



IP CORE KULLANIMI

1. Giriş

IP Core (Intellectual Property Core), onu geliştiren kişi veya şirkete ait, başkaları tarafından tekrar kullanılabilen donanım modülleridir. IP Core'lar genellikle kaynak kodu kapalı olacak şekilde yayınlanır. IP Core'u kullanacak olan kişi, sadece bu IP Core'un giriş/çıkış arayüzü (I/O Interface) bilir. IP Core'un içi ise kullanıcı için bir kara kutudur (black box). Yani, kullanıcı IP Core'un ne şekilde gerçekleştirildiğini bilmez.

Kullanım şekli açısından IP Core kullanmak, kendi gerçekleştirdiğiniz modülleri başka modüller içerisinde kullanmaya (instance oluşturmak) benzer. En önemli farkı, Instance'ini oluşturacağınız IP Core'un kodunu görememektir. IP Core'u sadece giriş/çıkış sinyallerinin ne işe yaradığını bilerek kullanabilirsiniz.

IP Core'lar oluşturma sırasında izin verilen ölçüde yapılandırılabilirler. Örneğin; bölme işlemi yapan bir IP Core'da bölünen ve bölen sayıların kaç bitlik ve hangi formatta olacağını belirlemek mümkün olabilir.

Bu derste, Vivado ile birlikte gelen iki farklı IP Core kullanarak uygulamalar gerçekleştireceksiniz.

2. TWO'S COMPLEMENT MULTIPLY-ADD MODULE

Bu bölümde, 32 bitlik ikiye tümleyen (two's complement) sayılarla çarpıp-toplama (multiply-add, $D = A*B + C$) aritmetik işleminin sonucunu hesaplayan bir modül gerçekleştirmeniz istenmektedir. Bu modülü, IP Core kullanarak gerçekleştirmelisiniz. Vivado penceresinin solunda "IP Catalog" butonuna tıklayarak bu IP Core'a aşağıdaki yolu izleyerek ulaşabilirsiniz:

- Baseline > Multiply Adder

IP Catalog'dan seçtikten sonra açılan yapılandırma penceresinde bu IP Core'u 32 bitlik sayılarla çarpıp-toplama işlemi yapacak şekilde ayarlamamız gerekiyor. Yani, bu IP Core, 32 bitlik 3 adet sayı girişi ve 32 bitlik çıkışa sahip olmalıdır. Yapmanız gereken ayarlamalar aşağıdaki şekilde görüldüğü gibi olmalıdır. Multiply Add IP Core'u her çevrimde çarpıp-toplama işleminin bir kısmını yapar. Dolayısıyla, sonuç birkaç ardışık çevrim sonunda üretilmiş olur. Şekilde görülen "Actual AB Latency" ve "Actual C Latency" sonucun girişler verildikten kaç çevrim (saatin yükselen kenarı) sonra hesaplanmış olacağını göstermektedir. Şekilde verilenlere göre, çarpılacak olan sayılar verildikten 6 çevrim sonra, toplanacak olan sayı verildikten 5 çevrim sonra sonuç hesaplanmış olacaktır. Dolayısıyla, toplanacak sayıyı C girişine, çarpılacak olan sayıları verdikten 1 çevrim sonra göndermeliyiz. Örneğin; $50*5 + 6$ işlemi yapmak istediğimizde, ilk çevrimde A ve B girişlerinden 50 ve 5 sayılarını göndermeliyiz. Sonraki çevrimde ise C girişinden 6 sayısını göndermeliyiz. A ve B'nin verildiği çevrimde C girişinde ne olduğu önemli değildir. IP Core'u kullanarak sizin yapacağınız modül, aynı çevrimde çarpılacak ve toplanacak sayıları alarak, toplanacak sayıyı IP Core'a bir çevrim geç göndermeyi kendi yapmalıdır.

Ayarlamalar sonucunda ortaya çıkan IP Core modülünün bir örneğini (instance) aşağıdaki gibi oluşturabilirsiniz. Buradaki giriş/çıkış sinyallerinin açıklamaları şu şekildedir:



BİL264L - Mantıksal Devre Tasarımı Laboratuvarı ELE263L - Sayısal Sistem Tasarımı Laboratuvarı

- **CLK**: saat sinyalıdır
- **CE** (Clock Enable): Bu giriş mantık-1 olduğunda modül her çevrim bir iş yapar ve belli sayıda çevrim sonra sonuç üretilir. Mantık-0 olduğunda ise modül bekleme (pause) durumuna geçer. Bunu geçen hafta yapılan sayaç uygulamasındaki “duraklat” girişine benzetebilirsiniz.
- **SCLR** (Synchronous Clear): CLK girişi ile senkron olan bu giriş, modülün içerisinde devam eden hesaplamaları sonlandırır. Bunu önceki hafta kullandığımız “rst” girişine benzetebilirsiniz.
- **A, B, C**: Sırasıyla çarpılacak ve toplanacak olan 32 bitlik sayılardır ($A*B + C$)
- **SUBTRACT**: Bu giriş mantık-1 olduğunda $A*B - C$, mantık-0 olduğunda ise $A*B + C$ işlemi yapılır.
- **P**: Çarpıp-toplama işleminin sonucunun verildiği çıkıştır.
- **PCOUT**: Bu çıkış modülün içerisinde yapılan işlemlerle ilgili ara çıktıları verir. Bu çıkışı dikkate almanız ve kullanmanız gerekmemektedir.

```
ip_multadd i_mult_add (  
    .CLK(),          // input wire CLK  
    .CE(),           // input wire CE  
    .SCLR(),         // input wire SCLR  
    .A(),            // input wire [31 : 0] A  
    .B(),            // input wire [31 : 0] B  
    .C(),            // input wire [31 : 0] C  
    .SUBTRACT(),     // input wire SUBTRACT  
    .P(),            // output wire [31 : 0] P  
    .PCOUT()         // output wire [47 : 0] PCOUT  
);
```

Component Name

P =	A	*	B	+	C
Input Type	<input type="text" value="Signed"/>		<input type="text" value="Signed"/>		<input type="text" value="Signed"/>
Input Width	<input type="text" value="32"/>		<input type="text" value="32"/>		<input type="text" value="32"/>
	[2,53]		[2,53]		[2,106]

☐ Use PCIN

Output MSB [0 - 106]
Output LSB [0 - 106]

Control and Latencies

Latency can be set to -1 or 0. The -1 selection will provide the optimum latency for max frequency for the given parameters. If either one of the latencies is set to -1, they both will be treated as having -1 set.

A:B - P Latency Actual AB Latency: 6
C - P Latency Actual C Latency: 5

Synchronous Controls and Clock Enable(CE) Priority



BİL264L - Mantıksal Devre Tasarımı Laboratuvarı ELE263L - Sayısal Sistem Tasarımı Laboratuvarı

- **[Gerçekleştirme]** 32 bitlik ikiye tümleyen sayılar ile çarpıp-toplama ($A*B + C$) işlemini gerçekleştirecek olan modüle “CarpTopla” ismini verin. Bu modülün her biri 32 bitlik “csayi1”, “csayi2” ve “tsayi” isminde girişleri olmalıdır. Bu girişlerin geçerli olduğunu ve o çevrimdeki sayılarla sonuç hesaplanması istendiğini gösteren “valid_in” isminde tek bitlik giriş olmalıdır. “valid_in” girişi mantık-1 olduğunda verilen sayılarla çarpıp-toplama işlemine başlanacak ve 6 çevrim sonra hesaplanmış olan sonuç “sonuç” isimli 32 bitlik çıkıştan verilecektir. Yani, “CarpTopla” modülü “sonuc = csayi1*csayi2 + tsayi” işlemini gerçekleştirecektir. Tek bitlik “valid_out” çıkışı ise mantık-1 olduğunda sonucun geçerli olduğunu gösterecektir. Yani “valid_in” mantık-1 olduktan 6 çevrim sonra, “valid_out” da bir çevrim boyunca mantık-1 olacaktır. Bu işlem için, yukarıda sözü edilen IP Core kullanılacaktır. Bu IP Core’u yapılandırırken, ikiye tümleyen sayılarla işlem yapan IP Core’a “ip_multadd” ismini verin.
- **[Simülasyon]** Gerçekleştirmiş olduğunuz “CarpTopla” modülünün istenildiği gibi çalıştığından emin olmak için, bu modülü test eden testbench kodunu yazın. Ardışık çevrimlerde modülünüzde geçerli işlemler başlatarak, 6 çevrim gecikmeli olarak doğru sonuçların hesaplandığını gözlemleyin. Vivado yazılımını kullanarak modülün simülasyonunu yapıp, karedalga (Waveform) görünümünden modülün doğru çalıştığını kontrol edin.

3. FLOATING-POINT MULTIPLY-ADDER

Bu bölümde, 32 bitlik kayan-nokta sayılarla çarpıp-toplama işleminin sonucunu hesaplayan modül gerçekleştirmeniz istenmektedir. Bu modülü IP Core kullanarak gerçekleştirmelisiniz. Vivado penceresinin solunda “IP Catalog” butonuna tıklayarak bu IP Core’a aşağıdaki yolu izleyerek ulaşabilirsiniz:

- Math Functions > Floating Point > Floating-point

IP Catalog’dan seçtikten sonra açılan yapılandırma penceresinde bu IP Core’u 32 bitlik sayılarla çarpıp-toplama işlemi yapacak şekilde ayarlamanız gerekiyor. Yapmanız gereken ayarlamalar aşağıdaki şekilde görüldüğü gibi olmalıdır.



BİL264L - Mantıksal Devre Tasarımı Laboratuvarı

ELE263L - Sayısal Sistem Tasarımı Laboratuvarı

Component Name:

Operation Selection | **Precision of Inputs** | Optimizations | Interface Options

A Precision Type

Please select floating-point precision

☐ Half ☒ Single ☐ Double ☐ Custom

Exponent Width: [0 - 64]

Fraction Width: [0 - 64]

Total Width : 32

Component Name:

Operation Selection | Precision of Inputs | Optimizations | Interface Options

Please select from the following functions:

Operation Selection: Add/Subtract and FMA Operator options

☐ Absolute Value ☐ Both

☐ Accumulator ☒ Add

☐ Add/Subtract ☐ Subtract

☐ Compare

☐ Divide

☐ Exponential

☐ Fixed-to-float

☐ Float-to-fixed

☐ Float-to-float

☒ Fused Multiply-Add

☐ Logarithm

☐ Multiply

☐ Reciprocal

☐ Reciprocal Square Root

☐ Square-root

Fused Multiply-Add operation selected. **RESULT = (A*B)+C**



BİL264L - Mantıksal Devre Tasarımı Laboratuvarı ELE263L - Sayısal Sistem Tasarımı Laboratuvarı

Component Name

Operation Selection Precision of Inputs Optimizations **Interface Options**

Flow Control Options

Flow Control Optimize Goal

☒ RESULT channel has TREADY

Latency and Rate Configuration

☒ Use Maximum Latency

Latency [1 - 17] Select in order to specify that maximum latency should be used.

Cycles/operation [1 - 27]

Control Signals

☐ ACLKEN ☐ ARESETn (active low)

ARESETn must be asserted for a minimum of two clock cycles

Optional Output Fields

☐ UNDERFLOW ☐ OVERFLOW ☐ INVALID OP

☐ DIVIDE BY ZERO ☐ ACCUM OVERFLOW ☐ ACCUM INPUT OVERFLOW

Channel	Has TLAST	Has TUSER	
A	<input type="checkbox"/>	<input type="checkbox"/>	1
B	<input type="checkbox"/>	<input type="checkbox"/>	1
C	<input type="checkbox"/>	<input type="checkbox"/>	1
OPERATION	<input type="checkbox"/>	<input type="checkbox"/>	1

TLAST Behavior

TLAST Behavior

Ayarlamalar sonucunda ortaya çıkan IP Core modülünün bir örneğini (instance) aşağıdaki gibi oluşturabilirsiniz.

```
ip_fp_multadd i_fp_multadd (  
    .aclk(),                // input wire aclk  
    .s_axis_a_tvalid(),     // input wire s_axis_a_tvalid  
    .s_axis_a_tready(),     // output wire s_axis_a_tready  
    .s_axis_a_tdata(),      // input wire [31 : 0] s_axis_a_tdata  
    .s_axis_b_tvalid(),     // input wire s_axis_b_tvalid  
    .s_axis_b_tready(),     // output wire s_axis_b_tready  
    .s_axis_b_tdata(),      // input wire [31 : 0] s_axis_b_tdata  
    .s_axis_c_tvalid(),     // input wire s_axis_c_tvalid  
    .s_axis_c_tready(),     // output wire s_axis_c_tready  
    .s_axis_c_tdata(),      // input wire [31 : 0] s_axis_c_tdata  
    .m_axis_result_tvalid(), // output wire m_axis_result_tvalid  
    .m_axis_result_tready(), // input wire m_axis_result_tready  
    .m_axis_result_tdata()  // output wire [31 : 0] m_axis_result_tdata  
);
```



BİL264L - Mantıksal Devre Tasarımı Laboratuvarı ELE263L - Sayısal Sistem Tasarımı Laboratuvarı

Buradaki giriş/çıkış sinyallerinin açıklamaları şu şekildedir:

- **aclk**: saat sinyalıdır.
 - **s_axis_a_tvalid/s_axis_b_tvalid/s_axis_c_tvalid**: Carpilacak ve toplanacak sayılar için geçerli (valid) girişleridir. Modülünüzde tüm girişleri aynı anda vereceğinizden tüm bu girişleri aynı sinyale bağlayabilirsiniz.
 - **s_axis_a_tready/s_axis_b_tready/s_axis_c_tready**: IP Core modülünün girişlerinin yeni sayılar almaya hazır olduğunu gösteren çıkışlardır. Bizim kullandığımız yapılandırmada IP Core her çevrimde yeni sayı almaya hazır olacağından bu çıkışları dikkate almanız gerekmiyor.
 - **s_axis_a_tdata/s_axis_b_tdata/s_axis_c_tdata**: 32 bitlik veri girişleridir.
 - **m_axis_result_tvalid**: Sonucun geçerli (valid) olduğunu gösteren çıkıştır.
 - **m_axis_result_tready**: Bu giriş ile sonucun okunmaya hazır olduğu belirtiliyor. Yani, ikiye tümleyen sayılar için kullandığımız IP Core'un CE girişi ile aynı görevi görüyor. Bu girişe mantık-1 bağlayın.
 - **m_axis_result_tdata**: 32 bitlik sonucun verildiği çıkıştır.
- **[Gerçekleştirme]** 32 bitlik kayan-nokta sayılar ile çarpıp-toplama ($A*B + C$) işlemini gerçekleştirecek olan modüle "FpCarpTopla" ismini verin. Bu modülün giriş ve çıkış sinyallerinin isimleri önceki bölümde gerçekleştirdiğiniz "CarpTopla" modülü ile tamamen aynı olmalıdır.
- **[Simülasyon]** Gerçekleştirmiş olduğunuz "FpCarpTopla" modülünün istenildiği gibi çalıştığından emin olmak için, bu modülü test eden testbench kodunu yazın. Ardışık çevrimlerde modülünüzde geçerli işlemler başlatarak, birkaç çevrim gecikmeli olarak doğru sonuçların hesaplandığını gözlemleyin. Vivado yazılımını kullanarak modülün simülasyonunu yapıp, karedalga (Waveform) görünümünden modülün doğru çalıştığını kontrol edin.

4. TWO'S COMPLEMENT & FLOATING-POINT MULTIPLY ADDER

Önceki iki bölümde gerçekleştirdiğiniz ikiye tümleyen ve kayan nokta sayılarla çarpıp-toplama işlemlerinin her ikisini de gerçekleştirebilen bir modül gerçekleştirin. Bu modülün giriş/çıkış sinyalleri önceki bölümlerdekiyle aynı olmalıdır. Sadece, girişlerden gelen sayıların hangi formatta (ikiye tümleyen veya kayan_nokta) olduğunu belirten bir bitlik "format" isminde giriş olacaktır.

Uyarı: İkiye tümleyen ve kayan-nokta sayılarla çarpıp-toplama işlemini yapan IP Core'lar farklı gecikmelerle sonuç üretmektedir. Bu nedenle, bu IP Core'lara girişleri farklı çevrimlerde vermiş olsanız bile, sonuçlar aynı çevrimde hesaplanabilir. Bu durumda ortaya çıkan sonuçlardan birini kaybetmemek için Multiply Add modülünün "CE" girişini kullanabilirsiniz.

- **[Gerçekleştirme]** Hem ikiye tümleyen hem de kayan-nokta sayılarla çarpıp-toplama işlemini yapan modülünüze "TcFpCarpTopla" ismini verin. Bu modülün, "format" adında girişlerdeki sayıların hangi türde (ikiye tümleyen veya kayan-nokta) olduğunu belirten bir bitlik giriş olacaktır. Modülün diğer girişleri 2. ve 3. bölümdeki gerçekleştirmenizin beklediği modüllerle aynı olmalıdır.



BİL264L - Mantıksal Devre Tasarımı Laboratuvarı ELE263L - Sayısal Sistem Tasarımı Laboratuvarı

- **[Simülasyon]** Gerçekleştirmiş olduğunuz “TcFpCarpTopla” modülünün istenildiği gibi çalıştığından emin olmak için, bu modülü test eden testbench kodunu yazın. Vivado yazılımını kullanarak modülün simülasyonunu yapıp, karedalga (Waveform) görünümünden modülün doğru çalıştığını kontrol edin.