Floating-Point Arithmetic

ELEC 418

Advanced Digital Systems

Dr. Ron Hayne

Images Courtesy of Thomson Engineering

THE CITADEL
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

IEEE 754 Floating-Point

- **Fraction**
  - Sign-Magnitude
- **Exponent**
  - Biased Notation

- **IEEE Single Precision Format**

S	Exponent	Fraction
1 bit	8 bits	23 bits

- **IEEE Double Precision Format**

S	Exponent	Fraction
1 bit	11 bits	52 bits

Special Cases

Single Precision		Double Precision		Object Represented
Exponent	Fraction	Exponent	Fraction	
0	0	0	0	0
0	nonzero	0	nonzero	± denormalized number
255	0	2047	0	± infinity
255	nonzero	2047	nonzero	NaN (not a number)

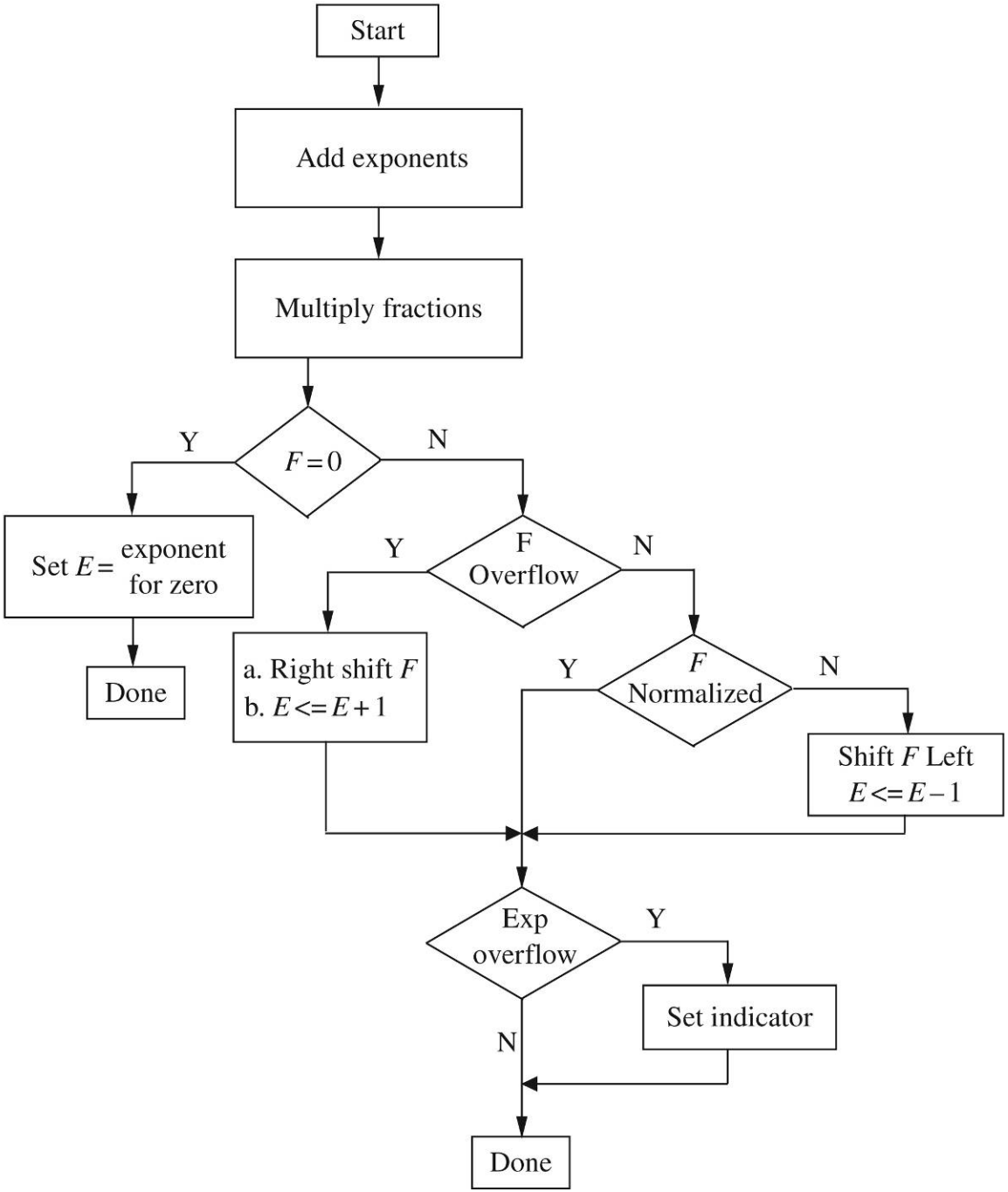
A Simple Floating-Point Format

- **N = F x 2^E**
  - Fraction
    - 4-bit 2's Complement
    - s.fff
    - Range -1 to +0.875
  - Exponent
    - 4-bit 2's Complement
    - eeee
    - Range -8 to +7
  - Normalization
    - Shift left until sign bit and next bit are different
  - Zero
    - F = 0.000
    - E = 1000
    - N = 0.000 x 2^{-8}

Floating-Point Multiplication

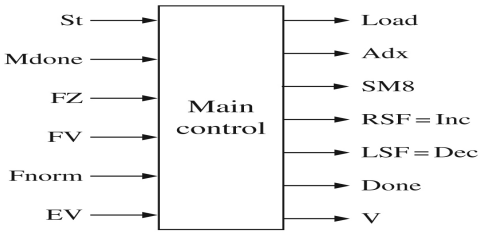
$$(F_1 \times 2^{E_1}) \times (F_2 \times 2^{E_2}) = (F_1 \times F_2) \times 2^{(E_1+E_2)}$$
$$= F \times 2^{E}$$

1. Add exponents
2. Multiply fractions
3. If product is 0, adjust for proper 0
4. Normalize product fraction
5. Check for exponent overflow or underflow
6. Round product fraction

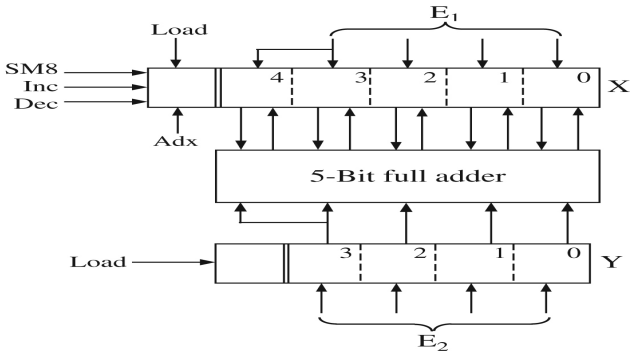

Start
Add exponents
Multiply fractions
F = 0
Y
N
Set E = exponent for zero
Done
F Overflow
Y
N
a. Right shift F
b. E <= E + 1
F Normalized
Y
N
Shift F Left
E <= E – 1
Exp overflow
Y
Set indicator
N
Done

Floating-Point Hardware

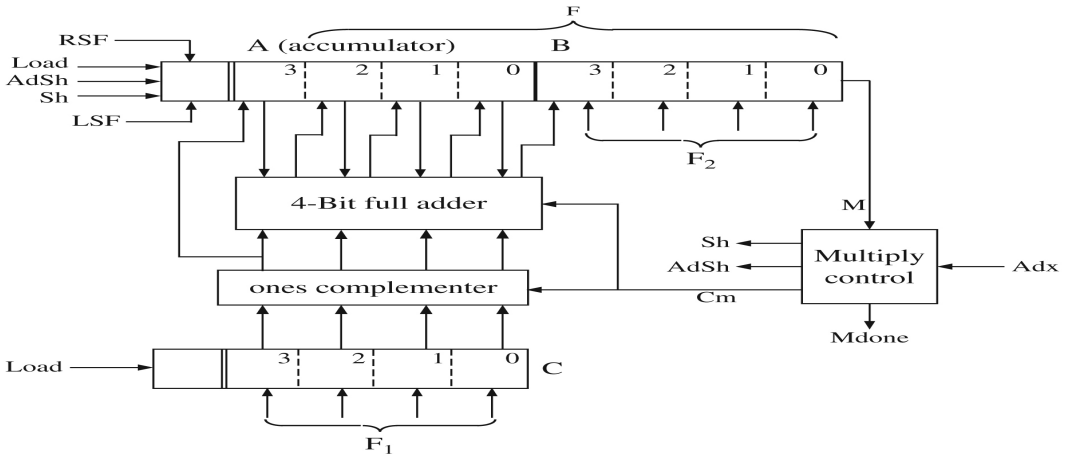
FIGURE 7-7: Major Components of a Floating-Point Multiplier



(a) Main control unit



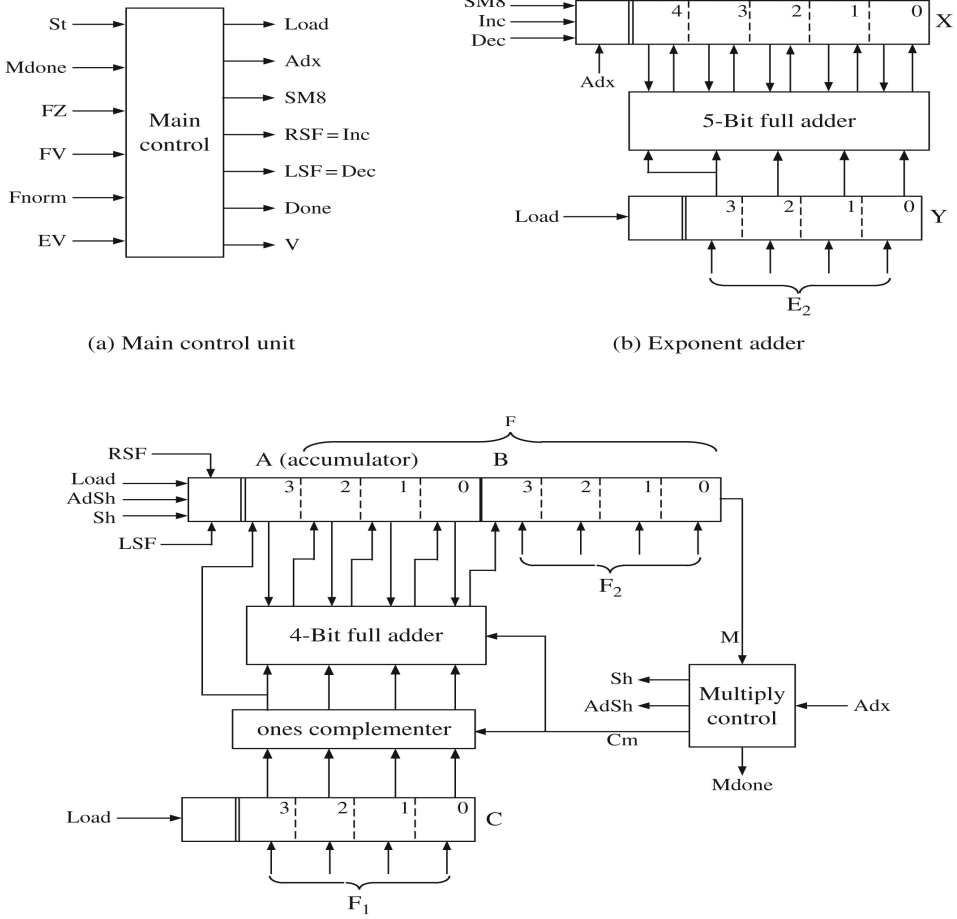
(b) Exponent adder



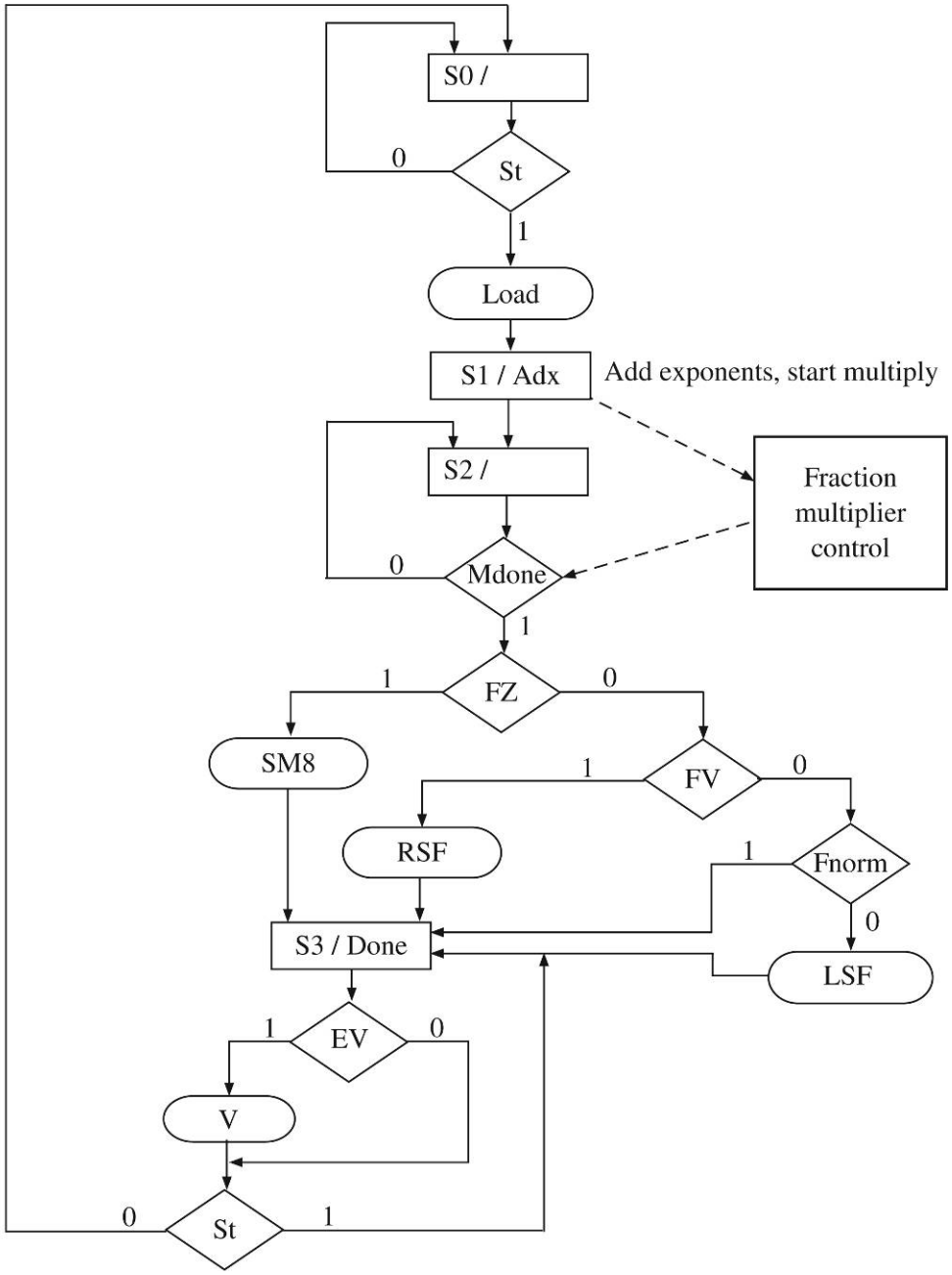
(c) Fraction multiplier

Floating-Point Hardware

FIGURE 7-7: Major Components of a Floating-Point Multiplier



SM Chart


S0 /
0
St
1
Load
S1 / Adx
Add exponents, start multiply
S2 /
Fraction multiplier control
0
Mdone
1
1
FZ
0
SM8
1
FV
0
RSF
1
Fnorm
0
S3 / Done
LSF
1
EV
0
V
0
St
1

VHDL Model

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity FMUL is
 port(CLK, St: in std_logic;
      F1, E1, F2, E2: in std_logic_vector(3 downto 0);
      F: out std_logic_vector(6 downto 0);
      E: out std_logic_vector(4 downto 0);
      V, done: out std_logic);
end FMUL2;
```

VHDL Model

```
architecture BEHAVE of FMUL is

signal A, B, C: std_logic_vector(3 downto 0);--F regs
signal compout, addout: std_logic_vector(3 downto 0);
alias M: std_logic is B(0);
signal X, Y: std_logic_vector(4 downto 0);  -- E regs

signal Load, Adx, SM8, RSF, LSF: std_logic; -- Main
signal AdSh, Sh, Cm, Mdone: std_logic; -- Mult

signal PS1, NS1: integer range 0 to 3;-- Main state
signal State, Nextstate: integer range 0 to 4;-- Mult
```

VHDL Model

```
begin
  main_control: process(PS1, St , Mdone, X, A, B)
  begin
    Load <= '0'; Adx <= '0'; NS1 <= 0; SM8 <= '0';
    RSF <= '0'; LSF <= '0'; V <= '0'; done <= '0';
    case PS1 is
      when 0 =>
        if St = '1' then
          Load <= '1';
          NS1 <= 1;
        end if;
      when 1 =>
        Adx <= '1';
        NS1 <= 2;
```

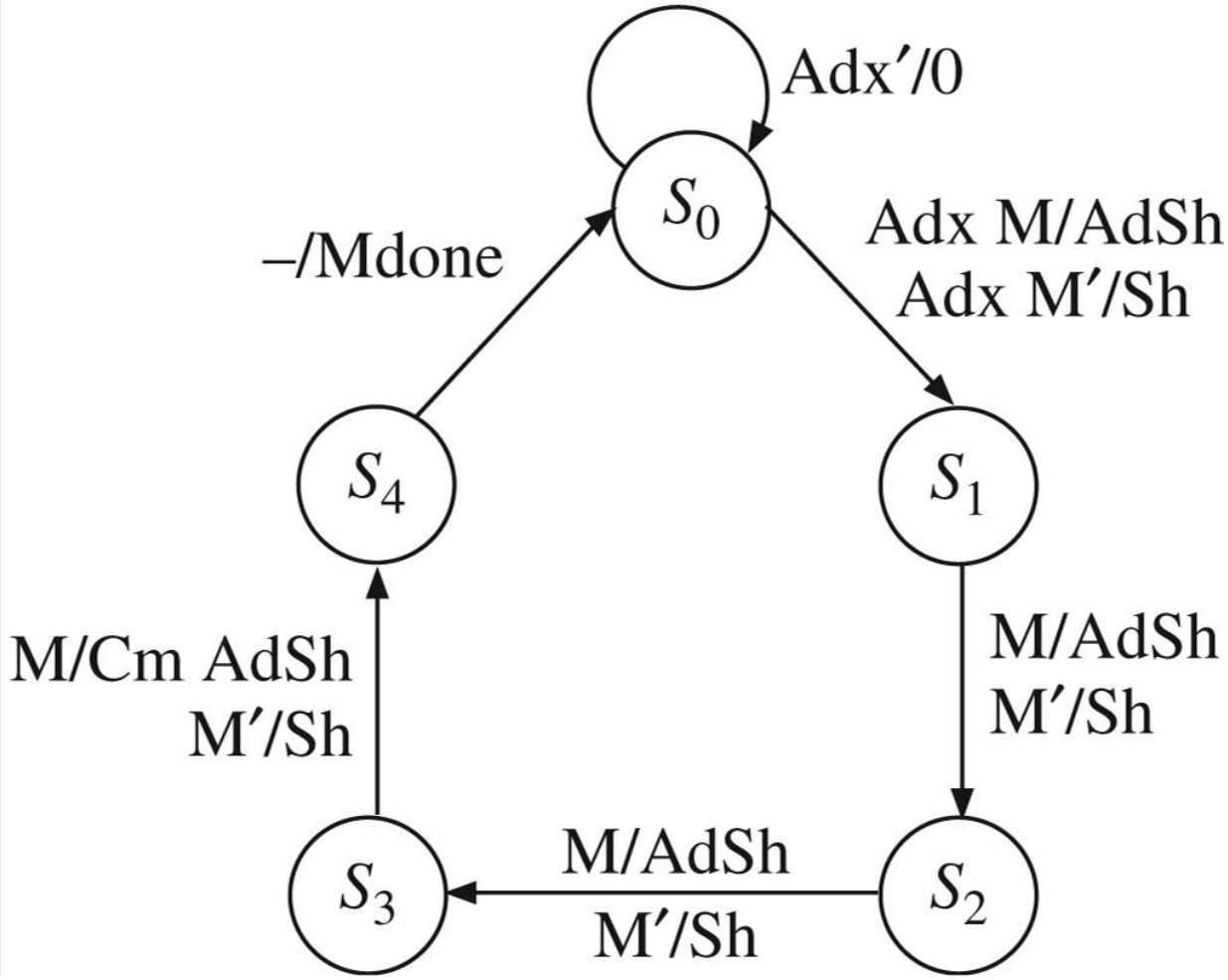
VHDL Model

```
when 2 =>
  if Mdone = '1' then
    if A = 0 then  -- FZ
      SM8 <= '1';
    elsif A = 4 and B = 0 then -- FV
      RSF <= '1';
    elsif A(2) = A(1) then -- not Fnorm
      LSF <= '1';
    end if;
    NS1 <= 3;
  else
    NS1 <= 2;
  end if;
```

VHDL Model

```
    when 3 =>
      done <= '1';
      if X(4) /= X(3) then -- EV
        V <= '1';
      end if;
      if ST = '0' then
        NS1 <= 0;
      end if;
    end case;
end process main_control;
```

Multiplier Control



VHDL Model

```
mul2c: process(State, Adx, M)
begin
  AdSh <= '0'; Sh <= '0'; Cm <= '0'; Mdone <= '0';
  Nextstate <= 0;
  case State is
    when 0 =>
      if Adx = '1' then
        if M = '1' then
          AdSh <= '1';
        else
          Sh <= '1';
        end if;
        Nextstate <= 1;
      end if;
```

VHDL Model

```
when 1 | 2 =>
  if M = '1' then
    AdSh <= '1';
  else
    Sh <= '1';
  end if;
  Nextstate <= State + 1;
```

VHDL Model

```
    when 3 =>
      if M = '1' then
        Cm <= '1';
        AdSh <= '1';
      else
        Sh <='1';
      end if;
      Nextstate <= 4;
    when 4 =>
      Mdone <= '1';
      Nextstate <= 0;
  end case;
end process mul2c;
```

Data Path

FIGURE 7-7: Major Components of a Floating-Point Multiplier

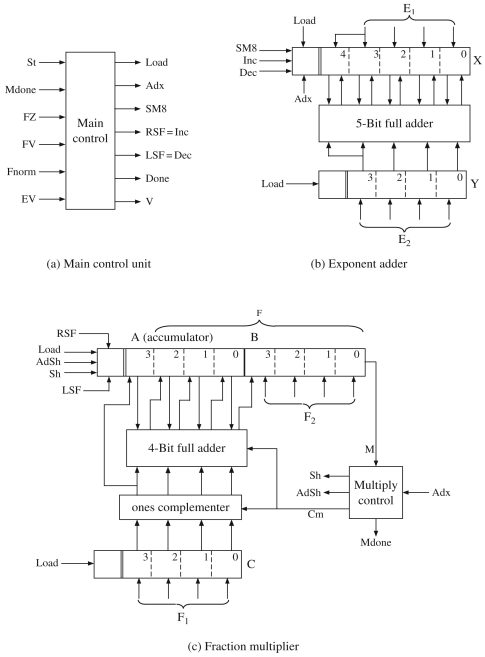
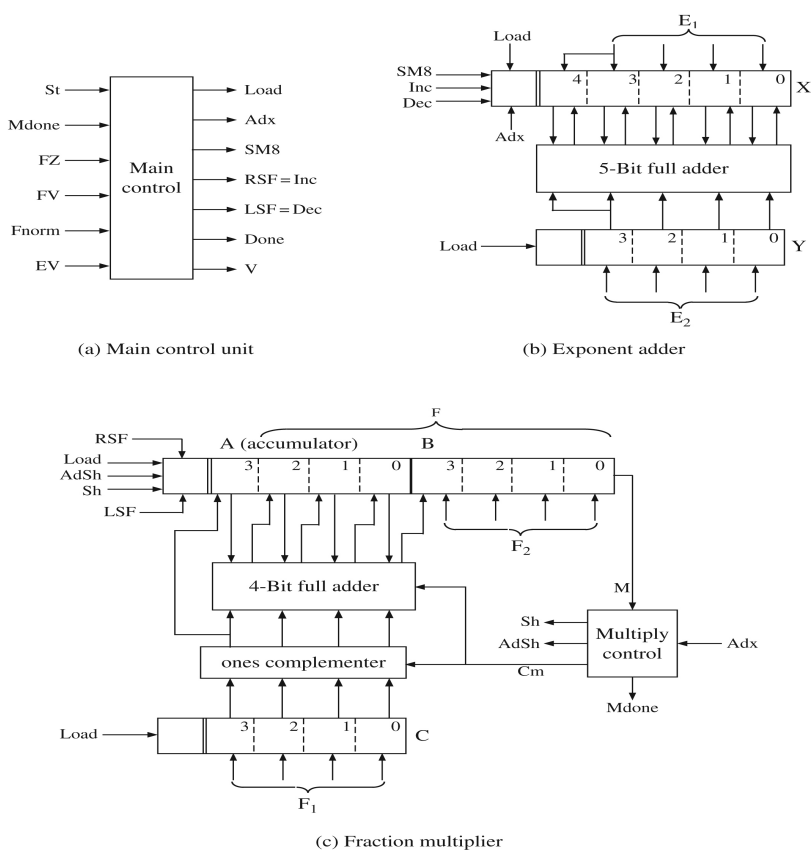


FIGURE 7-7: Major Components of a Floating-Point Multiplier



VHDL Model

```
compout <= not C when Cm = '1' else C;
addout <= A + compout + Cm;
datapath: process(CLK)
begin
  if rising_edge(CLK) then
    PS1 <= NS1;
    State <= Nextstate;
    if Load = '1' then
      X <= E1(3) & E1;
      Y <= E2(3) & E2;
      A <= "0000";
      B <= F2;
      C <= F1;
    end if;
```

VHDL Model

```
if ADX = '1' then
  X <= X + Y;
end if;
if SM8 = '1' then
  X <= "11000";
end if;
```

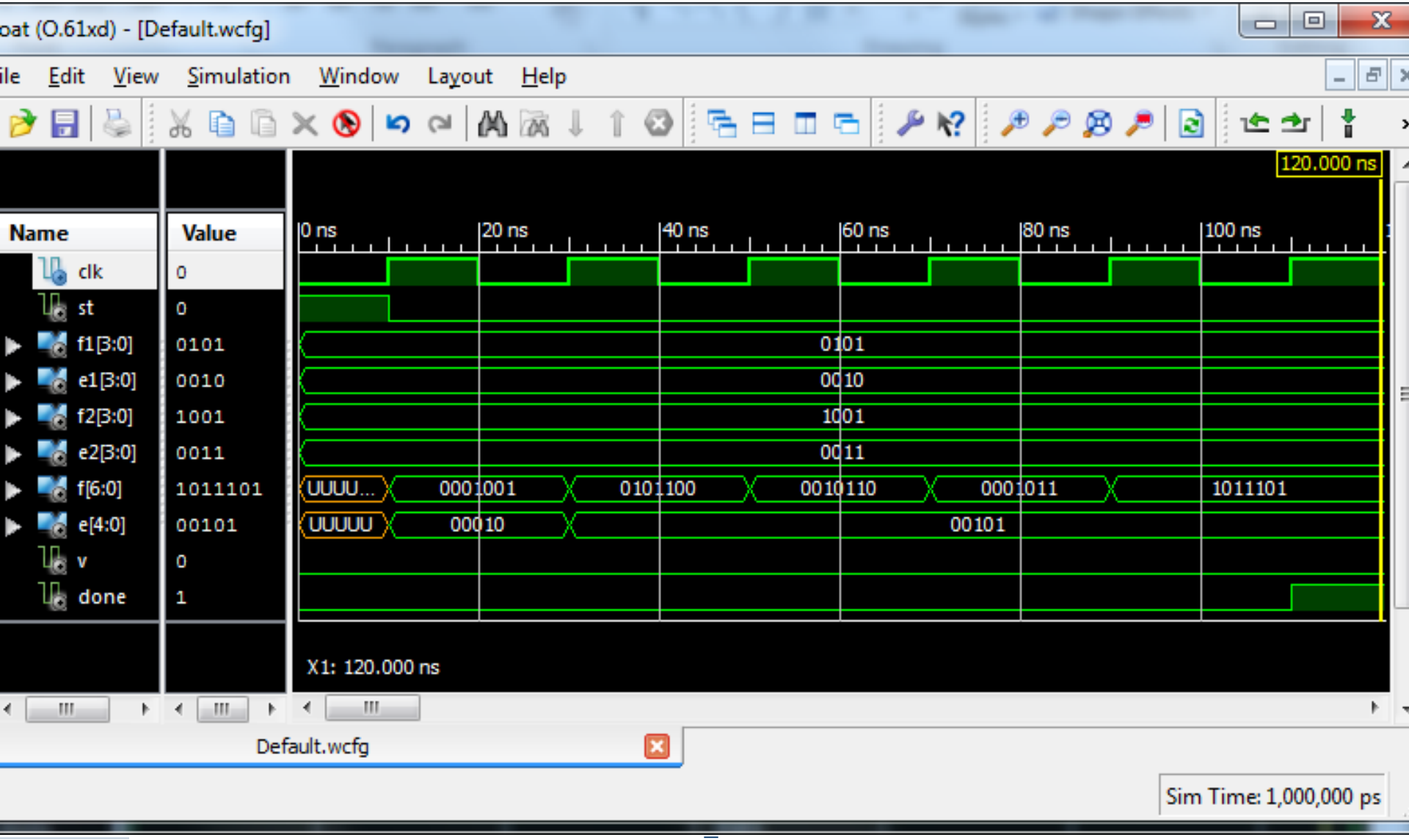
VHDL Model

```
if RSF = '1' then
  A <= '0' & A(3 downto 1);
  B <= A(0) & B(3 downto 1);
  X <= X + 1;
end if;
if LSF = '1' then
  A <= A(2 downto 0) & B(3);
  B <= B(2 downto 0) & '0';
  X <= X - 1;
end if;
```

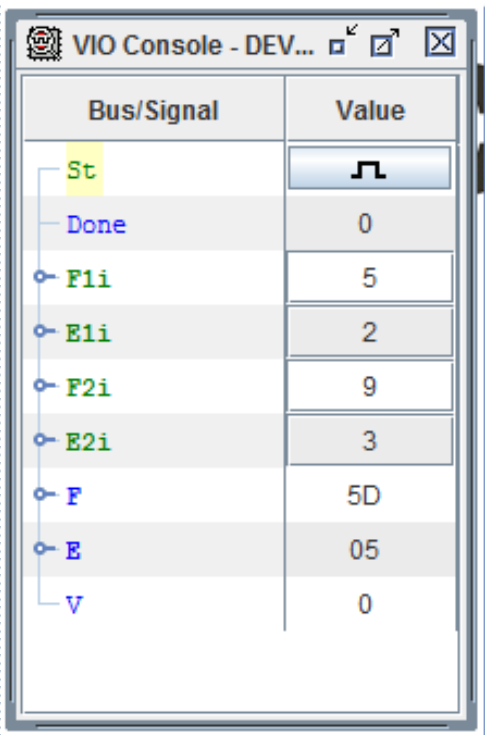
VHDL Model

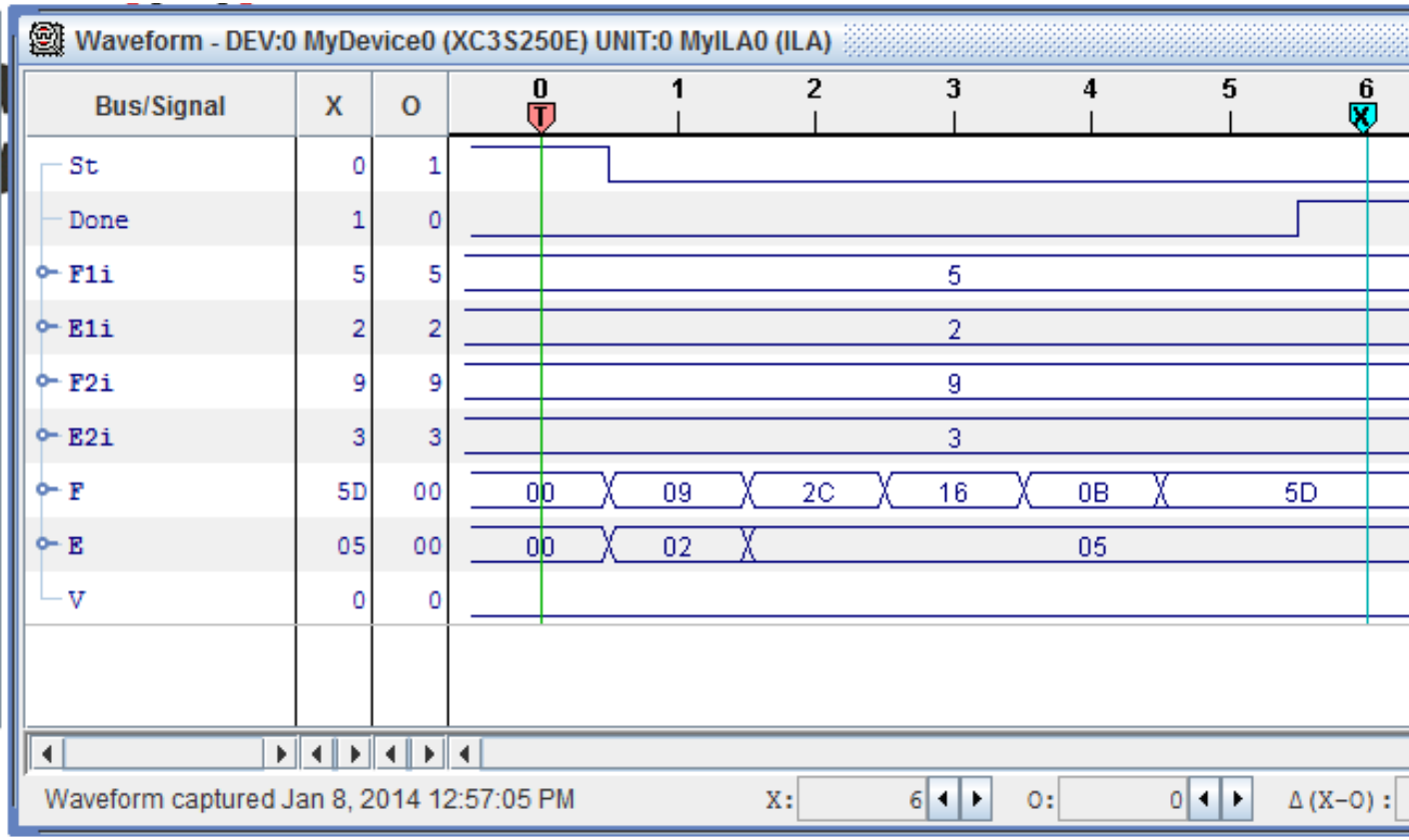
```
      if AdSh = '1' then
        A <= compout(3) & addout(3 downto 1);
        B <= addout(0) & B(3 downto 1);
      end if;
      if Sh = '1' then
        A <= A(3) & A(3 downto 1);
        B <= A(0) & B(3 downto 1);
      end if;
    end if;
  end process datapath;

  F <= A(2 downto 0) & B;
  E <= X;
end BEHAVE;
```

VHDL Simulation



FPGA Implementation





Floating-Point Addition

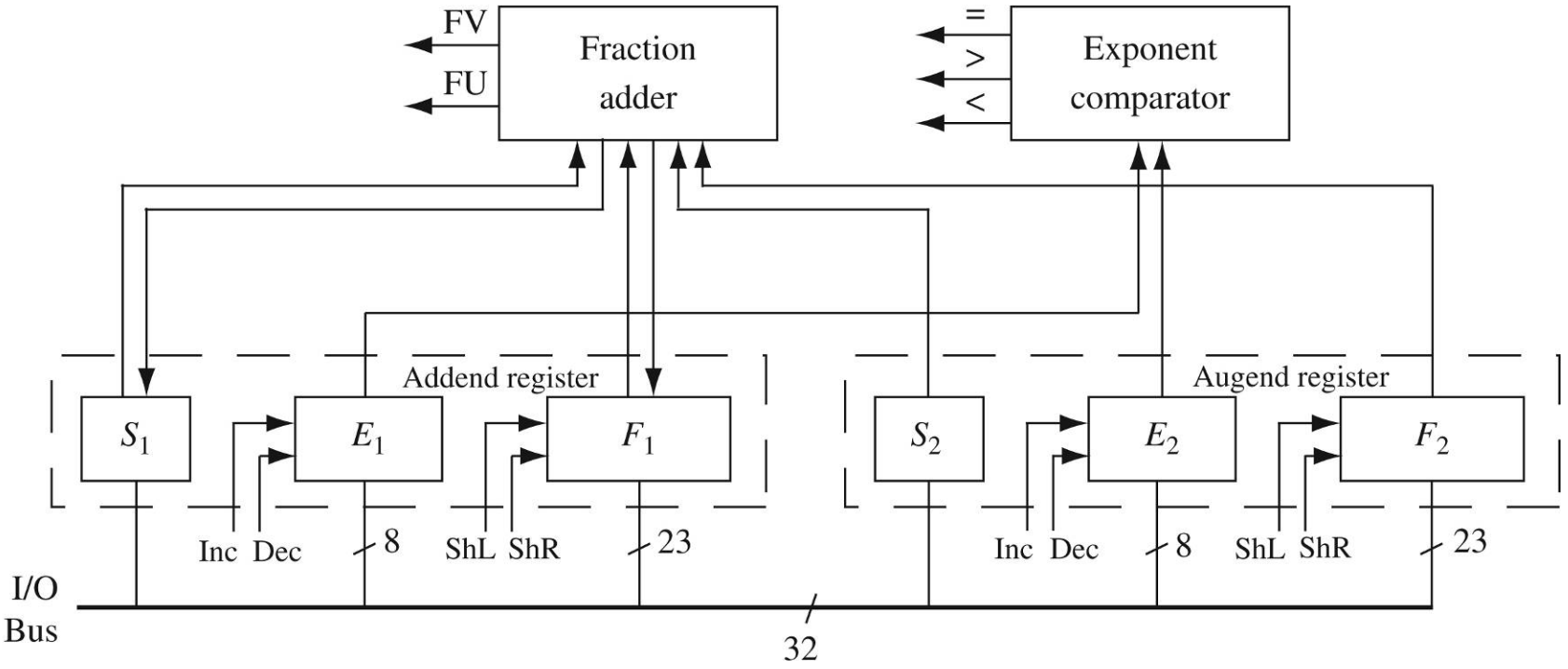
$$(F_1 \times 2^{E_1}) + (F_2 \times 2^{E_2}) = F \times 2^{E}$$

1. Compare exponents. If not equal, shift smaller fraction to right and add 1 to exponent (repeat).
2. Add fractions.
3. If result is 0, adjust for proper 0.
4. If fraction overflow, shift right and add 1.
5. If unnormalized, shift left and subtract 1 (repeat).
6. Check for exponent overflow.
7. Round fraction. If not normalized, go to step 4.

Floating-Point Hardware

- Comparator for exponents (step 1a)
- Shift register (right), incrementer (step 1b)
- ALU (adder) to add fractions (step 2)
- Shifter (right/left), incrementer/decrementer (steps 4,5)
- Overflow detector (step 6)
- Rounding hardware (step 7)

Floating-Point Adder



* IEEE Format

VHDL Model

```
entity FPADD is
  port(CLK, St: in std_logic;
       done, ovf, unf: out std_logic;
       FPinput: in std_logic_vector(31 downto 0);
       FPsum: out std_logic_vector(31 downto 0));
end FPADD;

architecture FPADDER of FPADD is
signal F1, F2: std_logic_vector(25 downto 0);
signal E1, E2: std_logic_vector(7 downto 0);
signal S1, S2, FV, FU: std_logic;
signal F1comp, F2comp: std_logic_vector(27 downto 0);
signal Addout, Fsum: std_logic_vector(27 downto 0);
signal State: integer range 0 to 6;
```

VHDL Model

```
begin
  F1comp <= not("00" & F1) + 1 when S1 = '1' else
            "00" & F1;
  F2comp <= not("00" & F2) + 1 when S2 = '1' else
            "00" & F2;
  Addout <= F1comp + F2comp;
  Fsum <= Addout when Addout(27) = '0' else
          not Addout + 1;
  FV <= Fsum(27) xor Fsum(26);
  FU <= not F1(25);
  FPsum <= S1 & E1 & F1(24 downto 2);
```

VHDL Model

```
process(CLK)
begin
 if rising_edge(CLK) then
  case State is
   when 0 =>
    if St = '1' then
     E1 <= FPinput(30 downto 23); S1 <= FPinput(31);
     F1(24 downto 0) <= FPinput(22 downto 0) & "00";
     if FPinput = 0 then
      F1(25) <= '0';
     else
      F1(25) <= '1';
     end if;
     done <= '0'; ovf <= '0'; unf <= '0'; State <= 1;
    end if;
```

VHDL Model

```
when 1 =>
 E2 <= FPinput(30 downto 23); S2 <= FPinput(31);
 F2(24 downto 0) <= FPinput(22 downto 0) & "00";
 if FPinput = 0 then
  F2(25) <= '0';
 else
  F2(25) <= '1';
 end if;
 State <= 2;
```

VHDL Model

```
when 2 =>
 if F1 = 0 or F2 = 0 then
  State <= 3;
 else
  if E1 = E2 then
   State <= 3;
  elsif E1 < E2 then
   F1 <= '0' & F1(25 downto 1); E1 <= E1 + 1;
  else
   F2 <= '0' & F2(25 downto 1); E2 <= E2 + 1;
  end if;
 end if;
```

VHDL Model

```
when 3 =>
 S1 <= Addout(27);
 if FV = '0' then
  F1 <= Fsum(25 downto 0);
 else
  F1 <= Fsum(26 downto 1); E1 <= E1 + 1;
 end if;
 State <= 4;
when 4 =>
 if F1 = 0 then
  E1 <= "00000000"; State <= 6;
 else
  State <= 5;
 end if;
```

VHDL Model

```
 when 5 =>
  if E1 = 0 then
   unf <= '1'; State <= 6;
  elsif FU = '0' then
   State <= 6;
  else
   F1 <= F1(24 downto 0) & '0'; E1 <= E1 - 1;
  end if;
 when 6 =>
  if E1 = 255 then
   ovf <= '1';
  end if;
  done <= '1'; State <= 0;
end case; end if; end process; end FPADDER;
```

Summary

- **IEEE Floating-Point Formats**
- **Simple Floating-Point Format**
- **Floating-Point Multiplication**
- **Floating-Point Addition**