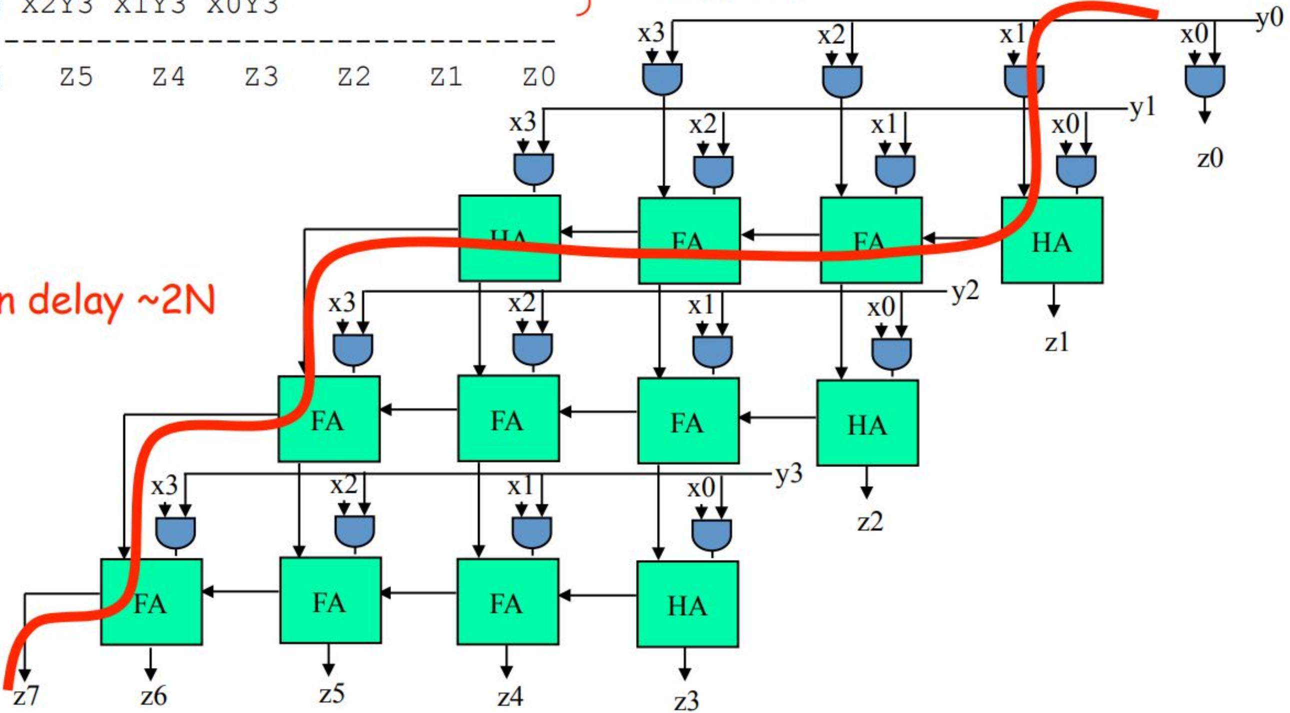
Multiplier and Pipeline

Combinational Multiplier (unsigned)

```
                        X3    X2    X1    X0   ← multiplicand
                      * Y3    Y2    Y1    Y0   ← multiplier
                      --------------------
                       X3Y0  X2Y0  X1Y0  X0Y0
+                X3Y1  X2Y1  X1Y1  X0Y1
+          X3Y2  X2Y2  X1Y2  X0Y2
+    X3Y3  X2Y3  X1Y3  X0Y3
-----------------------------------------
   Z7    Z6    Z5    Z4    Z3    Z2    Z1    Z0
```

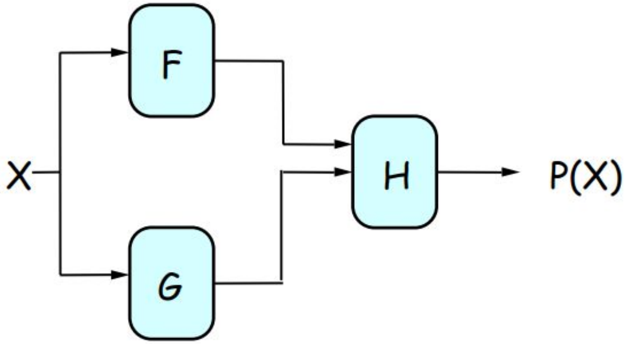
Partial products, one for each bit in multiplier (each bit needs just one AND gate)

- Propagation delay ~2N



x	y	z = x & y
0 *	0 =	0
0 *	1 =	0
1 *	0 =	0
1 *	1 =	1

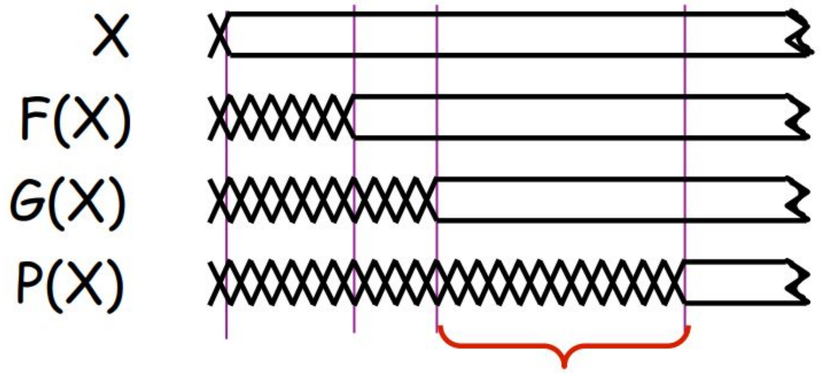
Performance of Combinational Circuits



For combinational logic:

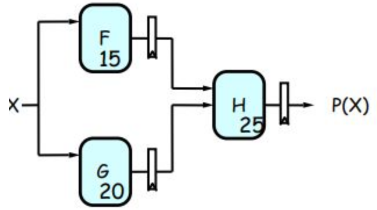
$L = t_{PD}$,
$T = 1/t_{PD}$.

We can't get the answer faster, but are we making effective use of our hardware at all times?



F & G are "idle", just holding their outputs stable while H performs its computation

Pipeline diagrams



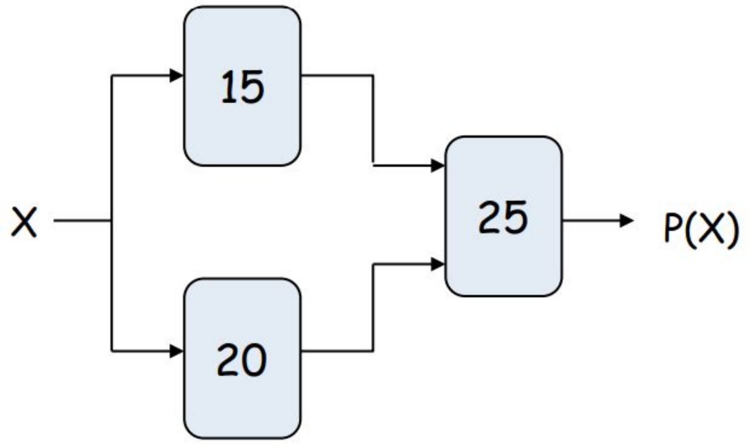
Clock cycle →

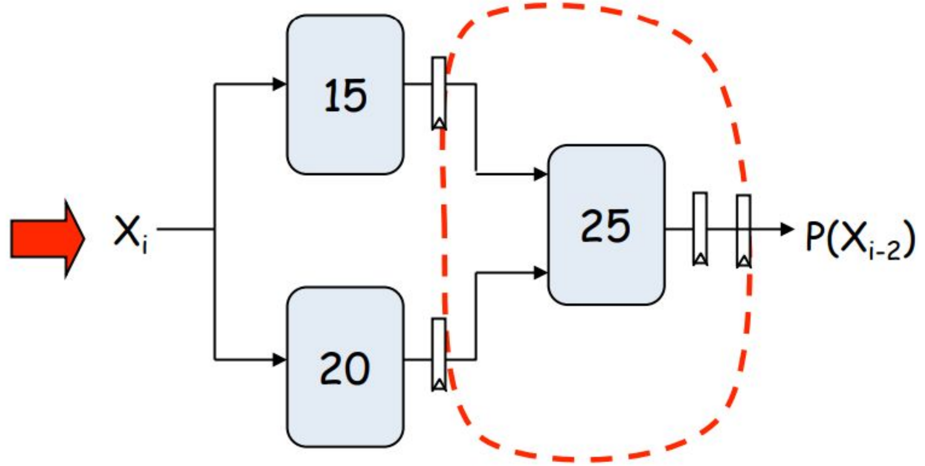
Pipeline stages ↓

	i	i+1	i+2	i+3	
Input	X_i	X_{i+1}	X_{i+2}	X_{i+3}	...
F Reg G Reg		$F(X_i)$ $G(X_i)$	$F(X_{i+1})$ $G(X_{i+1})$	$F(X_{i+2})$ $G(X_{i+2})$	...
H Reg			$H(X_i)$	$H(X_{i+1})$	$H(X_{i+2})$

The results associated with a particular set of input data moves *diagonally* through the diagram, progressing through one pipeline stage each clock cycle.

Retiming Combinational Circuits aka "Pipelining"





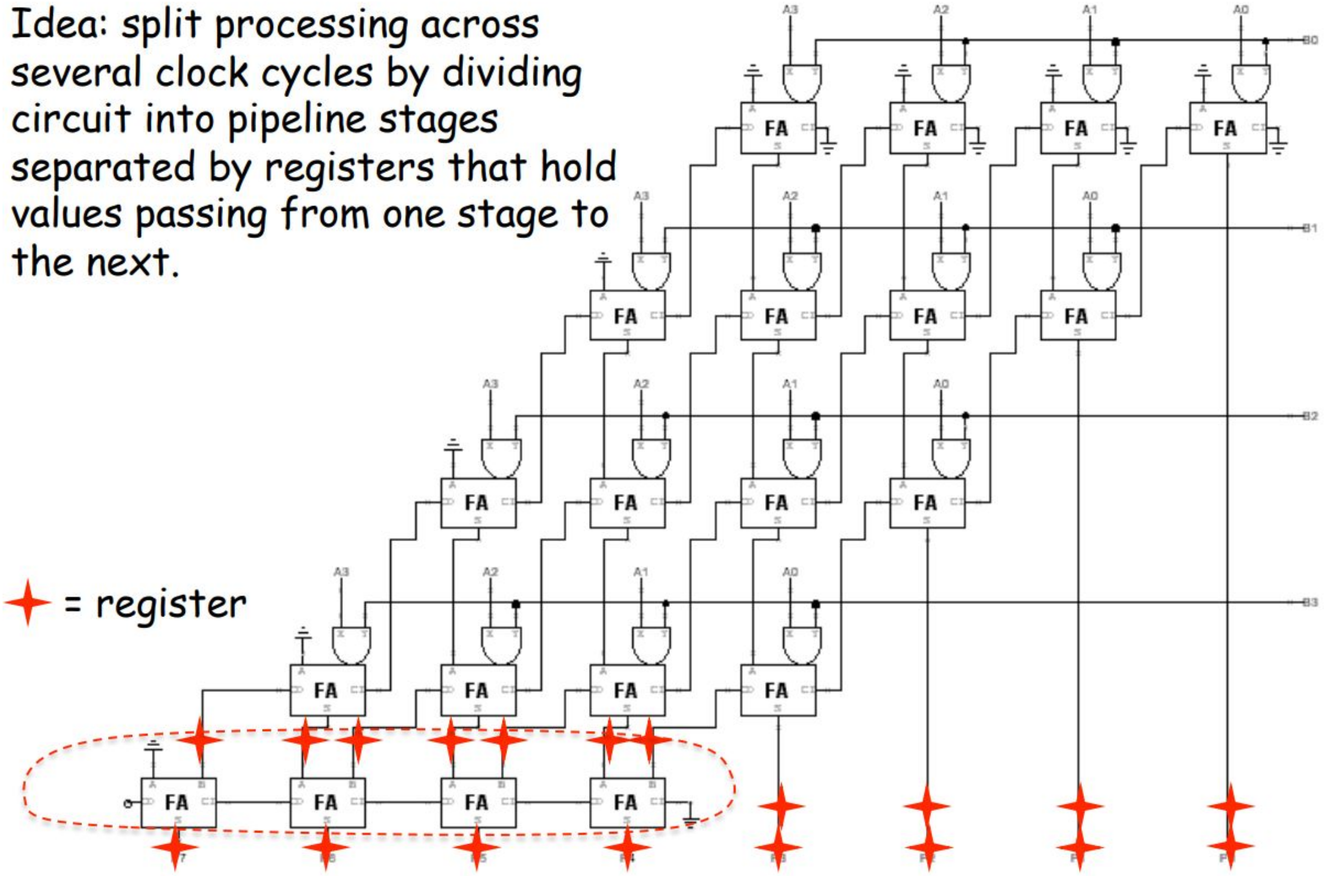
L = 45
T = 1/45

Assuming ideal registers:
i.e., $t_{PD} = 0$, $t_{SETUP} = 0$ → $t_{CLK} = 25$
$L = 2*t_{CLK} = 50$
$T = 1/t_{CLK} = 1/25$

Increasing Throughput: Pipelining

Idea: split processing across several clock cycles by dividing circuit into pipeline stages separated by registers that hold values passing from one stage to the next.

✦ = register



Throughput = $1/4t_{PD,FA}$ instead of $1/8t_{PD,FA}$)

Multiplication in Verilog

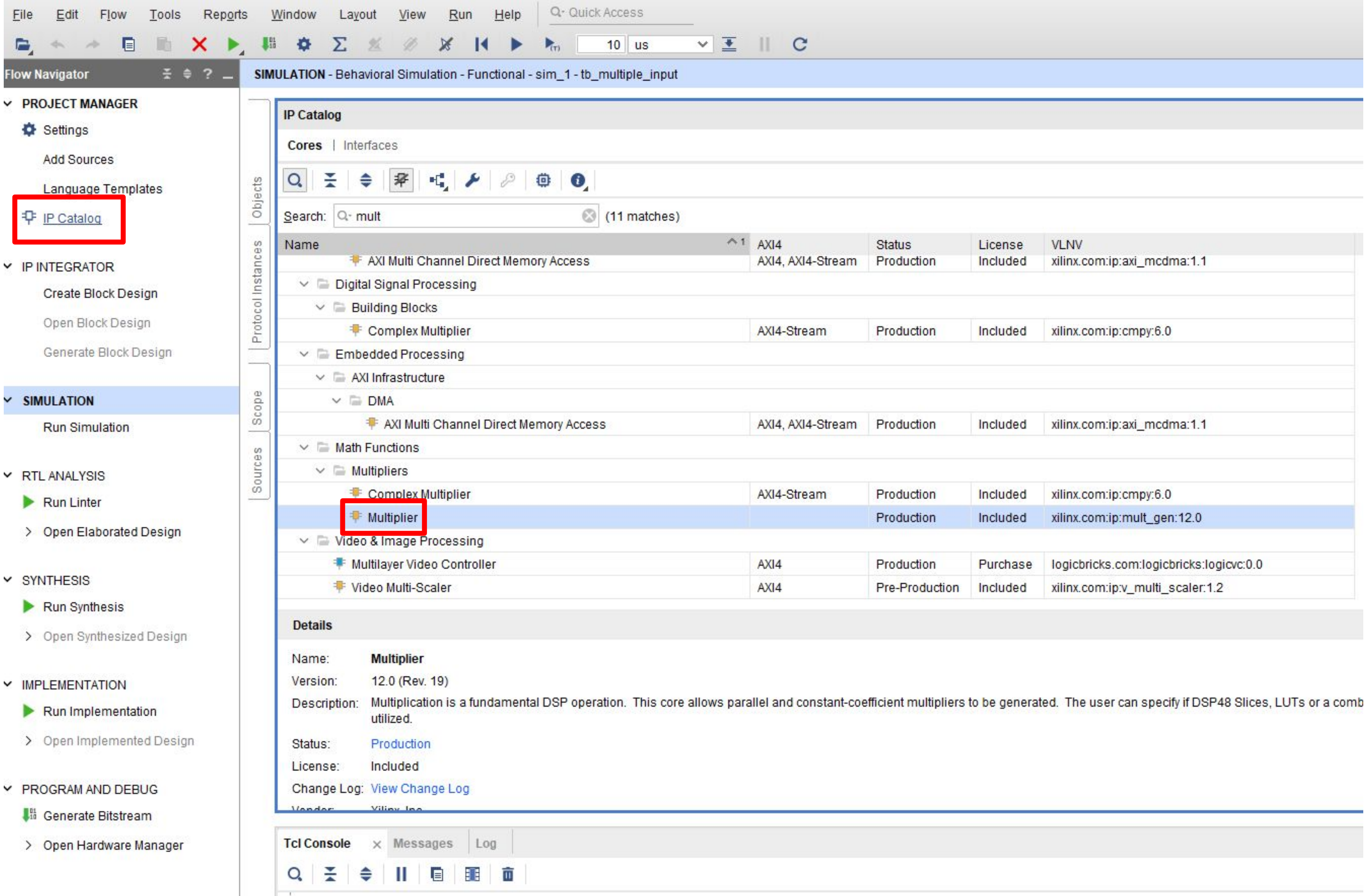
You can use the "*" operator to multiply two numbers:

```
wire [9:0] a,b;
wire [19:0] result = a*b;    // unsigned multiplication!
```

If you want Verilog to treat your operands as signed two's complement numbers, add the keyword signed to your wire or reg declaration:

```
wire signed [9:0] a,b;
wire signed [19:0] result = a*b;  // signed multiplication!
```

Remember: unlike addition and subtraction, you need different circuitry if your multiplication operands are signed vs. unsigned. Same is true of the >>> (arithmetic right shift) operator. To get signed operations all operands must be signed.



SIMULATION - Behavioral Simulation - Functional - sim_1 - tb_multiple_input

IP Catalog

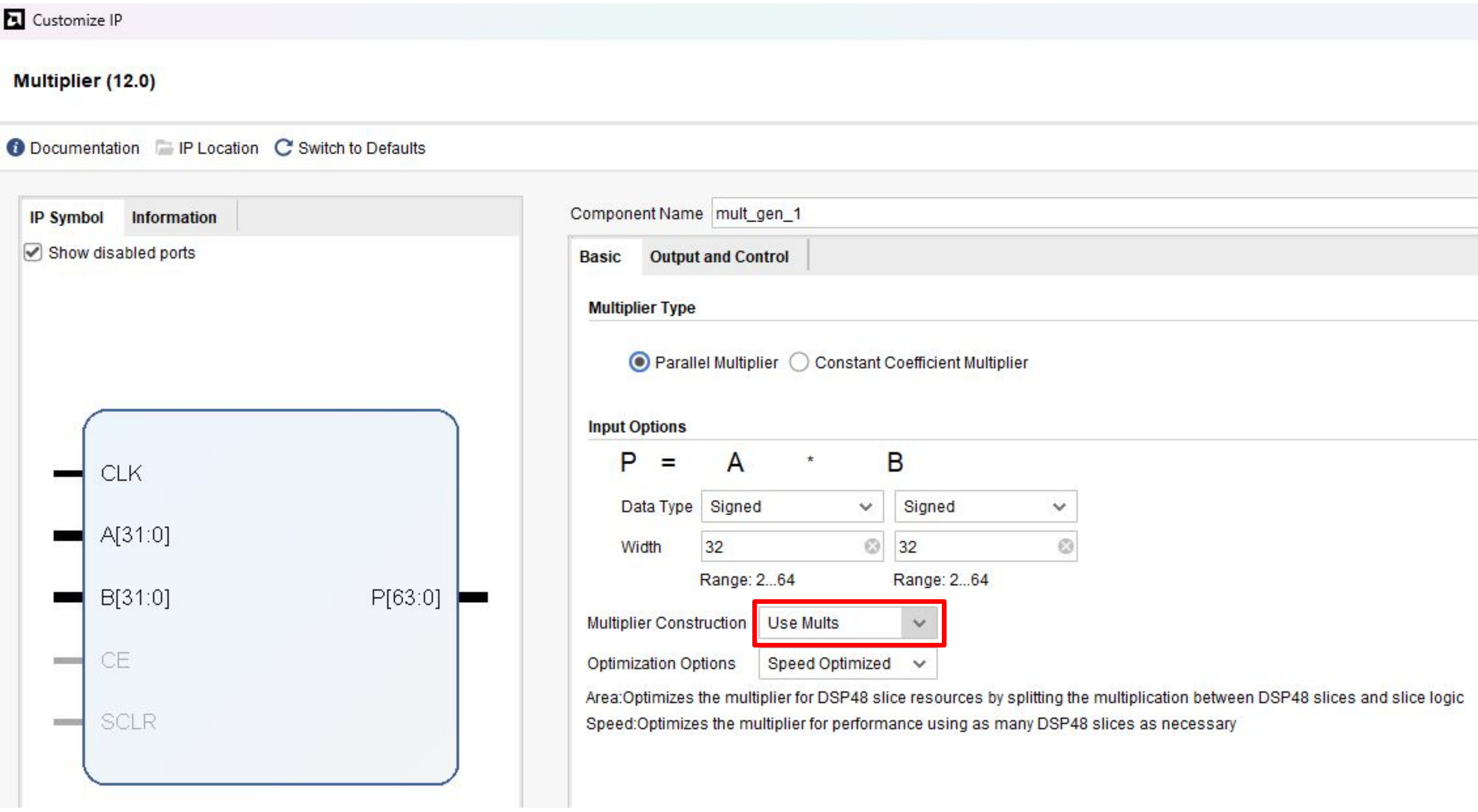
Cores | Interfaces

Search: mult (11 matches)

Name	AXI4	Status	License	VLNV
AXI Multi Channel Direct Memory Access	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:axi_mcdma:1.1
Digital Signal Processing				
Building Blocks				
Complex Multiplier	AXI4-Stream	Production	Included	xilinx.com:ip:cmpy:6.0
Embedded Processing				
AXI Infrastructure				
DMA				
AXI Multi Channel Direct Memory Access	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:axi_mcdma:1.1
Math Functions				
Multipliers				
Complex Multiplier	AXI4-Stream	Production	Included	xilinx.com:ip:cmpy:6.0
Multiplier		Production	Included	xilinx.com:ip:mult_gen:12.0
Video & Image Processing				
Multilayer Video Controller	AXI4	Production	Purchase	logicbricks.com:logicbricks:logicvc:0.0
Video Multi-Scaler	AXI4	Pre-Production	Included	xilinx.com:ip:v_multi_scaler:1.2

Details

Name:	**Multiplier**
Version:	12.0 (Rev. 19)
Description:	Multiplication is a fundamental DSP operation. This core allows parallel and constant-coefficient multipliers to be generated. The user can specify if DSP48 Slices, LUTs or a comb utilized.
Status:	Production
License:	Included
Change Log:	View Change Log
Vendor:	Xilinx, Inc


Customize IP

Multiplier (12.0)

Component Name: mult_gen_1

Multiplier Type

[✓ Parallel Multiplier] Constant Coefficient Multiplier

Input Options

P = A * B

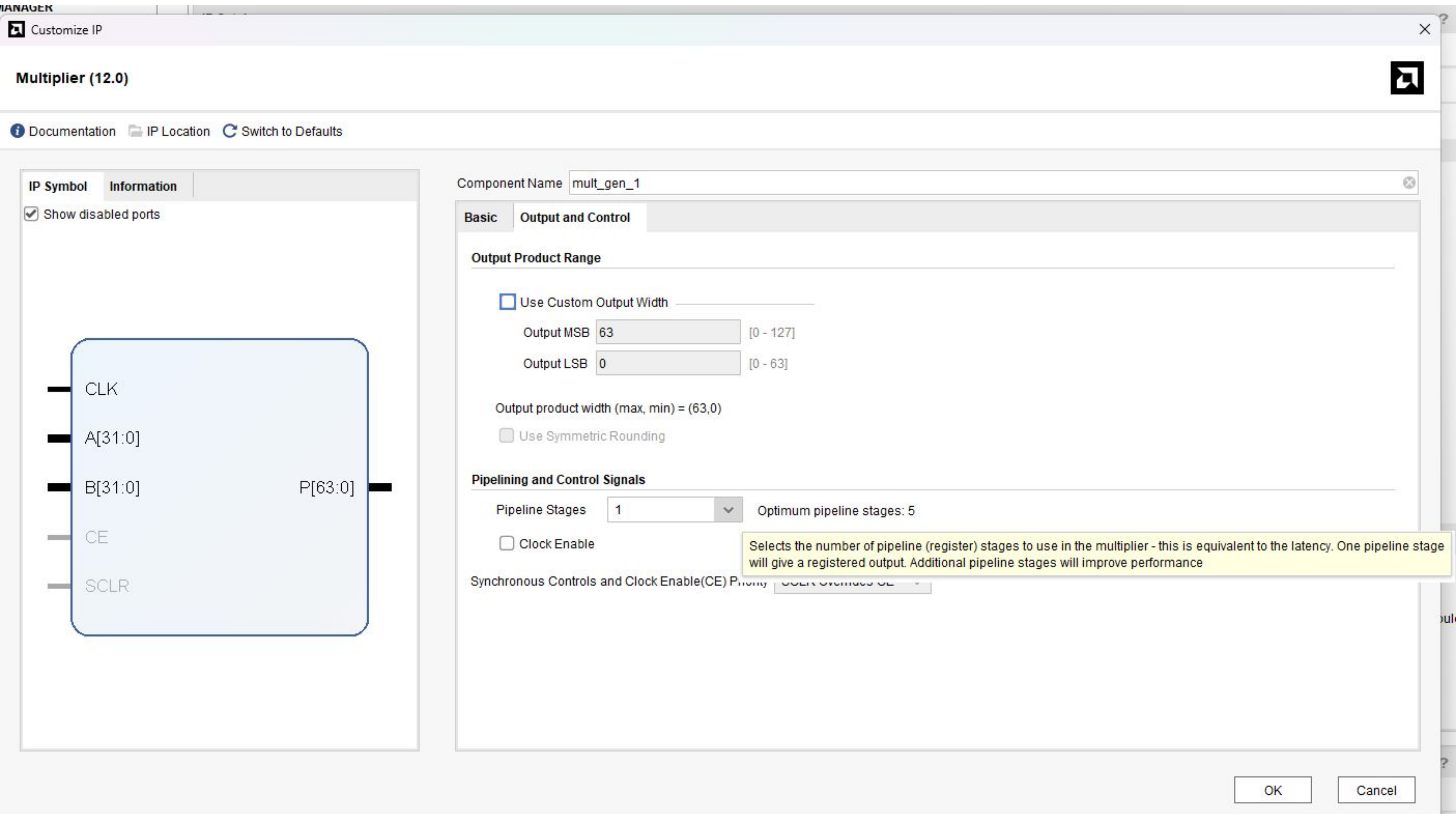
	A	B
Data Type	Signed	Signed
Width	32	32
	Range: 2...64	Range: 2...64

Multiplier Construction: Use Mults

Optimization Options: Speed Optimized

Area:Optimizes the multiplier for DSP48 slice resources by splitting the multiplication between DSP48 slices and slice logic

Speed:Optimizes the multiplier for performance using as many DSP48 slices as necessary



Customize IP

Multiplier (12.0)

Documentation | IP Location | Switch to Defaults

IP Symbol | Information

[✓] Show disabled ports

Component Name: mult_gen_1

Basic | **Output and Control**

Output Product Range

[ ] Use Custom Output Width

Output MSB: 63 [0 - 127]

Output LSB: 0 [0 - 63]

Output product width (max, min) = (63,0)

[ ] Use Symmetric Rounding

Pipelining and Control Signals

Pipeline Stages: 1 — Optimum pipeline stages: 5

[ ] Clock Enable

Selects the number of pipeline (register) stages to use in the multiplier - this is equivalent to the latency. One pipeline stage will give a registered output. Additional pipeline stages will improve performance

Synchronous Controls and Clock Enable(CE) Priority: SCLR Overrides CE

OK | Cancel