You can pass a variable to your `.csproj` file during `dotnet build` and use it in a `<Condition>` by leveraging **MSBuild properties**.

Here's the **complete approach**:

1️⃣ Pass a property from the command line

```
dotnet build -p:MyCustomFlag=true
```

2️⃣ Use it in your `.csproj`

Inside your `.csproj` file, you can reference it like this:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <!-- Default value if not passed -->
    <MyCustomFlag>false</MyCustomFlag>
  </PropertyGroup>

  <!-- Conditional property -->
  <PropertyGroup Condition="'$(MyCustomFlag)' == 'true'">
    <DefineConstants>$(DefineConstants);MY_FLAG</DefineConstants>
  </PropertyGroup>

  <!-- Conditional item inclusion -->
  <ItemGroup Condition="'$(MyCustomFlag)' == 'true'">
    <Compile Include="ExtraFeature.cs" />
  </ItemGroup>

</Project>
```

3️⃣ Use it in C# code

You can now use the constant in your C# code:

```
using System;
```

```
class Program
{
    static void Main()
    {
#if MY_FLAG
        Console.WriteLine("Extra feature enabled!");
#else
        Console.WriteLine("Default build.");
#endif
    }
}
```

4️⃣ Example build

```
dotnet build -p:MyCustomFlag=true
```

Output:

```
Extra feature enabled!
```

```
dotnet build
```

Output:

```
Default build.
```

✅ **Key points:**

- Use `-p:PropertyName=Value` to pass variables to MSBuild.
- In `.csproj`, use `Condition="'$(PropertyName)' == 'value'"`.
- You can use these properties to control constants, references, file inclusion, etc.

If you want, I can also show you **how to pass multiple variables** and use them in **multi-target builds**.
Do you want me to prepare that example?