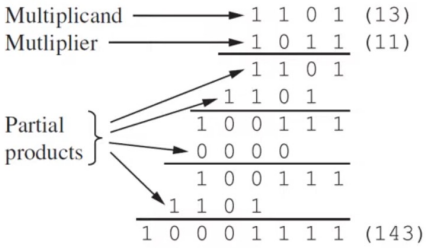
4.9 Shift and Add Multiplier

A Shift-and-Add Multiplier

- Problem description:
  - Design a multiplier for **unsigned** binary numbers.
  - Form the ***product*** A × B
    - the 1st operand (*A*): ***multiplicand***,
    - the 2nd operand (*B*): ***multiplier***.
- ***Serial-parallel multiplier***:
  - The multiplier bits are processed serially, but addition takes place in parallel.

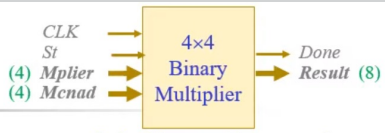
Multiplication for *Unsigned* Binary Numbers

- Multiplication for ***unsigned*** binary numbers:
  - E.g.:

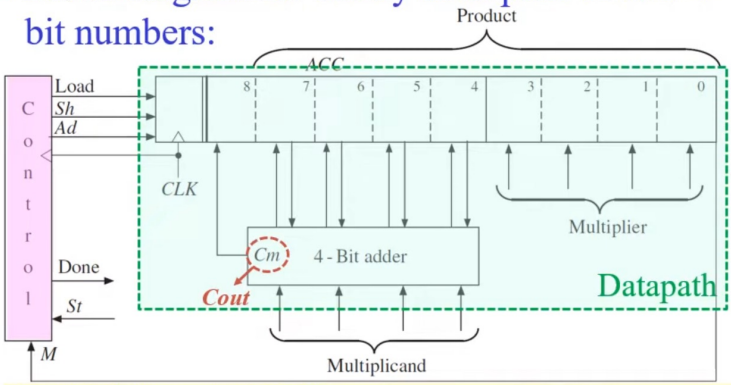


  - Binary multiplication requires only ***shifting*** and ***adding***.
  - Each ***partial product*** is either ***the multiplicand shifted over by the appropriate # of places*** or ***zero***.
  - Each new partial product is added in as soon as it is formed.

Block Diagram of 4×4 Multiplier

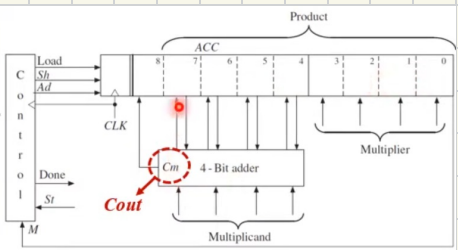


- Block diagram for binary multiplier of two 4-bit numbers:



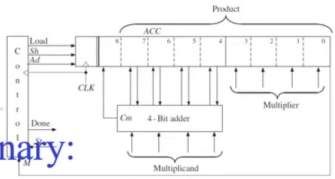
St: start signal; ***M***: current multiplier bit
Load: load multiplier into the lower 4 bits of ACC and clear the upper 5 bits of ACC
Sh: shift signal, shift all 9 bits of ACC to the **right** by the next clock pulse
Ad: add signal, transfers the adder outputs to the upper 5 bits of ACC by the next clock pulse
Done: done signal

St: start signal
M: current multiplier bit
Load: load multiplier into ACC[3:0] and clear the ACC[8:4]
Sh: shift signal, shift ACC[8:0] to the **right** by the next clock pulse
Ad: add signal, transfers the adder outputs to the ACC[8:4] by the next clock pulse
Done: done signal



- a 4-bit multiplicand
- a 4-bit multiplier register
- a 4-bit full adder
- an 8-bit register for the product
  - The product register serves as an accumulator to accumulate the sum of the ***partial product***.
  - The contents of the product register will be shift to the right each time before added with the multiplicand.
- an extra bit at the left end of the product register
  - temporarily stores any carry generated by the adder

Example



- Multiply 13_{10} by 11_{10} in binary:
  - the location of the bits in the regs at clock time:

initial contents of product register	0 0 0 0 0 1 0 1 1 ← M	(11)
(add multiplicand since $M = 1$)	1 1 0 1	(13)
after addition	0 1 1 0 1 1 0 1 1	
after shift	0 0 1 1 0 1 1 0 1 ← M	
(add multiplicand since $M = 1$)	1 1 0 1	
after addition	1 0 0 1 1 1 1 0 1	
after shift	0 1 0 0 1 1 1 1 0 ← M	
(skip addition since $M = 0$)		
after shift	0 0 1 0 0 1 1 1 1 ← M	
(add multiplicand since $M = 1$)	1 1 0 1	
after addition	1 0 0 0 1 1 1 1 1	
after shift (final answer)	0 1 0 0 0 1 1 1 1	(143)

dividing line between product and multiplier

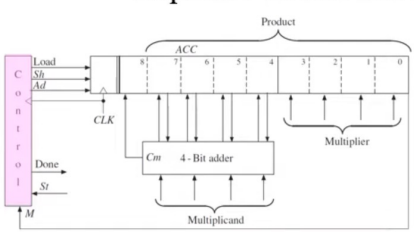
· First Controller Design

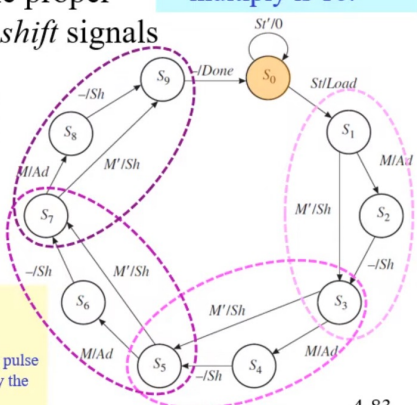
State Graph of 4×4 Multiplier

- State graph for control ckt:
  - designed to output the proper sequence of *add* and *shift* signals

* The max # of clock cycles needed for a multiply is 10.

← 2(n+1) where n = # (bits to add)

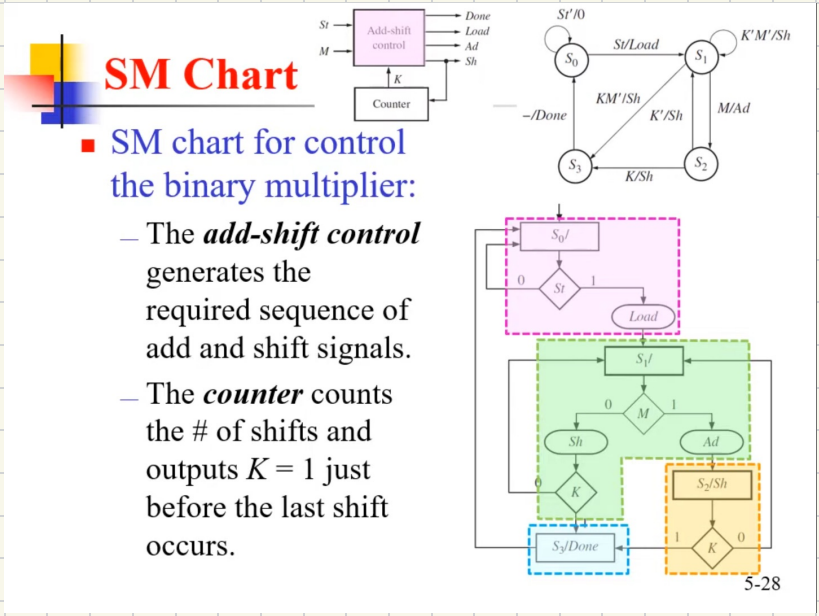




St: start signal
M: current multiplier bit
Load: load multiplier into ACC[3:0] and clear the ACC[8:4]
Sh: shift signal, shift ACC[8:0] to the right by the next clock pulse
Ad: add signal, transfers the adder outputs to the ACC[8:4] by the next clock pulse
Done: done signal

↖ this state machine design is bad when # (bits to add) is large

- Second Controller Design



- SM chart for control the binary multiplier:
  - The ***add-shift control*** generates the required sequence of add and shift signals.
  - The ***counter*** counts the # of shifts and outputs $K = 1$ just before the last shift occurs.