
     80 I/O RBBs
         82  Input pins
         32  Input FFs utilized
         78 Output pins
         60 Output FFs utilized
     1 Clock
         Clock 0: net  3296, wb_clk_i_c, Fanout: 382 RBBs
   need 0 DSP and 5 Logic-Memory EFLX 1K Tiles.
   or...
   need 0 DSP and 1 Logic-Memory EFLX 4K Tiles.


####   Resource Utilization Complete   ####
```

The following is the performance for the 8051 MCU on different TSMC technology nodes.

Table 11 8051 MCU performance.

Technology Corner	**TSMC 28HPC** tt0p9v25c_rctypical	**TSMC 16FFC** tt0p8v85c_rctypical	**TSMC7** tt0p75v25c_typical
EFLX Cores	1x EFLX4K Logic		
Max Frequency	114 MHz	154 MHz	170 MHz
LUT stages for worst-case timing path	13	13	10

7.3 AES-128

The AES-128 architecture is shown below, Fig. 33.

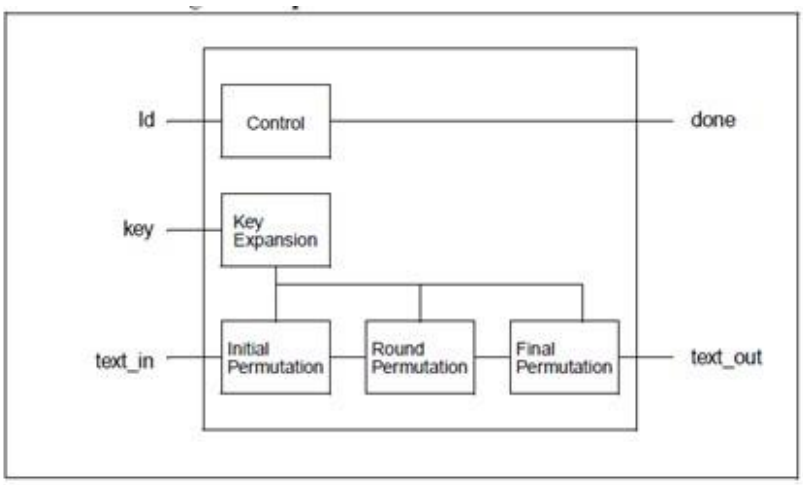


Figure 30 AES-128 architecture.

The latency of encryption is 12 cycles. The throughput = (Max_Freq * 128bits)/12.

The design has all the I/Os registered and was synthesized targeting EFLX Gen-2 Cores.

EFLX Gen 2 – TSMC 16 FFC
Resource Utilization Report
Top level module name: aes_cipher_top
Resource usage:
336 Logic 6-LUT RBBs
1189 RBB LUTs utilized (up to 4 per RBB)
406 RBB FFs utilized (up to 4 pairs per RBB)
194 I/O RBBs
258 Input pins
128 Input FFs utilized
129 Output pins
129 Output FFs utilized
1 Clock
Clock 0: net 1357, clk_c, Fanout: 279 RBBs
need 0 DSP and 3 Logic-Memory EFLX 1K Tiles.
or...
need 0 DSP and 1 Logic-Memory EFLX 4K Tiles.
Resource Utilization Complete

The encryption takes 12 clock cycles, therefore the throughput will be:

Throughput (bits/sec) = (128bits * frequency)/12

Table 12 Performance for AES-128.

Technology Corner	**TSMC 28HPC+** tt0p9v25c_rctypical	**TSMC 16FFC** tt0p8v85c_rctypical	**TSMC 7** tt0p75v25c_typical
EFLX Core	1x EFLX4K Logic		
Max Frequency	394 MHz	532 MHz	584 MHz
LUT stages for worst-case timing path	3	3	2
Throughput	4.20 Gb/s	5.67 Gb/s	6.22 Gb/s

7.4 256-Point FFT

The 256-point FFT is a radix-4, 16-bit fully streaming architecture with 4 data samples per cycle, 72 stages. The latency of the FFT is 678 clock cycles and the gap is 64 clocks. The architecture of the FFT is shown below.

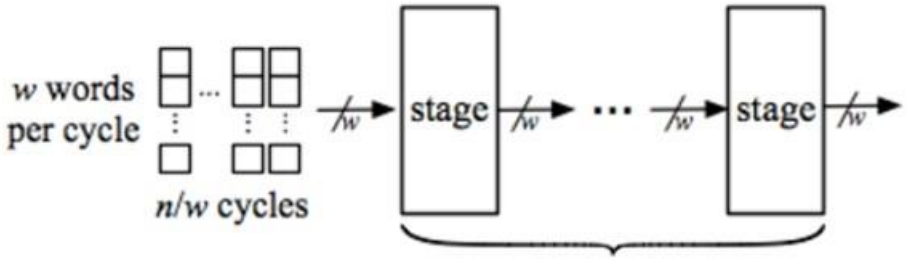


Figure 31 256-point FFT

The EFLX resources required for the FFT is shown below.

EFLX Gen 2 – TSMC 16 FFC
Resource Utilization Report
Top level module name: statementList19975
Resource usage:
16 Block RAMs mapped as 32 external RAMs
16 of which are 128x35b two-port (1R1W) RAM, mapped to 2048x18b dual-port external RAM
each require 2 External RAMs due to:
2x data extension
32 External RAMs
32 of which are 2048x18b dual-port RAM
32 dual-port RAM available in RAM-definition file
2384 Logic 6-LUT RBBs
776 of which require RAM and/or shift-register functions
9089 RBB LUTs utilized (up to 4 per RBB)
5618 RBB FFs utilized (up to 4 pairs per RBB)
66 I/O RBBs
66 Input pins

```
    65  Input FFs utilized
    65 Output pins
    65 Output FFs utilized
  1 Clock
    Clock 0: net 21912, clk_c, Fanout: 2018 RBBs
need 0 DSP and 20 Logic-Memory EFLX 1K Tiles.
or...
need 0 DSP and 5 Logic-Memory EFLX 4K Tiles.


#### Resource Utilization Complete ####
```

The compiler was run with with fp auto generate and the compiler added 32 external RAMs as follows:

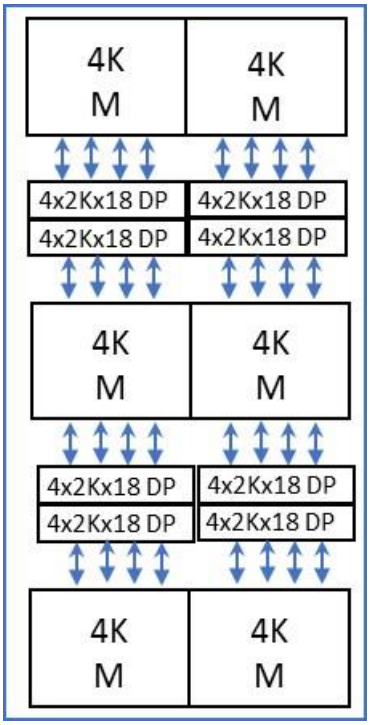


Figure 32 FFT Floor Plan with external RAMs

Table 13 Performance for 256-Point FFT.

Technology Corner	**TSMC 28HPC+** tt0p9v25c_rctypical	**TSMC 16FFC** tt0p8v85c_rctypical	**TSMC 7** tt0p75v25c_typical
EFLX Core	6x EFLX4K LM Tile		
Max Frequency	302 MHz	408 MHz	473 MHz
LUT stages for worst-case timing path	3	2	2
Throughput	2.41 GB/s	3.26 GB/s	3.78 GB/s

7.5 SHA-256

The SHA-256 uses a 32-bit input and 32-bit output data interface. The core used supports 512-bit message, 256-bit digest and 65 clock latency for the hash algorithm.

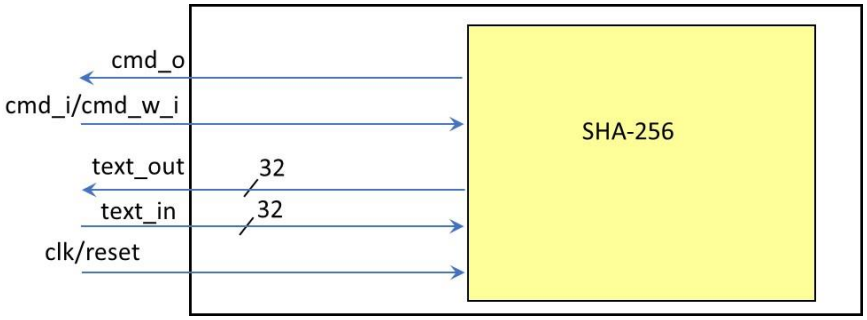


Figure 33 SHA-256 core

The EFLX resources required for the SHA-256 is shown below.

EFLX Gen 2 – TSMC 16 FFC
Resource Utilization Report
Top level module name: sha256
Resource usage:
527 Logic 6-LUT RBBs
1660 RBB LUTs utilized (up to 4 per RBB)
806 RBB FFs utilized (up to 4 pairs per RBB)
37 I/O RBBs
37 Input pins
36 Output pins
36 Output FFs utilized
1 Clock
Clock 0: net 2307, clk_i_c, Fanout: 402 RBBs
need 0 DSP and 4 Logic-Memory EFLX 1K Tiles.
or...
need 0 DSP and 1 Logic-Memory EFLX 4K Tiles.
Resource Utilization Complete

Table 14 Performance for SHA-256.

Technology Corner	**TSMC 28HPC+** tt0p9v25c_rctypical	**TSMC 16FFC** tt0p8v85c_rctypical	**TSMC 7** tt0p75v25c_typical
EFLX Core	1 EFLX4K LM Tile		
Max Frequency	173 MHz	251 MHz	270 MHz
LUT stages for worst-case timing path	11	7	10

7.6 JPEG

The JPEG architecture is shown below.

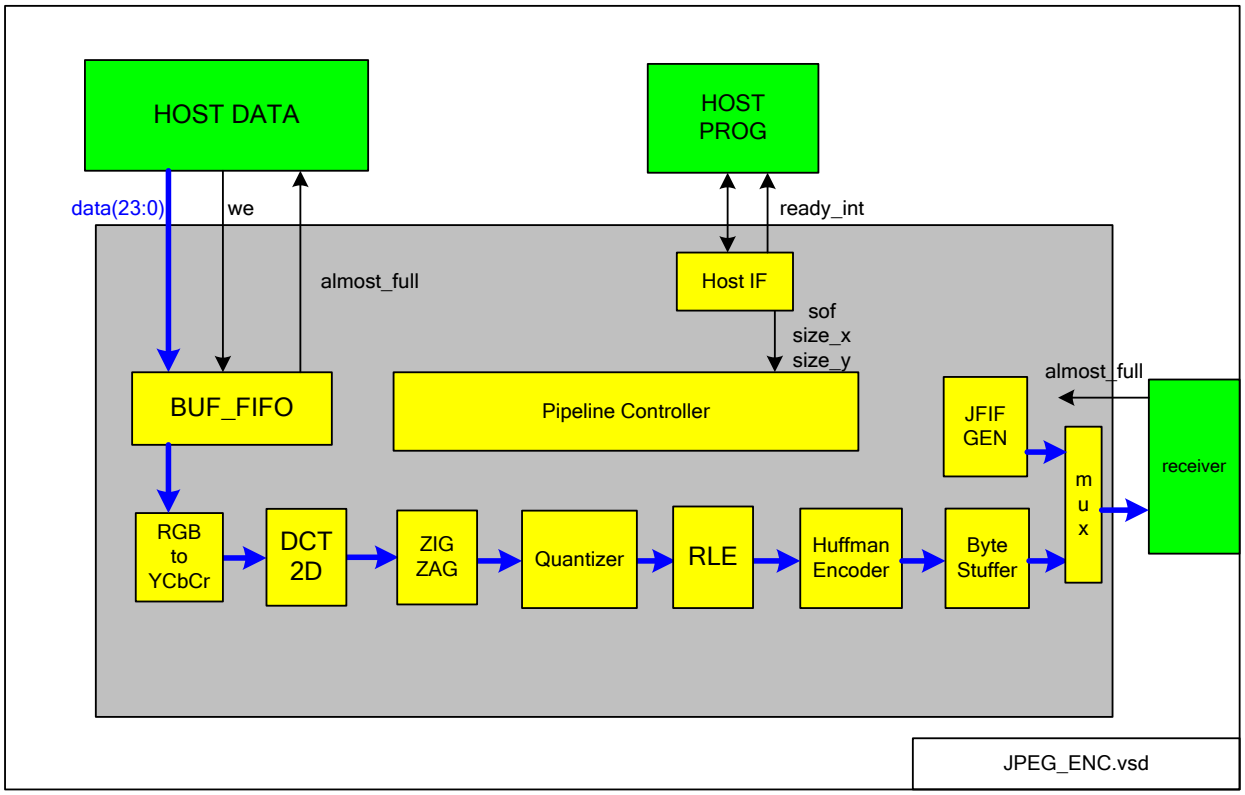


Figure 34 JPEG Architecture

The EFLX resources required for the JPEG is shown below.

EFLX Gen 2 – TSMC 16 FFC
Resource Utilization Report
Top level module name: jpeg_top
Resource usage:
2 Block RAMs mapped as 4 external RAMs
1 of which are 256x21b single-port RAM, mapped to 2048x18b dual-port external RAM
each require 2 External RAMs due to:
2x data extension
1 of which are 256x21b dual-port RAM, mapped to 2048x18b dual-port external RAM

```
                    each require 2 External RAMs due to:
                        2x data extension
              4 External RAMs
                  4 of which are 2048x18b dual-port RAM
                    80 dual-port RAM available in RAM-definition file
             227 DSP RBBs
            5843 Logic 6-LUT RBBs
                 119 of which require RAM and/or shift-register functions
               22987 RBB LUTs utilized (up to 4 per RBB)
               20835 RBB FFs utilized (up to 4 pairs per RBB)
              34 I/O RBBs
                  27  Input pins
                  39 Output pins
                  39 Output FFs utilized
              1 Clock
                  Clock 0: net 41346, clk_c, Fanout: 5845 RBBs
            need 23 DSP and 26 Logic-Memory EFLX 1K Tiles.
            or...
            need 6 DSP and 5 Logic-Memory EFLX 4K Tiles.


####   Resource Utilization Complete   ####
```

The compiler was run with 4x3 floorplan (two columms of DSP tiles and two columns of LM tiles) and 80 external RAMs as follows:

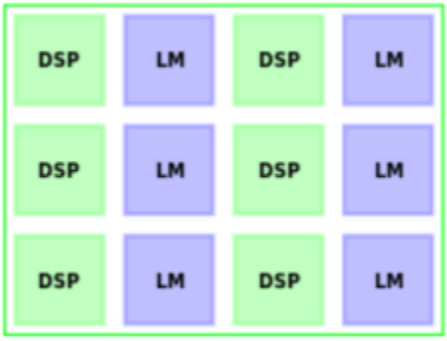


Figure 35 JPEG Encoder Floorplan

Table 15 Performance for JPEG.

Technology **Corner**	**TSMC 28HPC+** tt0p9v25c_rctypical	**TSMC 16FFC** tt0p8v85c_rctypical	**TSMC 7** tt0p75v25c_typical
EFLX Core	6x EFLX4K LM tiles, 6x EFLX4K DSP Tiles		
Max Frequency	219 MHz	296 MHz	349 MHz
LUT stages for worst-case timing path	6	1	2

8. Summary

Below is a summary of performance benchmarks discussed in this application note, and online resources.

8.1 Performance Benchmarking on EFLX Cores

Technology Corner	TSMC 40 ULP tt1p1v25c_Typical		TSMC 28HPC+ tt0p9v25c_rctypical		TSMC 16FFC tt0p8v85c_rctypical		TSMC N7 tt0p75v25c_typical	
EFLX Core	EFLX1K		EFLX4K		EFLX4K		EFLX4K	
	Floor plan	Freq (MHz)	Floor plan	Freq (MHz)	Floor plan	Freq (MHz)	Floor plan	Freq (MHz)
Single Stage- short	1K M	278	4K M	1,264	4K M	1,773	4K M	2,016
Single Stage- long	1K M	166	4K M	493	4K M	634	4K M	741
Single Stage Pipelined- short	1K M	338	4K M	2,016	4K M	2,762	4K M	2,865
Single Stage Pipelined- long	1K M	238	4K M	828	4K M	1,124	4K M	1,250
Virtual switch -straight through	1K M	407	4K M	2,380	4K M	2,923	4K M	3,436
Virtual switch- crossed	1K M	407	4K M	2,364	4K M	2,923	4K M	3,412
16bit counter	1K M	173	4K M	780	4K M	1,072	4K M	1,231
Wide Comparator	1K M	131	4K M	521	4K M	701	4K M	794
18x18 mult			4K DSP	846	4K DSP	1,312	4K DSP	1,358
5 tapfir			4K DSP	755	4K DSP	1,216	4K DSP	1,329
21 tap fir			4K DSP	462	4K DSP	651	4K DSP	691
40 tap fir			4K DSP	460	4K DSP	635	4K DSP	695
ARM Cortex M0			4K DSP M	101	4K DSP M	133	4K DSP M	150
8051			4K M	114	4K M	154	4K M	170
aes 128			4K M	394	4K M	532	4K M	584
FFT-256			4K 2x3 M	302	4K 2x3 M	408	4K 2x3 M	473
SHA-256			4K M	173	4K M	251	4K M	270
JPEG Encoder			4K 6xDSP 6xM	219	4K 6xDSP 6xM	296	4K 6xDSP 6xM	349

The performance for many other corners are also available upon request.

8.2 Online Resources

RTL code for desgin examples described in this document is available for download from:
www.flex-logix.com/PUT_PAGE_HERE