

# طراحی و پیاده‌سازی ضرب‌کننده ماتریس توسط Verilog

احمد سلیمی<sup>۱</sup>، کیمیا نوربخش<sup>۱</sup>، ساعی سعادت<sup>۱</sup>، علیرضا حسین‌پور<sup>۱</sup>

<sup>۱</sup> دانشگاه صنعتی شریف، دانشکده مهندسی کامپیوتر

چکیده—

کلمات کلیدی—

## ۱. مقدمه

هدف این پروژه، طراحی، شبیه‌سازی و سنتز یک ضرب‌کننده ماتریس با زبان ورپلاگ است. از کاربردهای چنین ضرب‌کننده‌ای می‌توان به آموزش مدل‌های شبکه‌های عصبی<sup>۱</sup> اشاره کرد. از آنجایی که ماتریس وزن‌ها در شبکه‌های عصبی ابعاد بالایی دارند، طراحی یک ضرب‌کننده که هم بتواند در زمان کوتاه پاسخ را آماده کند و هم از نظر سخت افزاری بهینه باشد، از اهمیت بالایی برخوردار است.

روشی که در این پروژه به کار برده شده است، تقسیم بندی ماتریس به بلوک‌های مربعی و استفاده از خاصیت بلوکی در ضرب ماتریس‌ها است. ساختار این ضرب‌کننده از ۴ لایه تشکیل شده است. بالاترین لایه، لایه ضرب‌کننده ماتریس موازی است، لایه بعدی ضرب‌کننده ماتریس سطری و ستونی است. لایه سوم ضرب‌کننده ماتریس ترتیبی و لایه آخر ضرب‌کننده و جمع‌کننده floating point است.

در ادامه و در بخش ۲، ابتدا به معماری سیستم و نحوه طراحی ماژول‌های آن پرداخته می‌شود. در بخش ۳، به نحوه انجام شبیه‌سازی‌ها و نتایج حاصل از آن‌ها پرداخته شده است و در نهایت، در بخش ۴، به نحوه انجام عملیات سنتز این سیستم روی FPGA و نتایج حاصل اشاره شده است.

## ۲. معماری سیستم

معماری این سیستم، از سه لایه اصلی تشکیل شده است. در ادامه، معماری و جزئیات هر یک از این لایه‌ها، توضیح داده شده است.

برای ارتباط بین تمامی ماژول‌ها، برای اطمینان از این که ورودی و خروجی‌ها هنگام استفاده شدن تغییر نمی‌کنند و

مقدار صحیحی دارند، برای هر کدام دو سیگنال stable و acknowledge در نظر می‌گیریم. نحوه‌ی استفاده از آن‌ها بدین گونه است که ماژولی که مقدار را دارد و می‌خواهد آن را پاس بدهد، با استفاده از سیگنال stable به ماژول گیرنده اعلام می‌کند که ورودی آماده‌ی استفاده است، سپس ماژول گیرنده با استفاده از سیگنال acknowledge اعلام می‌کند که ورودی را با موفقیت دریافت کرده و ماژول فرستنده می‌تواند آن را تغییر دهد.

### ۲.۱. ضرب‌کننده ماتریس ترتیبی

در این ماژول مانند ضرب ماتریسی عادی، دو ماتریس  $m \times m$  را در هم ضرب می‌کنیم. می‌دانیم که برای به دست آوردن درایه  $ij$  حاصلضرب، باید سطر  $i$  ام ماتریس اول را در ستون  $j$  ام ماتریس دوم ضرب کنیم. برای این موضوع به ازای هر  $1 \leq j \leq m, 1 \leq i \leq m$ ، داریم:

$$R_{ij} = \sum_{k=1}^m A_{ik} \times B_{kj}$$

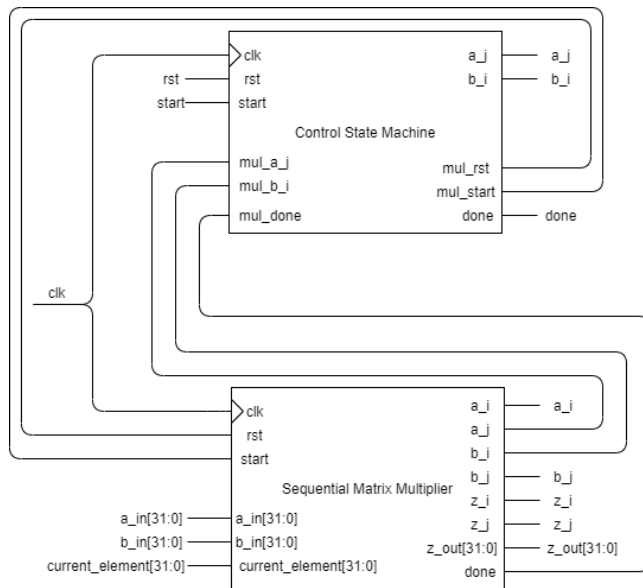
که در آن،  $R$  ماتریس  $m \times m$  حاصل ضرب است. در این ماژول برای محاسبه جمع و ضرب‌ها، از ماژول‌های جمع‌کننده و ضرب‌کننده اعشاری<sup>۲</sup> استفاده می‌کنیم. ماژول ضرب‌کننده ماتریس ترتیبی<sup>۳</sup> این فرایند را در قالب یک ماشین حالت انجام می‌دهد. برای محاسبه درایه  $i, j$  ام، یک accumulator برای نگهداری جواب نهایی در نظر می‌گیریم و سپس به ازای هر  $k$ ، ابتدا با استفاده از ماژول FP\_multiplier حاصل  $A_{ik} \times B_{kj}$  را محاسبه می‌کنیم و با استفاده از ماژول FP\_Adder، به ازای  $k$  های مختلف جواب را آپدیت می‌کنیم.

شکل ۱ بلوک دیاگرام ضرب‌کننده ماتریس ترتیبی را نشان می‌دهد. باید توجه کرد که حافظه‌ای که حاوی ماتریس‌های ورودی و ماتریس جواب است، در خارج این ماژول قرار دارد. در نتیجه، واحد کنترل در این ماشین

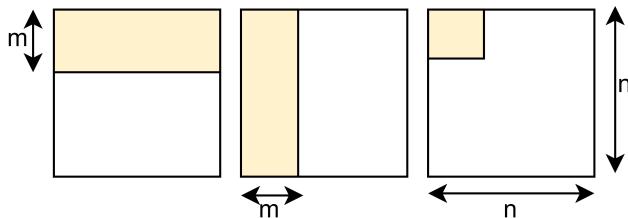
<sup>۲</sup>floatng point adder and multiplier

<sup>۳</sup>sequential matrix multiplier

<sup>۱</sup>Neural Networks



شکل ۲: بلوک دیاگرام ضرب‌کننده ماتریس سطری در ستونی.



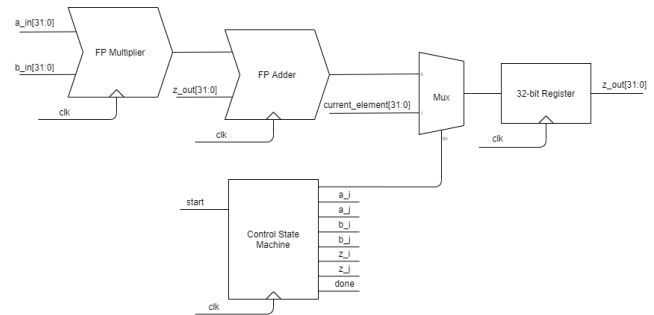
شکل ۳: روش محاسبه حاصل ضرب یک بلوک از ماتریس جواب، توسط یک ضرب‌کننده ماتریس سطری در ستونی.

ضرب است، و  $a_{jseq}$  و  $b_{iseq}$  اندیس‌هایی هستند که ضرب‌کننده ماتریس ترتیبی تعیین کرده است.

### ۲.۳. ضرب‌کننده ماتریس موازی

این ماژول دو ماتریس  $n \times n$  را به صورت موازی در هم ضرب می‌کند. به این صورت که ابتدا این ماتریس، به  $\lceil \frac{n}{m} \rceil^2$  ماتریس  $m \times m$  تقسیم‌بندی می‌شود. سپس مطابق شکل ۳، هر بلوک  $m \times m$  از ماتریس حاصل ضرب، توسط یک ضرب‌کننده ماتریس سطری در ستونی، بصورت موازی با دیگر بلوک‌های حاصل ضرب، محاسبه می‌شود. در نتیجه، در این لایه، به تعداد  $\lceil \frac{n}{m} \rceil^2$  ضرب‌کننده ماتریس سطری در ستونی ساخته می‌شود.

شکل ۴ بلوک دیاگرام ضرب‌کننده ماتریس موازی را نمایش می‌دهد. حافظه ماتریس‌های ورودی و خروجی در



شکل ۴: بلوک دیاگرام ضرب‌کننده ماتریس ترتیبی.

حالت محدود<sup>۴</sup> اندیس‌های درایه‌های موردنیاز خود، یعنی  $a_i, a_j, b_i, b_j, z_i, z_j$  را تعیین می‌کند، و مقادیر مربوط به هر درایه در ماتریس‌های ورودی در  $a_{in}$  و  $b_{in}$  قرار گرفته، و مقدار  $z_{out}$  نیز در ماتریس جواب قرار داده می‌شود.

### ۲.۴. ضرب‌کننده ماتریس سطری در ستونی

وظیفه این ماژول، این است که با استفاده از یک ماژول ضرب‌کننده ماتریس ترتیبی، حاصل ضرب یک ماتریس سطری  $m \times n$  در یک ماتریس ستونی  $n \times m$  را محاسبه کند. حاصل این ضرب، یک ماتریس  $m \times m$  خواهد بود.

برای انجام این کار، ابتدا ماتریس  $m \times n$  به  $\lceil \frac{n}{m} \rceil$  ماتریس  $m \times m$  تقسیم‌بندی می‌شود. مشابهاً ماتریس  $n \times m$  نیز به  $\lceil \frac{n}{m} \rceil$  ماتریس  $m \times m$  تقسیم‌بندی می‌شود. حال با استفاده از ماژول ضرب‌کننده ماتریس ترتیبی و با توجه به این که قاعده ضرب بلوکی در ماتریس‌ها برقرار است، هر یک از این ماتریس‌های  $m \times m$  به مانند یک عدد در نظر گرفته می‌شود و ماتریس‌های متناظر به ترتیب در هم ضرب می‌شوند.

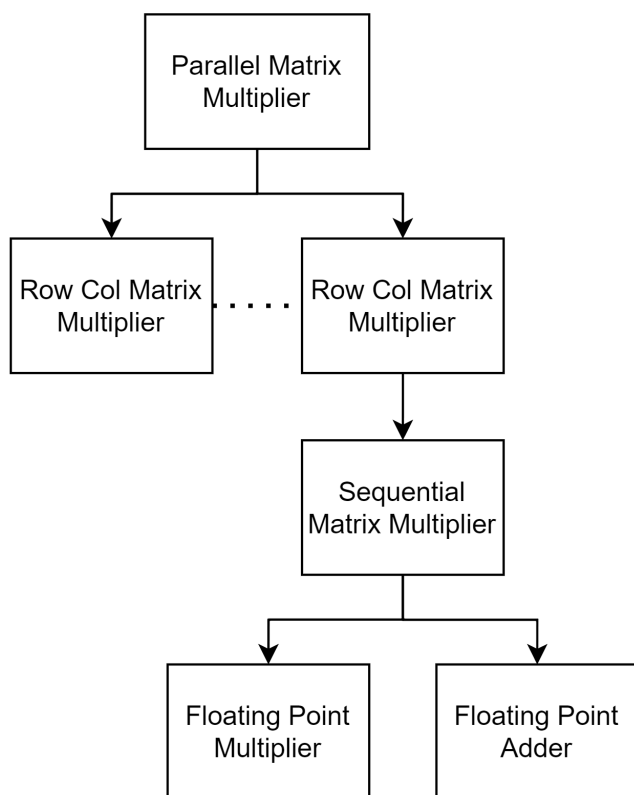
شکل ۲ بلوک دیاگرام ضرب‌کننده ماتریس سطری در ستونی است. اندیس‌های  $a_i$  و  $b_j$  مستقیماً از ضرب‌کننده ماتریس ترتیبی حاصل می‌شوند، اما دیگر اندیس‌ها بصورت

$$a_j = mk + a_{jseq}$$

$$b_i = mk + b_{iseq}$$

محاسبه می‌شوند، که در آن،  $k$  مشخص می‌کند که چندمین ماتریس  $m \times m$  در هر لحظه در حال محاسبه

<sup>۴</sup>Finite State Machine



شکل ۵: ساختار درختی سیستم.

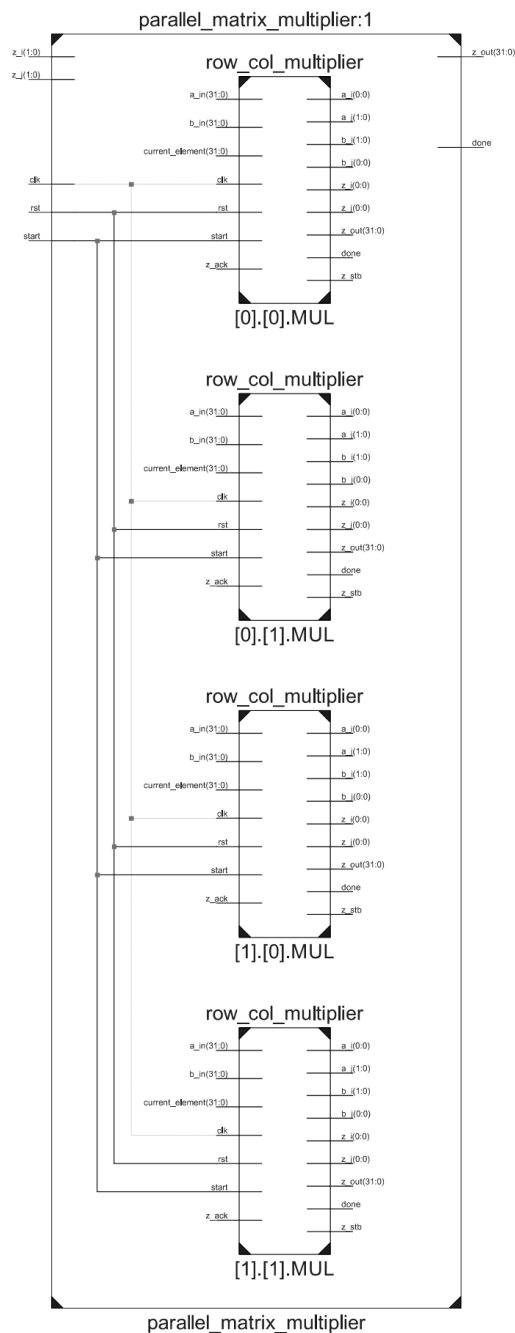
### ۳. شبیه‌سازی و نتایج

برای هر یک از ماژول‌های شرح داده شده در بخش قبل، یک Testbench نوشته شده است. مقادیر ورودی، در قالب فایل‌های باینری در پوشه data موجود اند و در ابتدای اجرای شبیه‌سازی، از فایل‌های مربوطه خوانده شده و در حافظه نوشته می‌شوند. همچنین پس از اجرای عملیات ضرب ماتریس، خروجی حاصل در فایل sim\_out.bin نوشته می‌شود.

برای بررسی و مقایسه پاسخ‌های حاصل از شبیه‌سازی، یک مدل طلایی توسط زبان پایتون نوشته شده است که ورودی‌ها و خروجی حاصل از شبیه‌سازی را دریافت کرده، و پاسخ صحیح را با پاسخ به دست آمده مقایسه می‌کند. اجرای مدل طلایی، توسط دستور زیر قابل انجام است.

```
python gold_standard/model.py data/<input_a
address> data/<input_b address> <sim_out.bin
address> <n> <k> <m>
```

در ادامه به هر کدام از Testbench‌ها به تفصیل



شکل ۴: بلوک دیاگرام ضرب‌کننده ماتریس موازی.

این ماژول موجود است، اما به علت پیچیدگی اتصالات مربوط به این حافظه‌ها، از نمایش آن‌ها در بلوک دیاگرام صرف نظر شده است.

در نهایت، ساختار درختی کل سیستم در شکل ۵ نمایش داده شده است.

پردازنده می‌شود.

ضرب‌کننده ماتریس ترتیبی فعال می‌شود. پس از فعال شدن سیگنال done در ماژول ضرب‌کننده، نتیجه حاصل در فایل خروجی نوشته می‌شود. نتیجه اجرای مدل طلایی نیز بصورت زیر است.

۳.۱. ضرب‌کننده ماتریس ترتیبی

برای آزمایش و شبیه‌سازی این ماژول، دو ماتریس

$4 \times 4$  به صورت زیر در هم ضرب شدند.

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \cdot \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 6 & 8 \\ 10 & 12 & 14 & 16 \\ 18 & 20 & 22 & 24 \\ 26 & 28 & 30 & 32 \end{bmatrix} \quad (1)$$

در این Testbench، ابتدا ورودی‌ها از فایل‌های مربوطه خوانده شده، سپس سیگنال start مربوط به ضرب‌کننده ماتریس ترتیبی فعال می‌شود. پس از فعال شدن سیگنال done در ماژول ضرب‌کننده، نتیجه حاصل در فایل خروجی نوشته می‌شود. نتیجه اجرای مدل طلایی نیز بصورت زیر است.

```
1 > python .\gold_standard\model.py .\data\row_input
    .bin .\data\col_input.bin .\sim_out.bin 4 8 4
2 A:
3 [[1. 2. 3. 4. 5. 6. 7. 8.]]
4 [1. 2. 3. 4. 5. 6. 7. 8.]
5 [1. 2. 3. 4. 5. 6. 7. 8.]
6 [1. 2. 3. 4. 5. 6. 7. 8.]]
7 B:
8 [[1.5 0. 0. 0. ]
9 [0. 1.5 0. 0. ]
10 [0. 0. 1.5 0. ]
11 [0. 0. 0. 1.5]
12 [1.5 0. 0. 0. ]
13 [0. 1.5 0. 0. ]
14 [0. 0. 1.5 0. ]
15 [0. 0. 0. 1.5]]
16 Actual:
17 [[ 9. 12. 15. 18.]
18 [ 9. 12. 15. 18.]
19 [ 9. 12. 15. 18.]
20 [ 9. 12. 15. 18.]]
21 Expected:
22 [[ 9. 12. 15. 18.]
23 [ 9. 12. 15. 18.]
24 [ 9. 12. 15. 18.]
25 [ 9. 12. 15. 18.]]
26 True
```

۳.۲. ضرب‌کننده ماتریس موازی

برای آزمایش و شبیه‌سازی این ماژول، همان دو ماتریس رابطه ۱ در هم ضرب شده‌اند. اما تفاوت در این‌جا این است که در این حالت هر یک از ماتریس‌ها، به چهار ماتریس  $2 \times 2$  تقسیم شده‌اند. ( $n = 2$ )

در این Testbench، ابتدا ورودی‌ها از فایل‌های مربوطه خوانده شده، سپس سیگنال start مربوط به ضرب‌کننده ماتریس ترتیبی فعال می‌شود. پس از فعال شدن سیگنال done در ماژول ضرب‌کننده، نتیجه حاصل در فایل خروجی نوشته می‌شود. نتیجه اجرای مدل طلایی نیز بصورت زیر است.

```
1 > python .\gold_standard\model.py .\data\
    square_input_a.bin .\data\square_input_b.bin
    .\sim_out.bin 4 4 4
2 A:
3 [[ 1.  2.  3.  4.]
4 [ 5.  6.  7.  8.]
5 [ 9. 10. 11. 12.]
6 [13. 14. 15. 16.]]
7 B:
8 [[2. 0. 0. 0.]
9 [0. 2. 0. 0.]
10 [0. 0. 2. 0.]
11 [0. 0. 0. 2.]]
12 Actual:
13 [[ 2.  4.  6.  8.]
14 [10. 12. 14. 16.]
15 [18. 20. 22. 24.]
16 [26. 28. 30. 32.]]
17 Expected:
18 [[ 2.  4.  6.  8.]
19 [10. 12. 14. 16.]
20 [18. 20. 22. 24.]
21 [26. 28. 30. 32.]]
22 True
```

۳.۲. ضرب‌کننده ماتریس سطری در ستونی

برای آزمایش و شبیه‌سازی این ماژول، دو ماتریس

$4 \times 4$  و  $8 \times 4$  به صورت زیر در هم ضرب شدند.

$A \times B$

$$= \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix} \cdot \begin{bmatrix} 1/5 & 0 & 0 & 0 \\ 0 & 1/5 & 0 & 0 \\ 0 & 0 & 1/5 & 0 \\ 0 & 0 & 0 & 1/5 \\ 1/5 & 0 & 0 & 0 \\ 0 & 1/5 & 0 & 0 \\ 0 & 0 & 1/5 & 0 \\ 0 & 0 & 0 & 1/5 \end{bmatrix}$$

$$= \begin{bmatrix} 9 & 12 & 15 & 18 \\ 9 & 12 & 15 & 18 \\ 9 & 12 & 15 & 18 \\ 9 & 12 & 15 & 18 \end{bmatrix}$$

در این Testbench، ابتدا ورودی‌ها از فایل‌های مربوطه خوانده شده، سپس سیگنال start مربوط به

```
1 > python .\gold_standard\model.py .\data\
    square_input_a.bin .\data\square_input_b.bin
    .\sim_out.bin 4 4 4
2 A:
3 [[ 1.  2.  3.  4.]
4 [ 5.  6.  7.  8.]
5 [ 9. 10. 11. 12.]
6 [13. 14. 15. 16.]]
7 B:
8 [[2. 0. 0. 0.]
9 [0. 2. 0. 0.]
10 [0. 0. 2. 0.]
11 [0. 0. 0. 2.]]
12 Actual:
13 [[ 2.  4.  6.  8.]
14 [10. 12. 14. 16.]
15 [18. 20. 22. 24.]
16 [26. 28. 30. 32.]]
17 Expected:
18 [[ 2.  4.  6.  8.]
```

Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan6
<b>Device</b>	XC6SLX150
Package	FGG484
Speed	-3
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	Verilog
Property Specification in Project File	Store all values
Manual Compile Order	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	<input type="checkbox"/>

شکل ۶: تنظیمات FPGA.

```

19 [10. 12. 14. 16.]
20 [18. 20. 22. 24.]
21 [26. 28. 30. 32.]]
22 True

```

#### ۴. سنتز بر روی FPGA و نتایج

با استفاده از نرم‌افزار XilinxISE، پروژه‌ای برای یک FPGA با تنظیماتی که در شکل ۶ نمایش داده شده است، ساخته شد. سپس فایل‌های FP\_adder.v، FP\_multiplier.v و sequential\_matrix\_multiplier.v، row\_col\_multiplier.v و parallel\_matrix\_multiplier.v به sources پروژه اضافه شد و مازول parallel\_matrix\_multiplier به عنوان System Top در نظر گرفته شد.

پس از تعیین منابع پروژه، System Top سنتز شد. نتایج بدست آمده از سنتز و همچنین طرح‌های شماتیک مدار نهایی در ادامه آمده‌اند.

#### ۵. نتیجه‌گیری

parallel_matrix_multiplier Project Status			
<b>Project File:</b>	Matrix_multiplier.xise	<b>Parser Errors:</b>	No Errors
<b>Module Name:</b>	parallel_matrix_multiplier	<b>Implementation State:</b>	Synthesized
<b>Target Device:</b>	xc6slx150-3fgg484	• <b>Errors:</b>	No Errors
<b>Product Version:</b>	ISE 14.7	• <b>Warnings:</b>	122 Warnings (0 new)
<b>Design Goal:</b>	Balanced	• <b>Routing Results:</b>	
<b>Design Strategy:</b>	Xilinx Default (unlocked)	• <b>Timing Constraints:</b>	
<b>Environment:</b>	System Settings	• <b>Final Timing Score:</b>	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	Used	Available	Utilization
Number of Slice Registers	2932	184304	1%
Number of Slice LUTs	4508	92152	4%
Number of fully used LUT-FF pairs	2261	5179	43%
Number of bonded IOBs	40	338	11%
Number of BUFG/BUFGCTRLs	1	16	6%
Number of DSP48A1s	16	180	8%

Detailed Reports					[-]
Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report	Current	Sat Feb 6 17:29:47 2021	0	122 Warnings (0 new)	423 Infos (0 new)
Translation Report					
Map Report					
Place and Route Report					
Power Report					
Post-PAR Static Timing Report					
Bitgen Report					

Secondary Reports		[-]
Report Name	Status	Generated
Post-Synthesis Simulation Model Report	Current	Sat Feb 6 17:45:44 2021

**Date Generated:** 02/06/2021 - 17:50:30