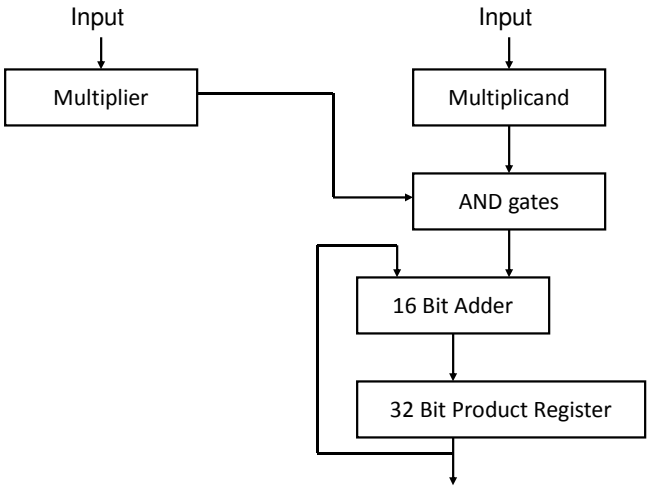
Multiplication – Simple Gradeschool Algorithm for 16 Bits (32 Bit Result)



Multiplication – Simple Gradeschool Algorithm for 16 Bits (32 Bit Result)

Step 1:
- Clear Product Register
- Clear Counter
- Load Multiplicand
- Load Multiplier

Step 2: (repeat 16 times)
- Increment Counter
- Load Product Register
- Shift Multiplier

Step 3:
- Done

VHDL: Use Processes, Buses to Implement Registers, Logic

- Process construct for registers: clear, clock activities
- Test conditions in system (both data path and control) with 'if' constructs
- Implement logic as needed in either process or signal assignment statements

Port in Entity

```
entity MULTIPLIER is
  port (
    MIER_VAL : in  STD_LOGIC_VECTOR ( 15 downto 0 );
    MCND_VAL : in  STD_LOGIC_VECTOR ( 15 downto 0 );
    PROD_VAL : out STD_LOGIC_VECTOR ( 31 downto 0 );
    SYS_CLK  : in  STD_LOGIC;
    SYS_RST  : in  STD_LOGIC;
    GO       : in  STD_LOGIC;
    DONE     : out STD_LOGIC
  );
end entity MULTIPLIER;
```

Multiplicand is Simple Register

```
in declaration area:

  signal MCAND : STD_LOGIC_VECTOR ( 15 downto 0 );

in work area:

process ( SYS_CLK, SYS_RST ) is
begin
  if SYS_RST = '1' then
    MCAND <= ( others => '0' );
  elsif RISING_EDGE ( SYS_CLK ) then
    if PSR = S1 then
      MCAND <= MCND_VAL;
    end if;
  end if;
end process;
```

Multiplier is Shift Register with Parallel Load

```
in declaration area:
  signal MIER : STD_LOGIC_VECTOR ( 15 downto 0 );
in work area:

process ( SYS_CLK, SYS_RST ) is
begin
  if SYS_RST = '1' then
    MIER <= ( others => '0' );
  elsif RISING_EDGE ( SYS_CLK ) then
    if PSR = S1 then
      MIER <= MIER_VAL;
    elsif PSR = S2 then
      MIER <= '0' & MIER ( 15 downto 1 );
    end if;
  end if;
end process;
```

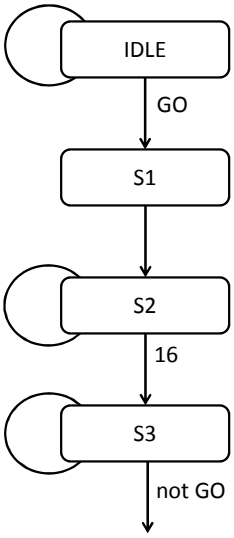
AND Function, Adder as Signal Assignment Statements

```
in declaration area:

  signal AND_OUT : STD_LOGIC_VECTOR ( 15 downto 0 );
  signal ADDER   : STD_LOGIC_VECTOR ( 16 downto 0 );

in work area;

  AND_OUT <= MCAND when MIER(0) = '1' else
             ( others => '0' );

  ADDER <= ( '0' & PRODUCT ( 31 downto 16 ) ) +
           ( '0' & AND_OUT                  );
```

Product is Simple Register

```
in declaration area:
  signal PRODUCT : STD_LOGIC_VECTOR ( 31 downto 0 );
in work area:

process ( SYS_CLK, SYS_RST ) is
begin
  if SYS_RST = '1' then
    PRODUCT <= ( others => '0' );
  elsif RISING_EDGE ( SYS_CLK ) then
    if PSR = S1 then
      PRODUCT <= ( others => '0' );
    elsif PSR = S2 then
      PRODUCT <= ADDER & PRODUCT ( 15 downto 1 );
    end if;
  end if;
end process;
```

State Machine Implements Sequence



Define States as Enumeration Type

in declaration area:

```
type PSR_TYPE is ( IDLE, S1, S2, S3 );
signal PSR : PSR_TYPE;
```

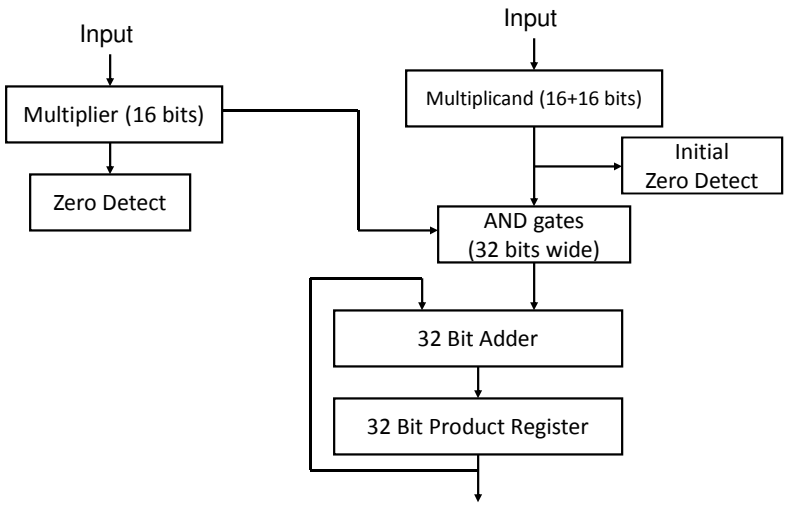
Implement State Machine as Process

```
process ( SYS_CLK, SYS_RST ) is
begin
  if SYS_RST = '1' then
    PSR <= IDLE;
  elsif RISING_EDGE ( SYS_CLK ) then
    case PSR is
      when IDLE =>
        if GO = '1' then PSR <= S1; end if;
      when S1 => PSR <= S2;
      when S2 => if COUNT = 15 then PSR <= S3; end if;
      when S3 => if GO = '0' then PSR <= IDLE; end if;
    end case;
  end if;
end process;
```

Implement Counter as Process

```
in declaration area:
  signal COUNT : STD_LOGIC_VECTOR ( 4 downto 0 );
in work area:

process ( SYS_CLK, SYS_RST ) is
begin
  if SYS_RST = '1' then
    COUNT <= ( others => '0' );
  elsif RISING_EDGE ( SYS_CLK ) then
    if PSR = S1 then
      COUNT <= ( others => '0' );
    elsif PSR = S2 then
      COUNT <= COUNT + 1 ;
    end if;
  end if;
end process;
```

Implement the Modified Gradeschool Multiplication Algorithm



Multiplication – Modified Gradeschool Algorithm for 16 Bits (32 Bit Result)

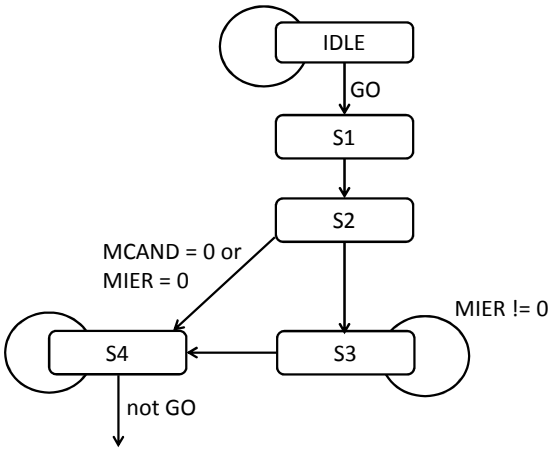
Step 1:
- Clear Product Register
- Load Multiplicand
- Load Multiplier

Step 2:
- If MIER = 0 OR MCAND = 0, done

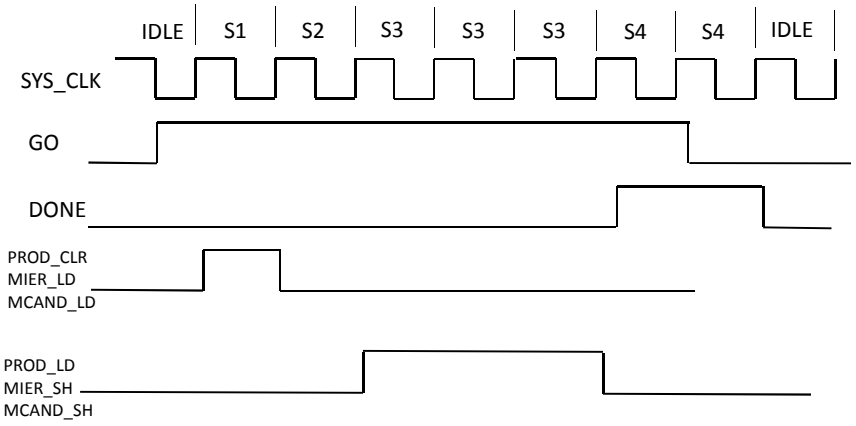
Step 3:
- Load Product Register
- Shift Multiplier and Multiplicand

Step 4:
- If MIER = 0, then Done
- else Go to Step 3

State Machine Implements Sequence



Timing for Multiply



Multiplicand Is Load/Shift Register

```
in declaration area:
  signal MCAND : STD_LOGIC_VECTOR ( 31 downto 0 );
in work area:

process ( SYS_CLK, SYS_RST ) is
begin
  if SYS_RST = '1' then
    MCAND <= ( others => '0' );
  elsif RISING_EDGE ( SYS_CLK ) then
    if PSR = S1 then
      MCAND <= ZEROS & MCND_VAL;
    elsif PSR = S3 then
      MCND <= MCND ( 30 downto 0 ) & '0';
    end if;
  end if;
end process;
```

Multiplier Is Again Shift Register with Parallel Load

```
in declaration area:
  signal MIER : STD_LOGIC_VECTOR ( 15 downto 0 );
in work area:

process ( SYS_CLK, SYS_RST ) is
begin
  if SYS_RST = '1' then
    MIER <= ( others => '0' );
  elsif RISING_EDGE ( SYS_CLK ) then
    if PSR = S1 then
      MIER <= MIER_VAL;
    elsif PSR = S3 then
      MIER <= '0' & MIER ( 15 downto 1 );
    end if;
  end if;
end process;
```

AND Function, Adder as before - Signal Assignment Statements

```
in declaration area:

  signal AND_OUT : STD_LOGIC_VECTOR ( 31 downto 0 );
  signal ADDER   : STD_LOGIC_VECTOR ( 31 downto 0 );

in work area;

  AND_OUT <= MCAND when MIER(0) = '1' else
             ( others => '0' );

  ADDER <=  PRODUCT + AND_OUT;
```

Product Is Again Simple Register

```
in declaration area:
  signal PRODUCT : STD_LOGIC_VECTOR ( 31 downto 0 );
in work area:

process ( SYS_CLK, SYS_RST ) is
begin
  if SYS_RST = '1' then
    PRODUCT <= ( others => '0' );
  elsif RISING_EDGE ( SYS_CLK ) then
    if PSR = S1 then
      PRODUCT <= ( others => '0' );
    elsif PSR = S3 then
      PRODUCT <= ADDER;
    end if;
  end if;
end process;
```

Testing for Zero: Conditional Signal Assignment Statements

```
in declaration area:

  signal MCND_ZRO : STD_LOGIC;
  signal MIER_ZRO : STD_LOGIC;

in work area;

  MIER_ZRO <= '1' when MIER = X"0000" else '0';

  MCND_ZRO <= '1' when MCAND ( 15 downto 0 ) =
                               X"0000" else '0';
```