Lecture 10 (Chapter 4)

Design Examples: Add-shift Multipliers

Khaza Anuarul Hoque
ECE 4250/7250

Introduction

- In any design:
  - State the problem.
  - Obtain clear design specifications.
  - Define basic building blocks necessary to accomplish what is specified, e.g., adders, shift registers, counters, etc.

Introduction (continued)

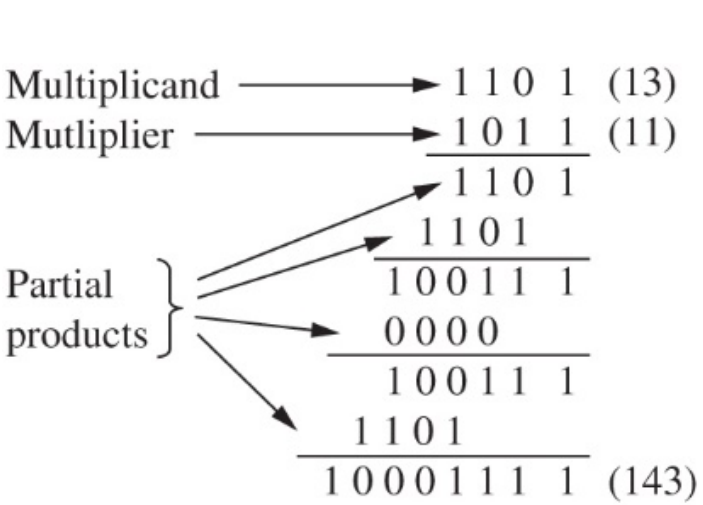
- Traditional design methodology:
  - **Data path**: hardware that performs the data processing.
    - For microprocessor: the arithmetic logic unit (ALU)
  - **Controller**: sends control signals to the data path and can obtain feedback in the form of status signals from the data path.
    - Many modifications can be made here.

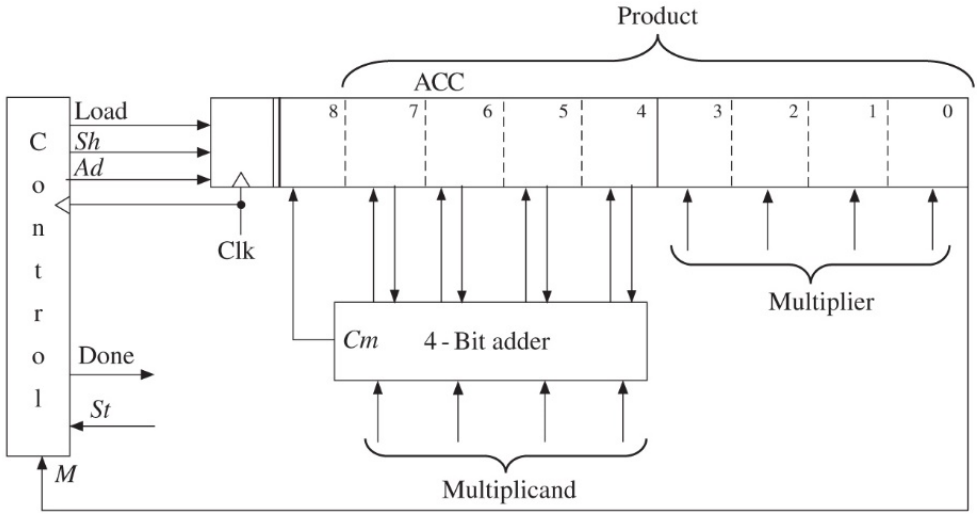
Add-and-Shift Multiplier

- In A × B, the first operand (A) is called the **multiplicand**, and the second operand (B) is called the **multiplier**.
- Binary multiplication process:
  - Multiplicand is shifted.
  - Next bit of multiplier is examined (also a shifting step).
  - If this bit is 1, the shifted multiplicand is added to the product.

Add-and-Shift Multiplier (continued)

- Example: multiply 13_{10} by 11_{10} in binary:

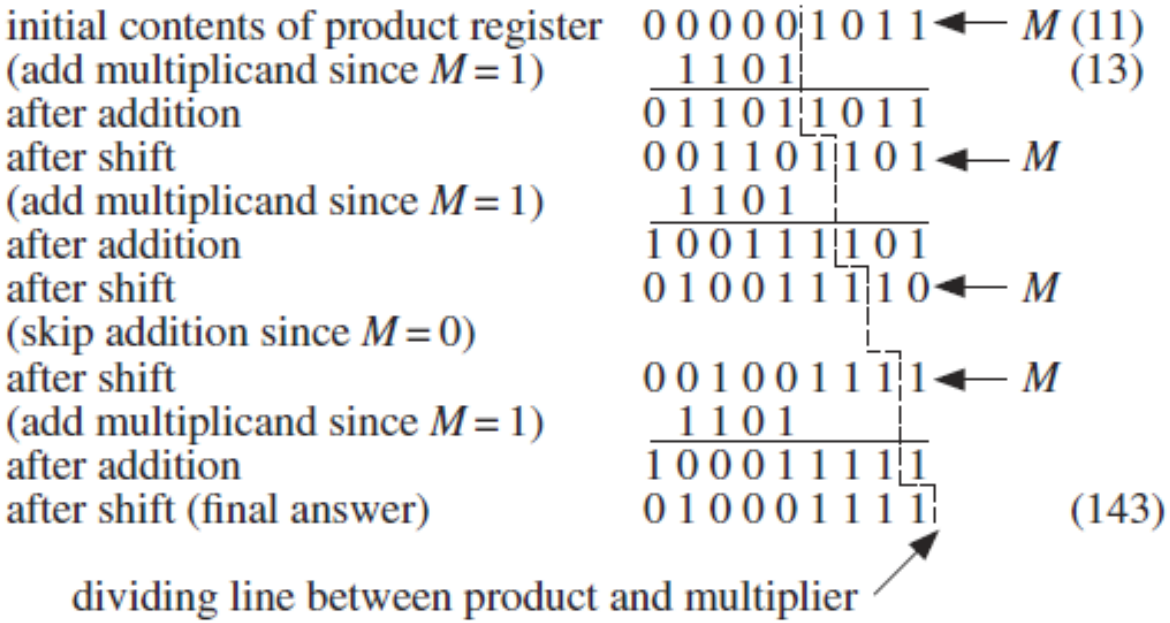




  - Sometimes called a serial-parallel multiplier, as the multiplier bits are processed serially while addition takes place in parallel.

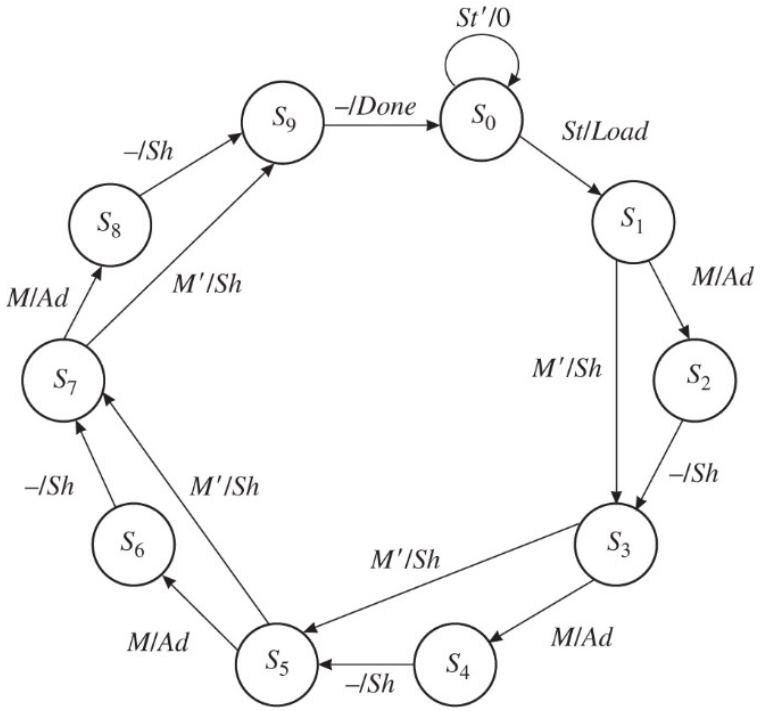
Add-and-Shift Multiplier (continued)

- Example reworked to show the location of the bits in the registers at clock time:



Add-and-Shift Multiplier (continued)

- State graph for control circuit (designed to output the proper sequence of add and shift signals):



Behavioral model of a 4 x 4 binary ADD-Shift multiplier in VHDL

```
entity mult4X4 is
  port(Clk, St: in bit;
       Mplier, Mcand: in unsigned(3 downto 0);
       Done: out bit);
end mult4x4;

architecture behave1 of mult4x4 is
signal State: integer range 0 to 9;
signal ACC: unsigned(8 downto 0); -- accumulator
alias M: bit is ACC(0);      -- M is bit 0 of ACC
begin
  process(Clk)
  begin
    if Clk'event and Clk = '1' then  -- executes on rising edge of clock
      case State is
        when 0 =>                -- initial State
          if St = '1' then
            ACC(8 downto 4) <= "00000"; -- begin cycle
            ACC(3 downto 0) <= Mplier;  -- load the multiplier
            State <= 1;
          end if;
        when 1 | 3 | 5 | 7 =>      -- "add/shift" State
          if M = '1' then          -- add multiplicand
            ACC(8 downto 4) <= '0' & ACC(7 downto 4) + Mcand;
            State <= State + 1;
          else
            ACC <= '0' & ACC(8 downto 1);        -- shift accumulator right
            State <= State + 2;
          end if;
        when 2 | 4 | 6 | 8 =>                    -- "shift" State
          ACC <= '0' & ACC(8 downto 1);          -- right shift
          State <= State + 1;
        when 9 =>                                -- end of cycle
          State <= 0;
      end case;
    end if;
  end process;
  Done <= '1' when State = 9 else '0';
end behave1;
```