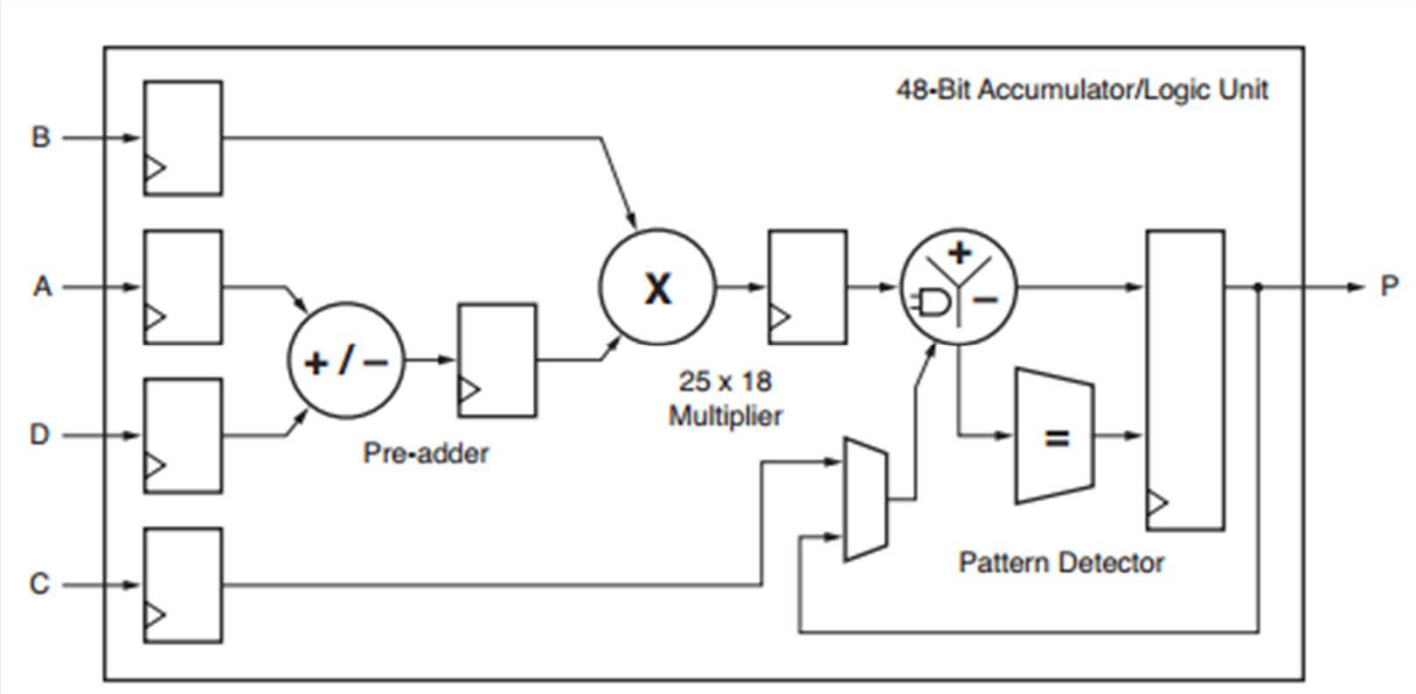
DSP Resources

- Specialized FPGA columns for complex arithmetic functionality
- DSP48 Tile: two DSP48 slices, interconnect
- Each DSP48 is a self-contained arithmetic-logical unit with add/sub/multiply/logic operations

DSP Resources

- DSP columns are, whenever possible, paired with Block RAM columns
- DSP48 tiles line up with RAMB36 (36Kb Block RAM)
- DSP48 slices line up with RAMB18
- Setup results in very fast routing from Block RAM to DSP slices

DSP Resources

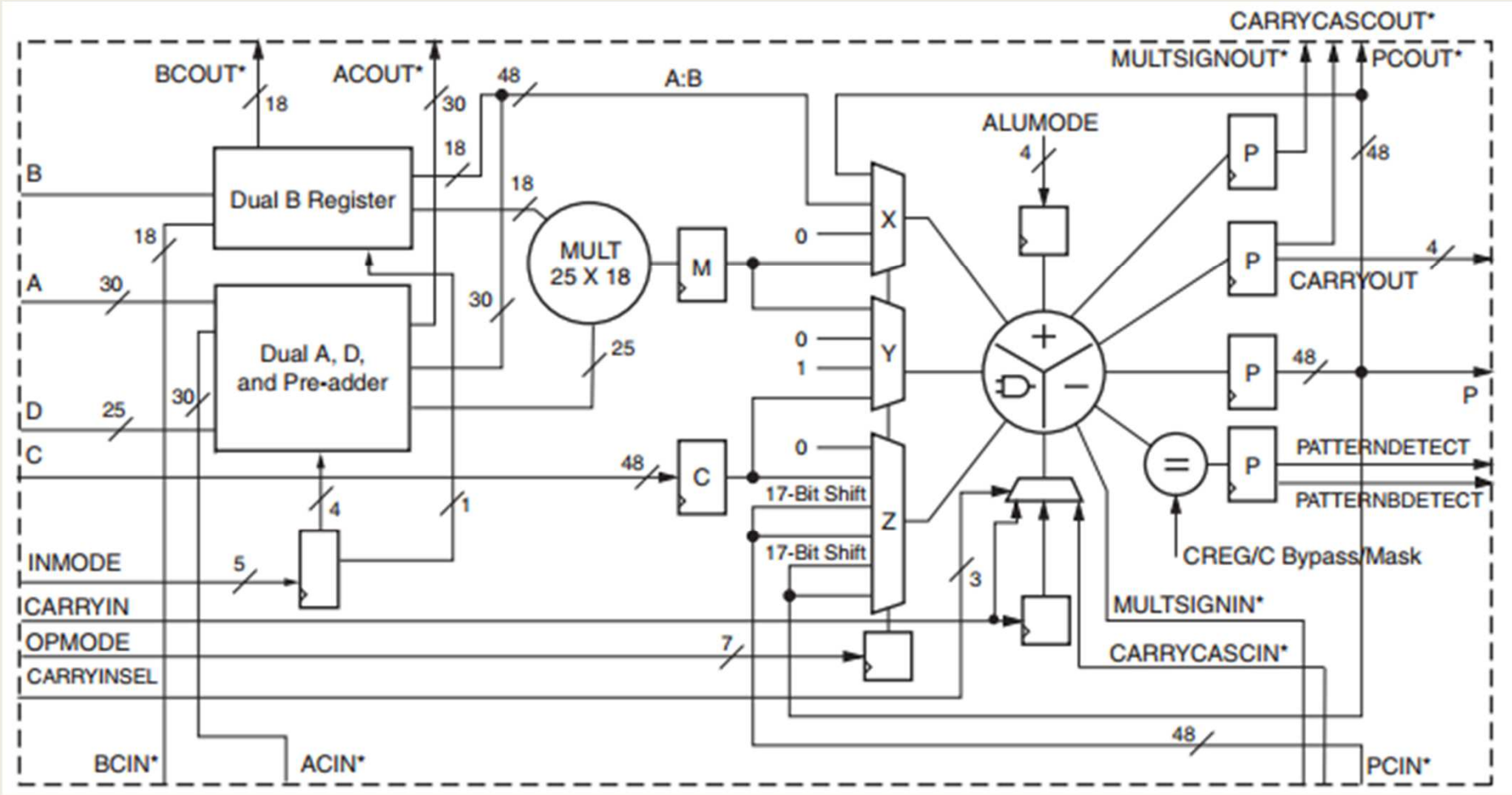


- Main features: 1 adder-subtractor, 1 multiplier, 1 add/sub/logic ALU, 1 comparator, several pipeline stages

Common DSP48 Applications

- Generic multiplication
- Generic Multiply-Accumulate: $y = \sum_i a(i)b(i)$
- FIR Filters: $\mathrm{y(n)} = \sum_j^{N-1} c(j)x(n-j)$
  - symmetric coefficients: $c(j) = \pm c(N-1-j)$
- Barrel shifters
- Replacing fabric adder trees with fast adder cascades

DSP48 Slice

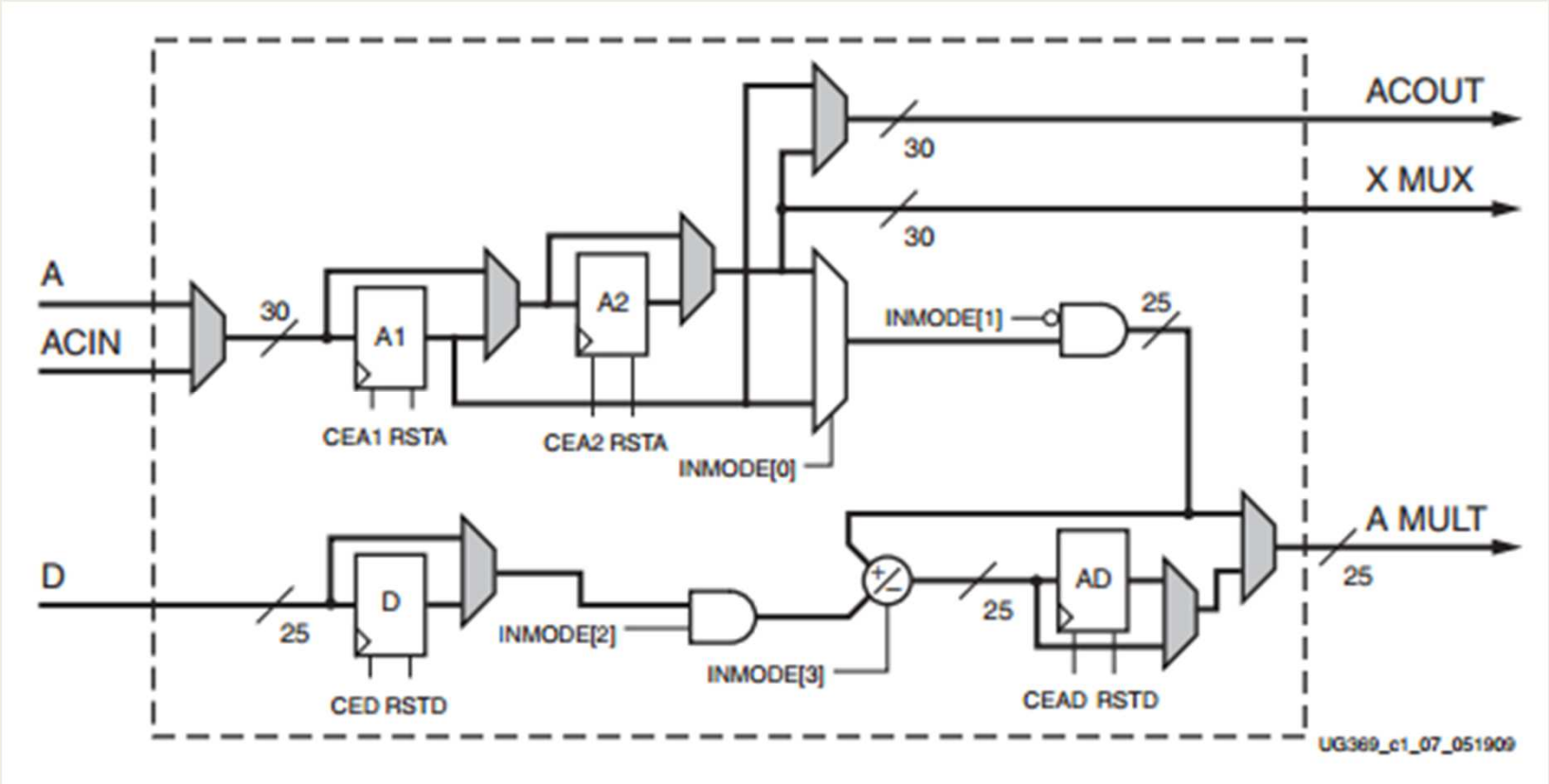

CARRYCASCOUT*
MULTSIGNOUT*
PCOUT*
BCOUT*
ACOUT*
A:B
ALUMODE
B
Dual B Register
MULT
25 X 18
M
X
P
A
Dual A, D,
and Pre-adder
Y
CARRYOUT
D
C
Z
PATTERNDETECT
PATTERNBDETECT
17-Bit Shift
17-Bit Shift
CREG/C Bypass/Mask
INMODE
CARRYIN
OPMODE
CARRYINSEL
MULTSIGNIN*
CARRYCASCIN*
BCIN*
ACIN*
PCIN*

DSP48 Slice

- 4 main control signals:
  - INMODE (5b) controls pre-adders and input registers
  - OPMODE (7b) controls X, Y, Z multiplexers
  - CARRYINSEL (3b) controls source of carry for the final ALU
  - ALUMODE (4b) controls ALU function
- Each has positive clock-enable, synchronous positive reset (OPMODE and CARRYINSEL share control signals)

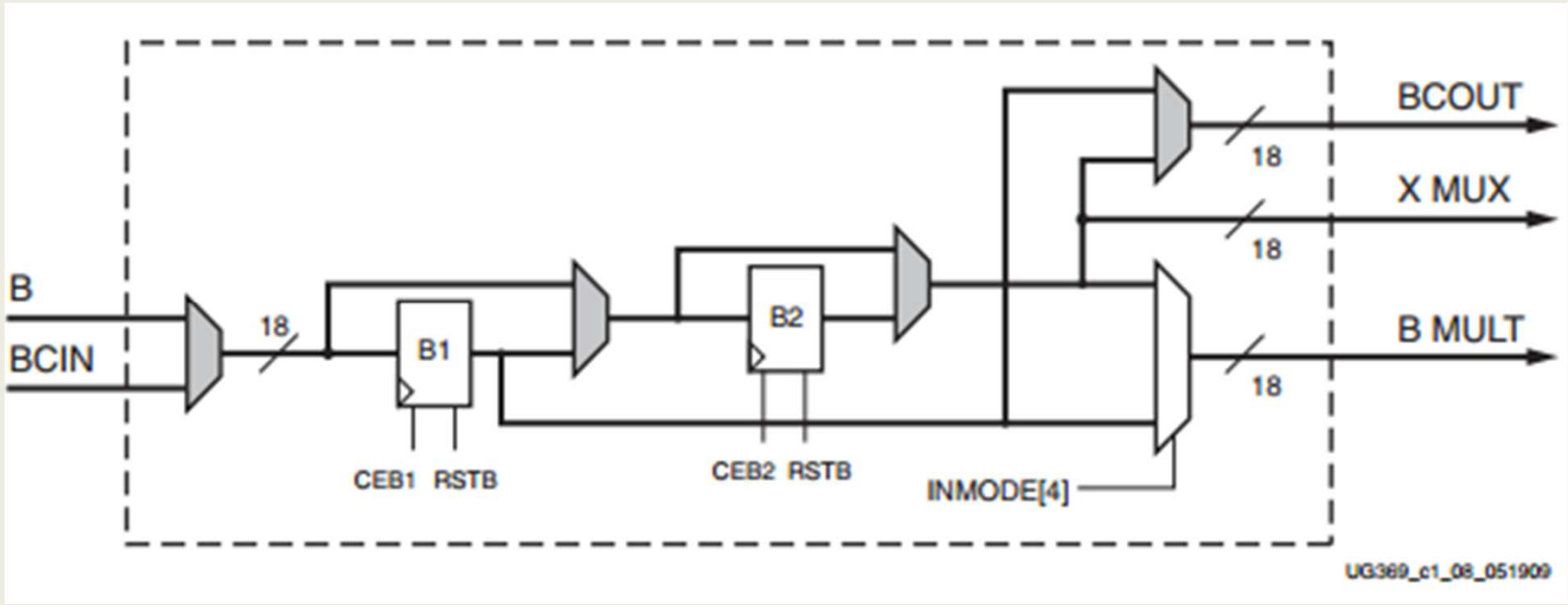
DSP48 Slice: Input Preadder Block

- Preadder on A, D inputs
- Pipeline registers

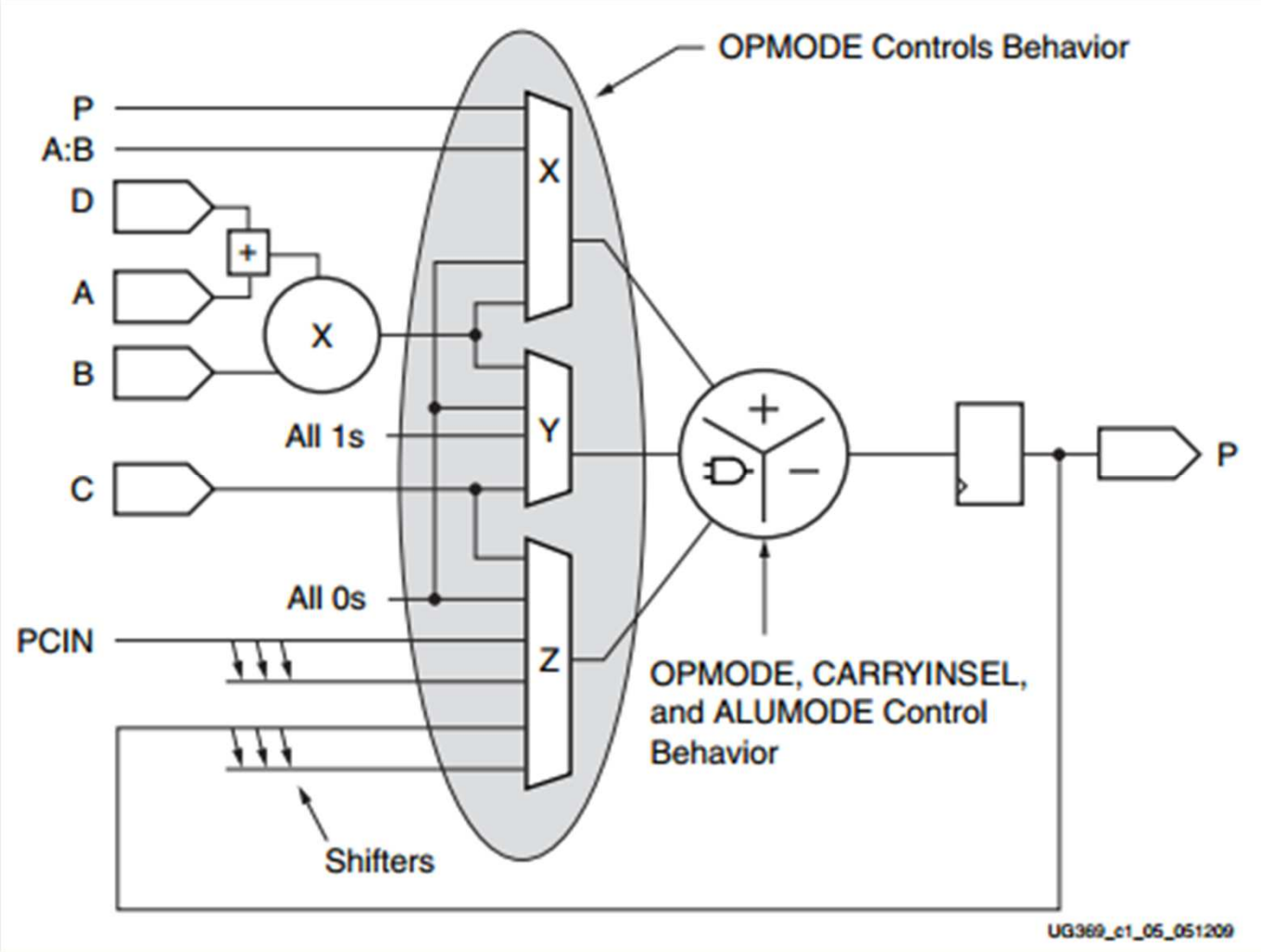


DSP48 Slice: Input Pipeline Block

- Pipeline on on B input



DSP48E1 Slice: Multiplexers



DSP48E1 Slice: ALU

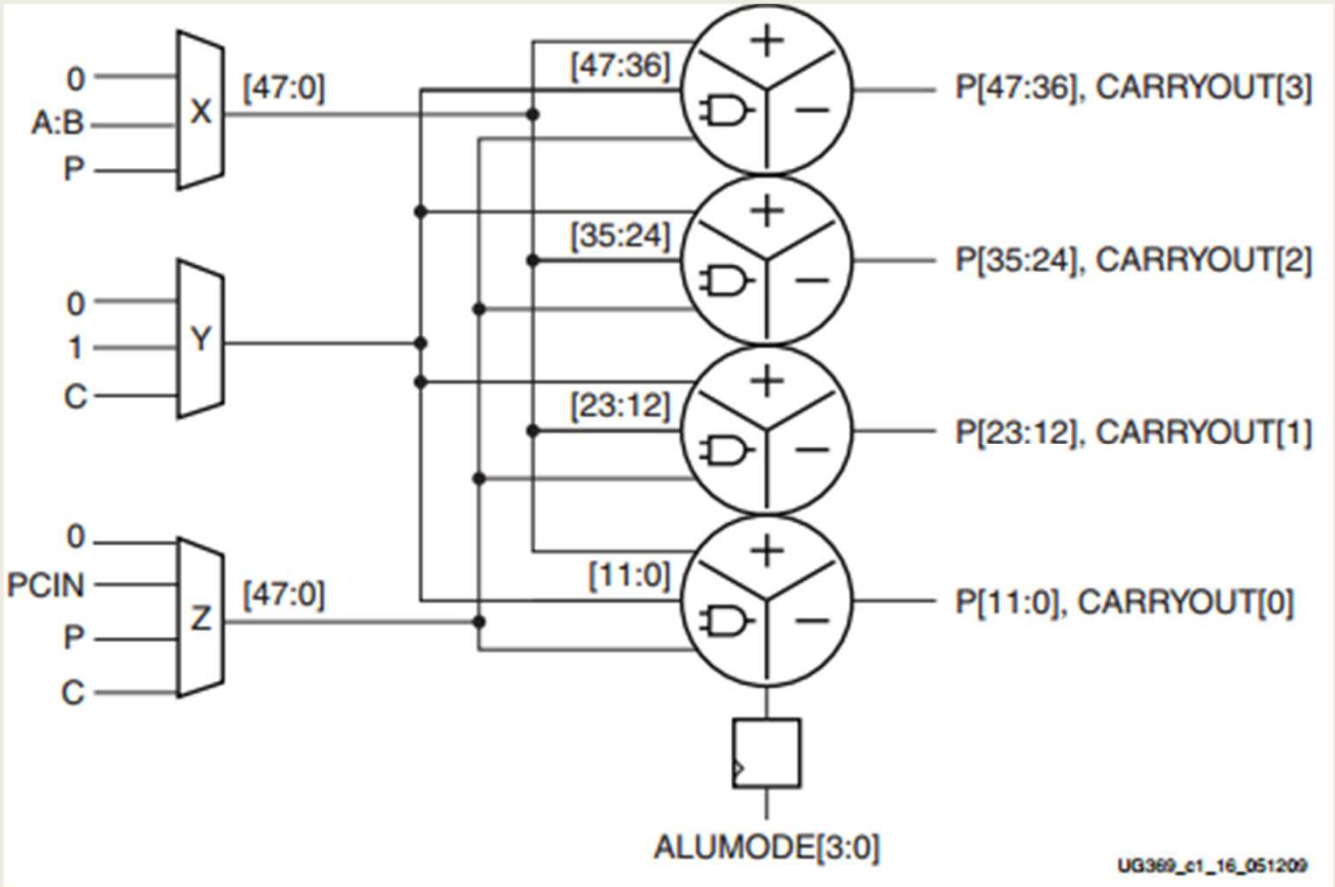
- 2 Modes: 3-input and 2-input
- 3-Input mode (arithmetic):

ALUMODE	Function
4'd0	X+Y+Z+CIN
4'd1	X+Y+CIN-Z-1
4'd2	-X-Y-Z-CIN-1
4'd3	Z-(X+Y+CIN)

- 2-Input mode (logic): 16 logic functions between X and Z

DSP48E1 Slice: ALU

- Arithmetic can be segmented into 2 or 4 SIMD



HDL Guide to Multipliers

- Described using multiplication operator (“*”)
  - Can be combinatorial or registered
- Implemented as fabric (LUTs) or DSP48 slices based on operand size
  - Per-device thresholds
  - Virtex-7: 6x6 bit multiplier in fabric, larger in DSP48
- DSP48 multiplication is natively signed
  - 18x25 bit signed multiplication
  - 17x24 unsigned multiplication (sign bit set to zero)

HDL Guide to Multipliers

- Pipeline registers can be absorbed into the DSP48 slice
  - Up to 2 output pipeline levels
  - Control signals: synchronous positive reset, positive enable

```
always @(posedge clock) begin
    if(reset) begin
            result_r1 <= 0;
            result_r2 <= 0;
    end else if(en) begin
            result_r1 <= op_a * op_b;
            result_r2 <= result_r1;
    end
end
```

HDL Guide to Multipliers

- Preadd-multipliers

```
always @(posedge clock)
    if(reset)
        if(en) begin
            preadd <= op_b + op_c;
            result_r1 <= op_a * preadd;
            result_r2 <= result_r1;
        end
```

- Adder, multiplier and all pipeline stages absorbed into a DSP48 slice
- Same result for pre-subtraction

HDL Guide to Multipliers

- Preaddsub-multipliers (?)

```
always @(posedge clock)
        if(reset) begin
                mult_in_preadd <= 0;
                result_r1 <= 0;
                result_r2 <= 0;
        end else if(en) begin
                if(paddsub)
                        mult_in_preadd <= op_b - op_c;
                else
                        mult_in_preadd <= op_b + op_c;
                result_r1 <= op_a * mult_in_preadd;
                result_r2 <= result_r1;
        end
```

HDL Guide to Multipliers

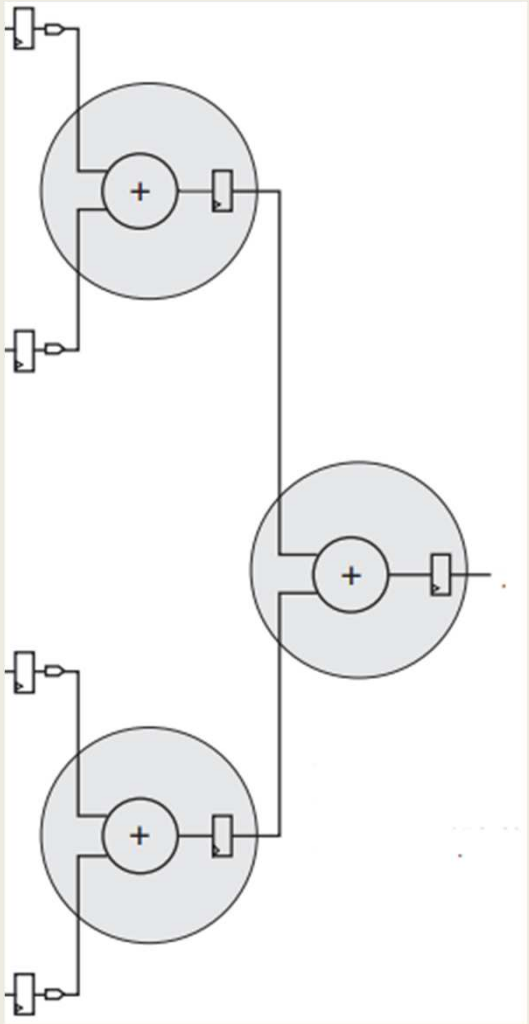
- Preaddsub-multipliers
  - B and C operands do not exceed 25 bits
  - Paddsub control signal selects subtraction when high (fabric inverter inserted if not)
  - Optionally, a clock-enabled register on paddsub input can be absorbed to improve timing
    - Because paddsub controls part of the INMODE register in the DSP48 slice, and INMODE is reset as a whole, reset is not available for paddsub only
    - Use of reset will result in fabric Flip-Flop for paddsub
    - Clock-enable can be dedicated

HDL Guide to MAC

- Multiply-accumulate block with pre-adder
  - Multiplication result can be accumulated into 48-bit result output
  - 32 samples accumulation at maximum multiplication operand size (32-stage FIR)
- Related function: preaddsub-multiply-postaddsub
  - multiplier output added or subtracted from 48-bit input
  - Switching operands results in fabric postaddsub
  - Postaddsub control signals same as preaddsub

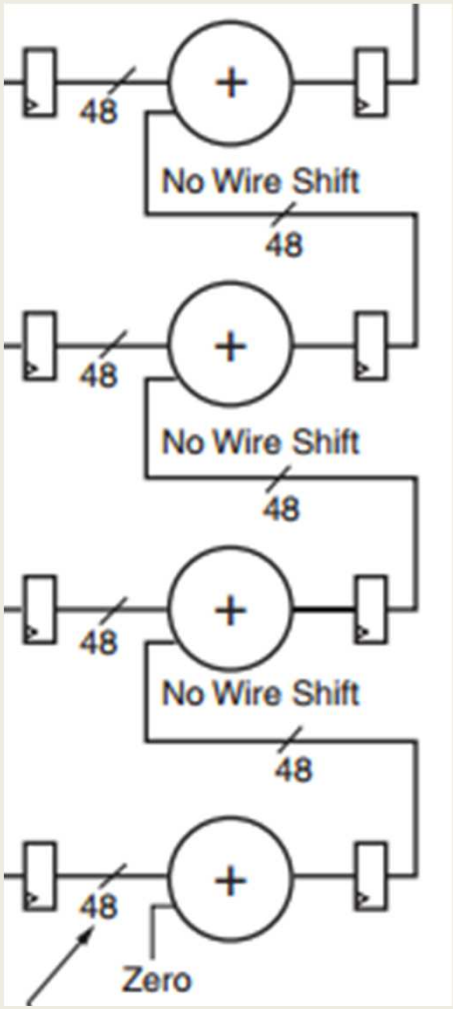
HDL Guide to DSP48 Adder Cascades

- Adder trees can only be implemented in fabric
  - Asymptotically optimal delay O(logN)
  - Slow speed (fmax)
  - Consumes Resources
  - Wastes Power
  - 8-input adder tree (48b): ~4ns delay and 336 LUTs



HDL Guide to DSP48 Adder Cascades

- Adder cascades
  - Non-optimal delay O(N)
  - Fast speed (fmax)
  - Requires only DSP48 resources
  - 8-input, 48b adder tree converts to 7-stage adder cascade @700MHz: ~10ns delay (7 DSP48 slices)



HDL Guide to DSP48 Barrel Shifters

- Traditional way to code barrel shifter:

  ```
  assign sh = (input << amount) | (input >> amount)
  ```

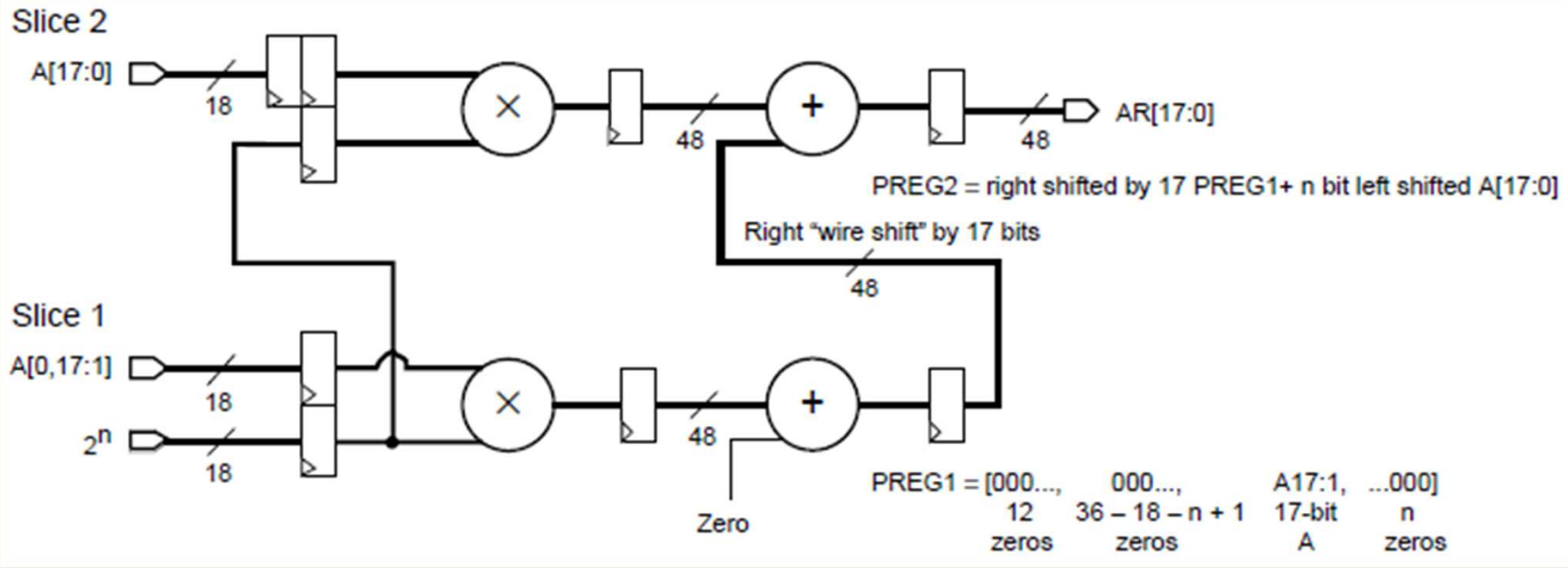
- 16-bit input results in 48-LUT implementation
- Idea: use shift-by-multiply:

  ```
  assign mult = input * (1 << amount)
  assign sh = mult[15:0] | mult[31:16]
  ```

- Implemented using 1 DSP48 and 32 LUTs
- If amount is already decoded: 1 DSP48, 16 LUTs

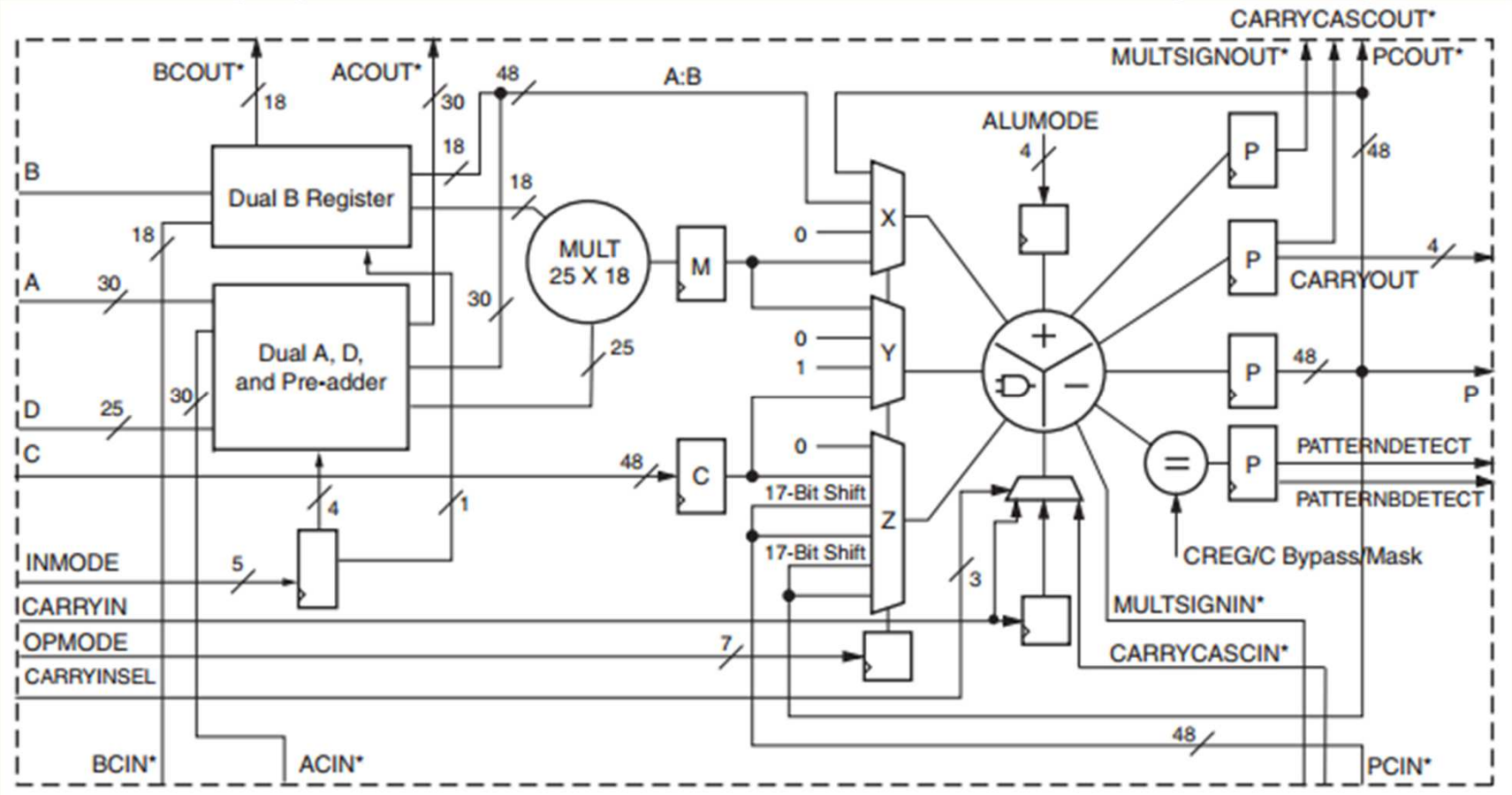
HDL Guide to DSP48 Barrel Shifters

- Eliminate LUTs by using DSP48 built-in wire shift and adders



HDL Guide to DSP48 Barrel Shifters

- Implement in single DSP48 slice using multiply-accumulate and time multiplexing



Exercise: Complex Number Multiplier using DSP48 Slices

- X = a + bj
- Y = c + dj
- Compute X*Y and report resource utilization, fmax

Further Reading

- Virtex-4 FPGA XtremeDSP User Guide
- 7-Series FPGA DSP Resources User Guide
- DSP: Designing for Optimal Results