Exponent Circuit

Parsa Salamatipour

May 2024

Abstract

For computing $\mathbf{a^b}$ in floating point format, there are 3 steps required:

1. Square root ($\sqrt{\mathbf{a}}$)
2. Square ($\mathbf{a}^2$)
3. Multiply powers of 2 ($\mathbf{a}^{2^k}$) and roots of 2 ($\sqrt[2^k]{\mathbf{a}}$) with respect to $\mathbf{b}$ to calculate $\mathbf{a^b}$.

We describe each in detail in each section.

1 Square Root

1.1 Integer Square Root

The following algorithm calculates square root of a binary unsigned integer using *Restoring-Method*

Algorithm 1 INT-SQRT

Input: $\mathbf{a} = a_{n-1}a_{n-2}\dots a_0$
Output: $\mathbf{q} = q_{m-1}\dots q_0 = \sqrt{\mathbf{a}}$
  $n :=$ #bits($\mathbf{a}$)
  $m :=$ #bits($\mathbf{q}$) $= \frac{n}{2}$

  if n is even **then**
    Group $a_{n-1}a_{n-2}$, $a_{n-3}a_{n-4}$, $\dots$, a_1a_0
    as g_{m-1} , g_{m-2} , $\cdots$, g_0
  else
    Group $0a_{n-1}$, $a_{n-2}a_{n-3}$, $\dots$, a_1a_0
    as g_{m-1} , g_{m-2} , $\cdots$, g_0

  for i from $m-1$ to 0 **do**
    if $g_{m-1}\dots g_i \geq q_{m-1}\dots q_{i+1}\mathtt{01}$ **then**
      $g_{m-1}\dots g_i \leftarrow g_{m-1}\dots g_i - q_{m-1}\dots q_{i+1}\mathtt{01}$
      $q_i \leftarrow \mathtt{1}$
    else
      $q_i \leftarrow \mathtt{0}$
  return q

For simplicity of the INT-SQRT circuit we use a **Subtract if Positive** (SUBPOS) component which is implemented using a **subtractor** and a **mux**.

Algorithm 2 Subtract if Positive (SUBPOS)

Input: **a** , **b**
Output: **c**
  if a ≥ b then
    return a − b ▷ SUBTRACT
  else
    return a ▷ RESTORE

The integer square root with 8 bit input and 4 bit output.

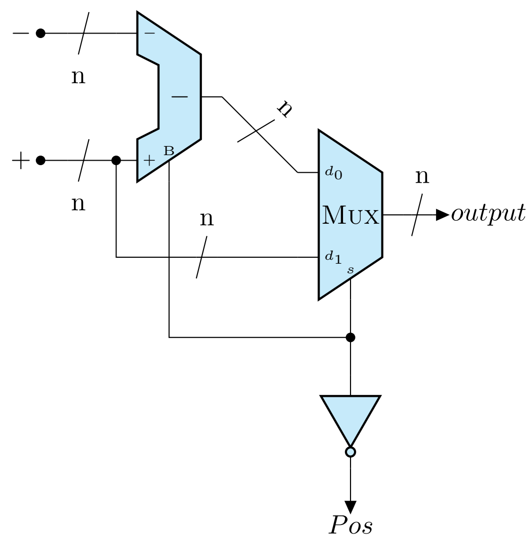

Figure 1: SubPos

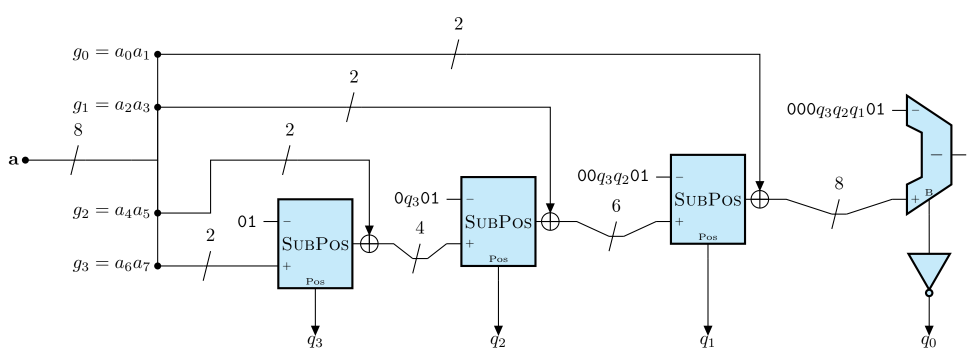

Figure 2: 8 bit Int-Sqrt

1.2 Fraction Square Root

Using the INT-SQRT we can calculate square root of a fraction (`0.110010...`). the trick is that if the number is n bits, we add $\underbrace{0\ldots0}_{n}$ to the right side of the number. This doesn't change the value of the number, but we can use a $2n \times n$ INT-SQRT to calculate square root of the number with n bit accuracy. For example if we want to calculate SQRT(`0.10100101`) we can give `1010010100000000` to a 16×8 INT-SQRT. The result of the INT-SQRT would be `11001101`, so the actual answer is `0.11001101` (the square root of `0.10100101` is actually `0.11001101`).

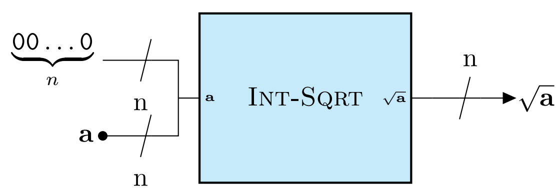

Figure 3: FRACTION-SQRT

From fig. 3 we can simplify the FRACTION-SQRT circuit knowing that the right half bits are all zero:

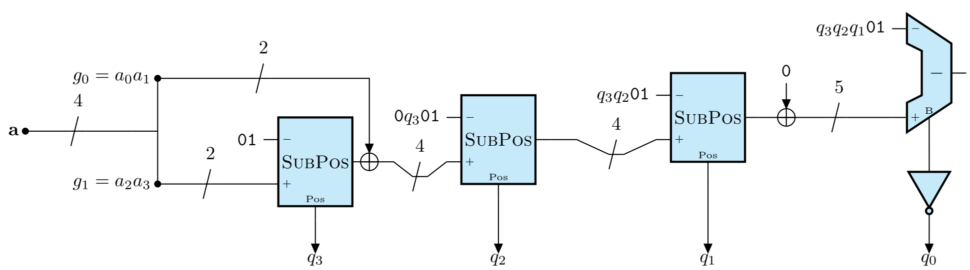

Figure 4: 4 bit FRACTION-SQRT simplified version

1.3 Floating-Point Square Root

Algorithm 3 FLOAT-SQRT

Input: $\mathbf{a} = S : E : F$ ▷ Sign, Exponent, Fraction
Output: $\sqrt{\mathbf{a}} = S' : E' : F'$

```
if S is negative (1) then
    terminate
if F and E are 0 then
    Return 0
S' ← 0
if E is even then
    F' ← FRACTION-SQRT(01 : F)
else
    F' ← FRACTION-SQRT(1 : F : 0)
E' ← E/2                        ▷ E' ← bias+ SAR(E − bias)
F' ← F'[1 : end − 1]            ▷ F' is now f bits
return √a
```

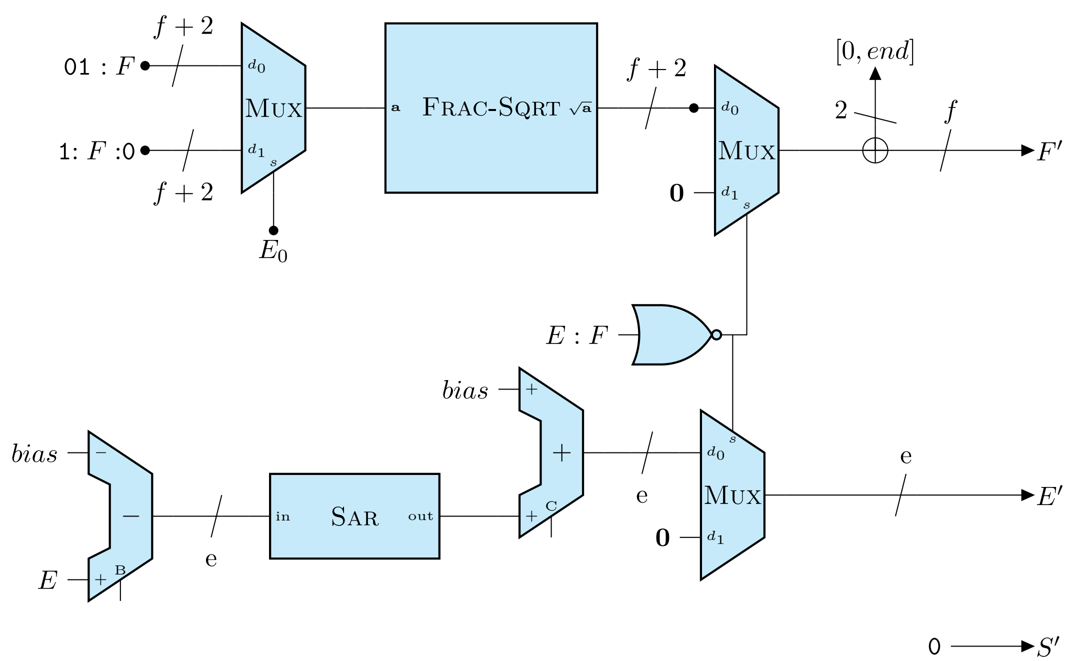

Figure 5: FLOAT-SQRT

2 Square

2.1 Integer Square

For squaring a number we can use a simple array multiplier. But the problem is that the multiplicand and multiplier are the same and some simplifications can be applied. **Every bit has hidden carries in it that would be carried to the next level in the implementation**. The plus (+) sign is addition.

$$
\begin{aligned}
q_0 &= a_0a_0 && = a_0 \\
q_1 &= a_0a_1 + a_1a_0 && = 0 \\
q_2 &= a_0a_2 + a_1a_1 + a_2a_0 && = a_0a_1 + a_1 \\
q_3 &= a_0a_3 + a_1a_2 + a_2a_1 + a_3a_0 && = a_0a_2 \\
q_4 &= a_0a_4 + a_1a_3 + a_2a_2 + a_3a_1 + a_4a_0 && = a_0a_3 + a_1a_2 + a_2 \\
q_5 &= a_0a_5 + a_1a_4 + a_2a_3 + a_3a_2 + a_4a_1 + a_5a_0 && = a_0a_4 + a_1a_3 \\
&\vdots && \vdots
\end{aligned}
$$

For 4 bit squarer it would be like this:

$$
\begin{aligned}
q_0 &= a_0a_0 && = a_0 \\
q_1 &= a_0a_1 + a_1a_0 && = 0 \\
q_2 &= a_0a_2 + a_1a_1 + a_2a_0 && = a_0a_1 + a_1 \\
q_3 &= a_0a_3 + a_1a_2 + a_2a_1 + a_3a_0 && = a_0a_2 \\
q_4 &= a_1a_3 + a_2a_2 + a_3a_1 && = a_0a_3 + a_1a_2 + a_2 \\
q_5 &= a_2a_3 + a_3a_2 && = a_1a_3 \\
q_6 &= a_3a_3 && = a_2a_3 + a_3 \\
q_7 &= 0 && = 0
\end{aligned}
$$

2.2 Float Square

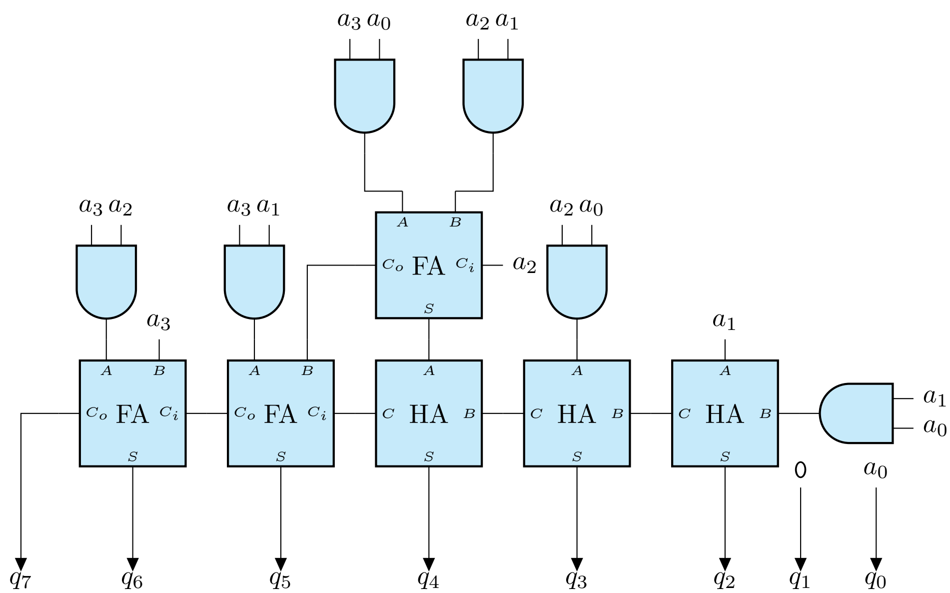

Figure 6: 4 bit Int-Square

3 Main Algorithm

For computing $\mathbf{a}^{\mathbf{b}}$ we first check for E of exponent ($\mathbf{b}_E$). If it is negative, then we take the Float-Sqrt($\mathbf{a}$) and if the corresponding digit in the $\mathbf{b}_F$ with respect to E is `1` then we multiply q by $\mathbf{a}$. If E is posative, we every time compute Float-Sqrt($\mathbf{a_1}$) and Float-Square($\mathbf{a_2}$) and multiply q by $\mathbf{a_1}$ and $\mathbf{a_2}$ if the corrisponding digit in $\mathbf{b}_F$ with respect to E is `1`.

3.1 Registers

$\mathbf{a}$, $\mathbf{b}(S : \mathbf{E} : \mathbf{F})$, $\mathbf{a}'$, $\mathbf{q}$, End, $\mathbf{F_1}$, $\mathbf{F_2}$, i, j, $\mathbf{SC}$, $\mathbf{SC}_{\frac{1}{x}}$, **ReciprocalRegister**, $ShiftCtrl$, $MultiplyCtrl$, $ReciprocalCtrl$

Algorithm 4 EXPONENT

Input: $\mathbf{a} = a_S\mathbf{a}_E\mathbf{a}_F$, $\mathbf{b}{=}S\mathbf{EF}$
Output: $\mathbf{q} = q_S\mathbf{q}_E\mathbf{q}_F = \mathbf{a}^{\mathbf{b}}$

```
e : #bits of Exponent
f : #bits of Fraction
q ← 0 : bias : 0
E = b − bias
if E < 0 then
    while E < 0 do
        a ← FLOAT-SQRT(a)
        E ← E + 1
    q ← FLOAT-MULTIPLY(a)
    for i in digits of F do
        a ← FLOAT-SQRT(a)
        if i is 1 then
            q ← FLOAT-MULTIPLY(a)
else if E ≥ f then
    while E > f do
        a ← FLOAT-SQUARE(a)
        E ← E − 1
    for i in reversed digits of F do
        a ← FLOAT-SQUARE(a)
        if i is 1 then
            q ← FLOAT-MULTIPLY(a)
    a ← FLOAT-SQUARE(a)
    q ← FLOAT-MULTIPLY(q, a)
else
    a′ : square root register
    F′1 : F′2 : real position of Fraction register

    a′ ← a
    F′1 ← 0...01
    F′2 ← b_F
    while E > 0 do
        SHL(F′1 : F′2)
        E ← E − 1
    for i from f to 1 do
        SHR(F′1 : F′1)
        SHL(F′2 : F′2)
        a ← FLOAT-SQUARE(a)
        a′ ← FLOAT-SQRT(a′)
        if F′1 is 1 then
            q ← FLOAT-MULTIPLY(q, a)
        if F′2 is 1 then
            q ← FLOAT-MULTIPLY(q, a′)
return q
```

Algorithm 5 EXPONENT-RTL

$a_{LD}|\mathbf{a} \leftarrow \mathbf{a_{in}}$
$b_{LD}|\mathbf{b} \leftarrow \mathbf{b_{in}}$
$a_{LD}|\mathbf{a'} \leftarrow \mathbf{a_{in}}$
$Start|End \leftarrow \mathtt{0}$
$Start|\mathbf{q} \leftarrow 0 : bias : \mathbf{0}$
$Start|E_{<0} : \mathbf{E} \leftarrow \mathbf{E} - bias$
$Start|E_{>f} \leftarrow \mathbf{E} - bias > f$
$Start|\mathbf{F_1} \leftarrow \mathtt{0}\ldots\mathtt{01}$
$Start|\mathbf{F_2} \leftarrow \mathbf{F}$
$Start|i \leftarrow \mathtt{0}$
$Start|j \leftarrow \mathtt{0}$
$Start|ShiftCtrl \leftarrow \mathtt{1}$
$Start|MultiplyCtrl \leftarrow \mathtt{0}$
$Start|ReciprocalCtrl \leftarrow \mathtt{0}$
$ShiftCtrl \cdot E_{<0}|\mathbf{a} \leftarrow \textsc{Float-Sqrt}(\mathbf{a})$
$ShiftCtrl \cdot E_{>f}|\mathbf{a} \leftarrow \textsc{Float-Square}(\mathbf{a})$
$ShiftCtrl \cdot \overline{E_{>f}} \cdot \overline{E_{<0}}|\textsc{Shl}(\mathbf{F_1} : \mathbf{F_2})$
$ShiftCtrl \cdot \overline{E_{>f}}|\mathbf{E}\text{--}$
$ShiftCtrl \cdot E_{>f}|\mathbf{E}\text{++}$
$ShiftCtrl \cdot E_{=0}|ShiftCtrl \leftarrow \mathtt{0}$
$ShiftCtrl \cdot E_{=0}|MultiplyCtrl \leftarrow \mathtt{1}$
$ShiftCtrl \cdot E_{=0}|\mathbf{SC} \leftarrow f$
$MultiplyCtrl \cdot \overline{E_{>f}} \cdot \overline{E_{<0}}|i \leftarrow \textsc{Shr}(\mathbf{F_1})$
$MultiplyCtrl \cdot \overline{E_{>f}} \cdot \overline{E_{<0}}|j \leftarrow \textsc{Shl}(\mathbf{F_2})$
$MultiplyCtrl \cdot \overline{E_{>f}} \cdot \overline{E_{<0}}|\mathbf{a} \leftarrow \textsc{Float-Square}(\mathbf{a})$
$MultiplyCtrl \cdot \overline{E_{>f}} \cdot \overline{E_{<0}}|\mathbf{a'} \leftarrow \textsc{Float-Sqrt}(\mathbf{a'})$
$MultiplyCtrl \cdot E_{<0}|i \leftarrow \textsc{Shr}(\mathbf{F})$
$MultiplyCtrl \cdot E_{<0}|j \leftarrow \mathtt{0}$
$MultiplyCtrl \cdot E_{<0}|\mathbf{a} \leftarrow \textsc{Float-Sqrt}(\mathbf{a})$
$MultiplyCtrl E_{>f}|i \leftarrow \textsc{Shl}(\mathbf{F})$
$MultiplyCtrl E_{>f}|j \leftarrow \mathtt{0}$
$MultiplyCtrl E_{>f}|\mathbf{a} \leftarrow \textsc{Float-Square}(\mathbf{a})$
$MultiplyCtrl|\mathbf{SC}\text{--}$
$MultiplyCtrl \cdot i|\mathbf{q} \leftarrow \textsc{Float-Mult}(\mathbf{q}, \mathbf{a})$
$MultiplyCtrl \cdot j|\mathbf{q} \leftarrow \textsc{Float-Mult}(\mathbf{q}, \mathbf{a'})$
$MultiplyCtrl \cdot SC_{=0}|MultiplyCtrl \leftarrow \mathtt{0}$
$MultiplyCtrl \cdot SC_{=0} \cdot S|ReciprocalCtrl \leftarrow \mathtt{1}$
$MultiplyCtrl \cdot SC_{=0} \cdot S|\mathbf{SC}_{\frac{1}{x}} \leftarrow \#(\frac{1}{x}\ iterations)$
$MultiplyCtrl \cdot SC_{=0} \cdot \overline{S}|End \leftarrow \mathtt{1}$
$ReciprocalCtrl|ReciprocalRegister_{LD} \leftarrow \mathtt{1}$
$ReciprocalCtrl|\mathbf{SC}_{\frac{1}{x}}\text{--}$
$ReciprocalCtrl \cdot SC_{\frac{1}{x}=0}|ReciprocalCtrl \leftarrow \mathtt{0}$
$ReciprocalCtrl \cdot SC_{\frac{1}{x}=0}|ReciprocalRegister_{LD} \leftarrow \mathtt{0}$
$ReciprocalCtrl \cdot SC_{\frac{1}{x}=0}|\mathbf{q} \leftarrow \mathbf{ReciprocalRegister}$
$ReciprocalCtrl \cdot SC_{\frac{1}{x}=0}|End \leftarrow \mathtt{1}$

For designing this circuit, we break it into 2 parts: Controller and Arithmetic

3.2 Arithmetic

The following circuits are all for the Arithmetic part and is concluded from the EXPONENT-RTL (Algorithm 5). All the MUXs are controlled with decoded selects:

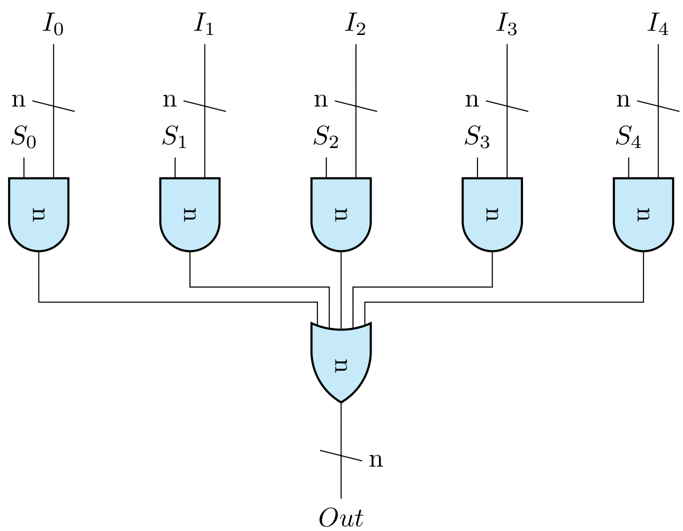

Figure 7: MUX with decoded select (MUX)

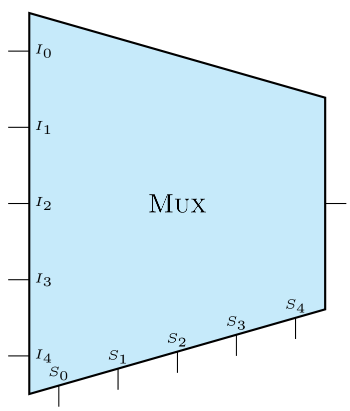

Figure 8: That excact same (MUX)

Arithmetic circuit:

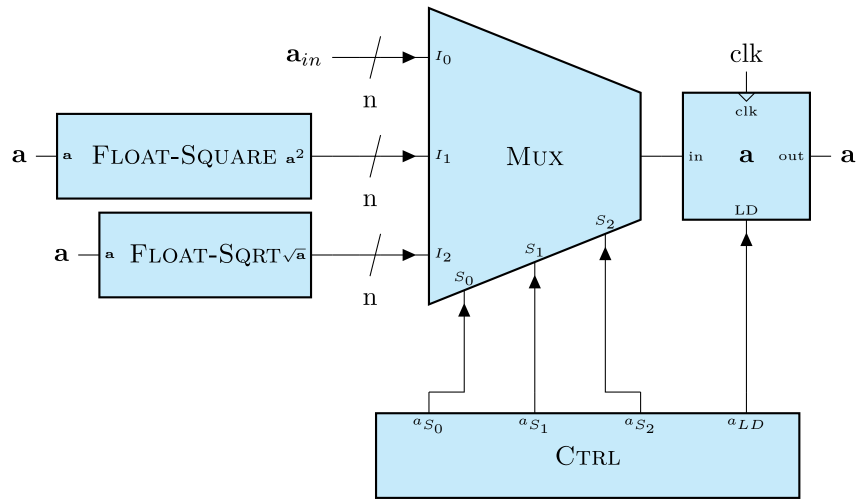

Figure 9: Arithmetic of $\mathbf{a}$ (Connected to CTRL))

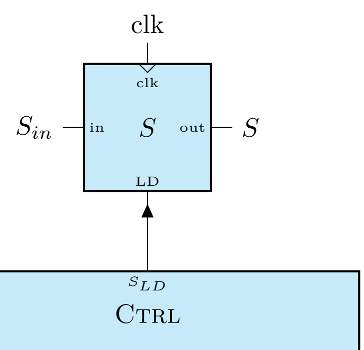

Figure 10: Arithmetic of S (Connected to CTRL))

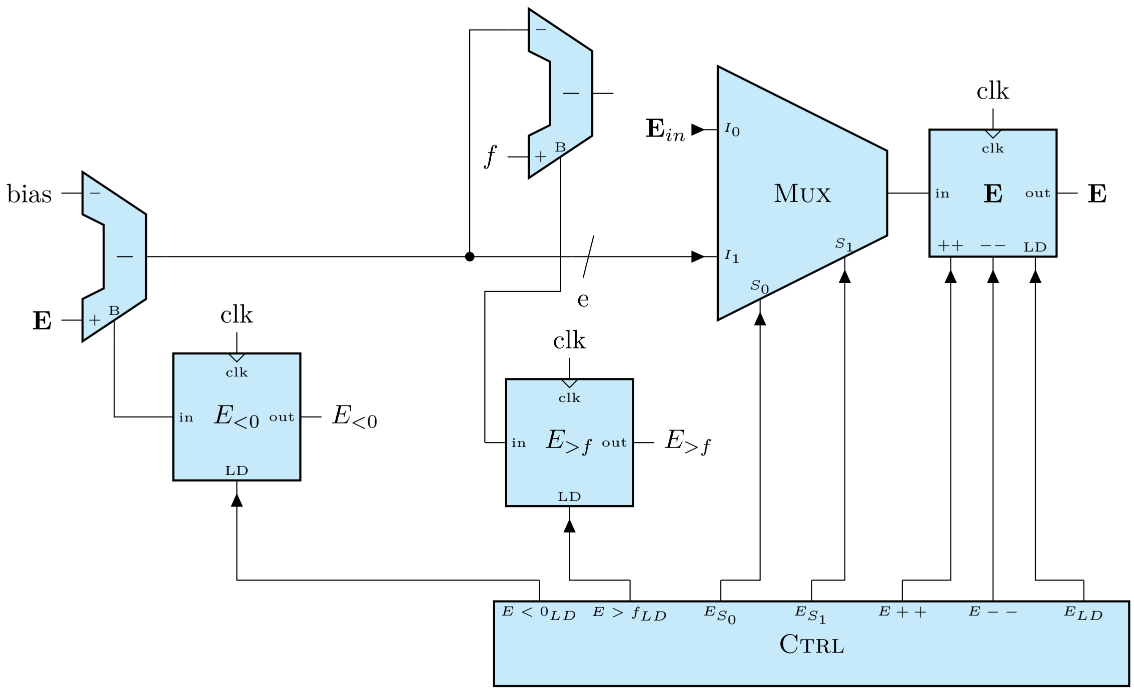

Figure 11: Arithmetic of $\mathbf{E}$, $E_{<0}$ and $E_{>f}$ (Connected to Ctrl))

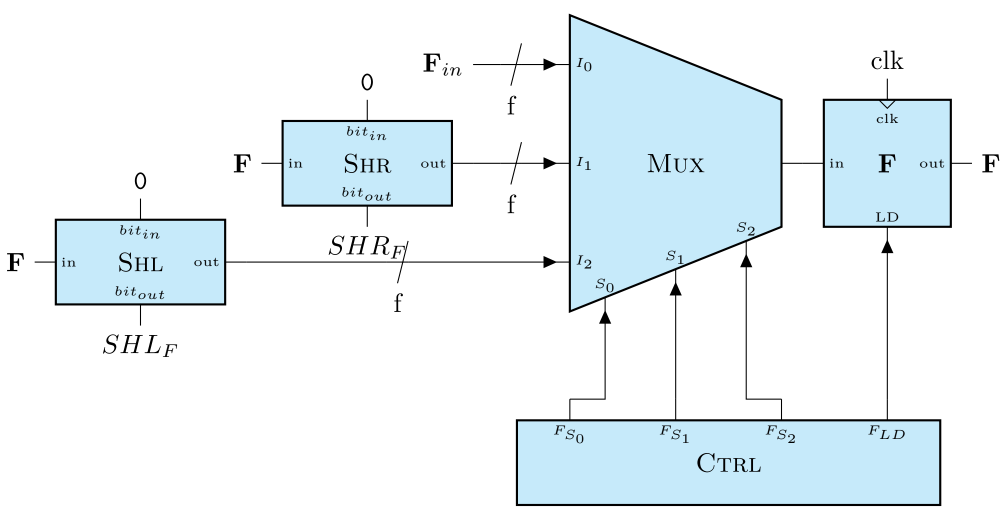

Figure 12: Arithmetic of $\mathbf{F}$ (Connected to Ctrl))

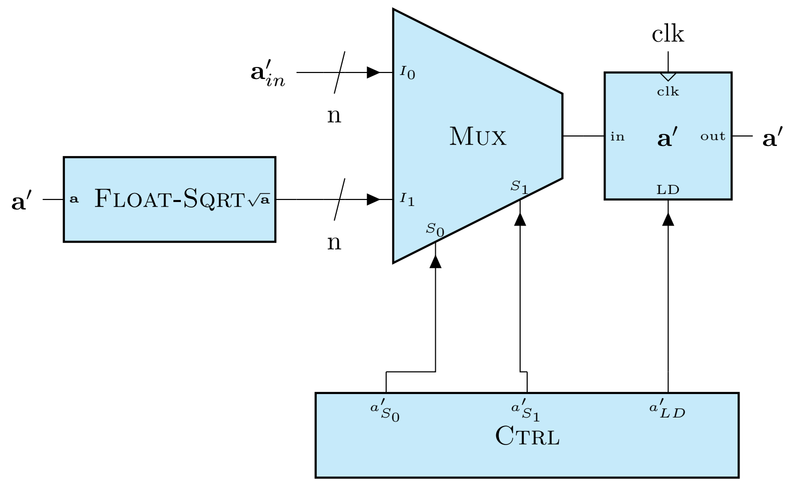

Figure 13: Arithmetic of $\mathbf{a}'$ (Connected to Ctrl))

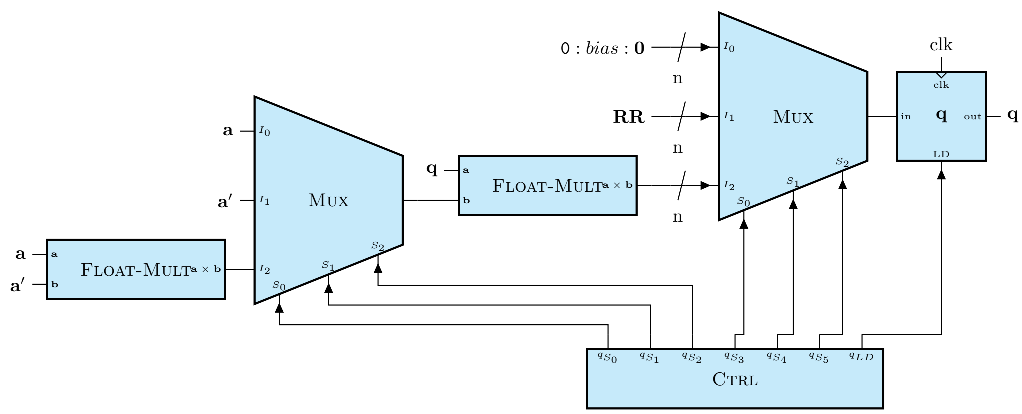

Figure 14: Arithmetic of $\mathbf{q}$ (Connected to Ctrl))

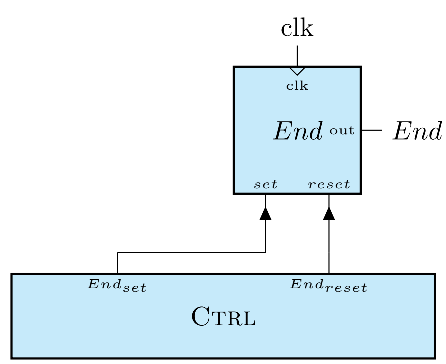

Figure 15: Arithmetic of *End* (Connected to Ctrl))

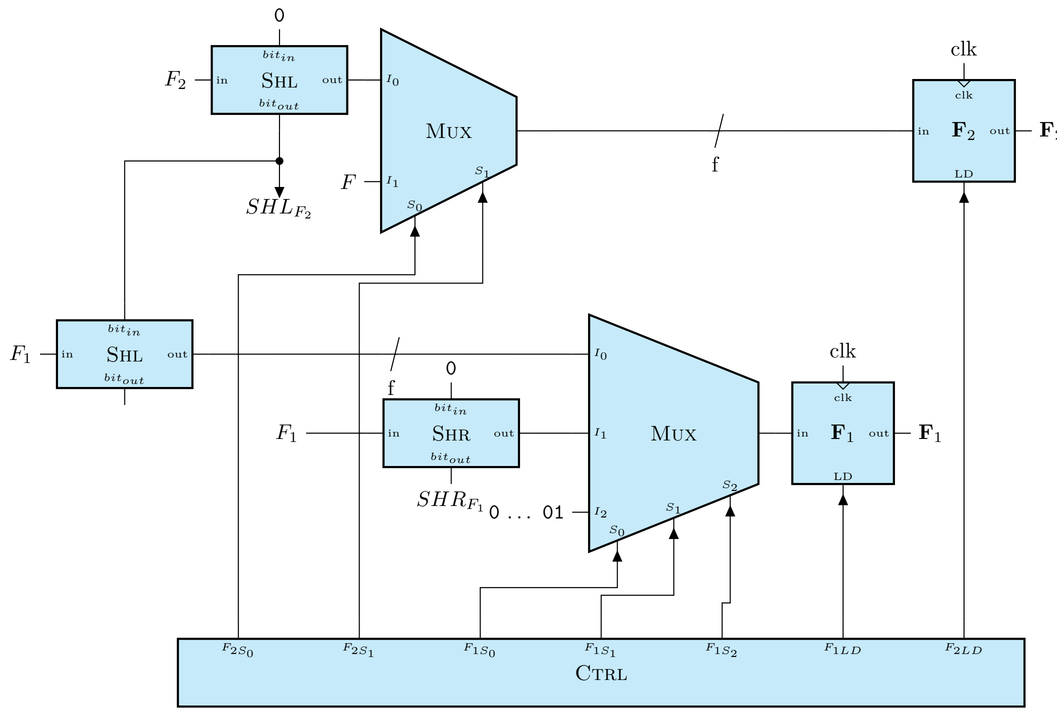

Figure 16: Arithmetic of F_1 and F_2 (Connected to CTRL))

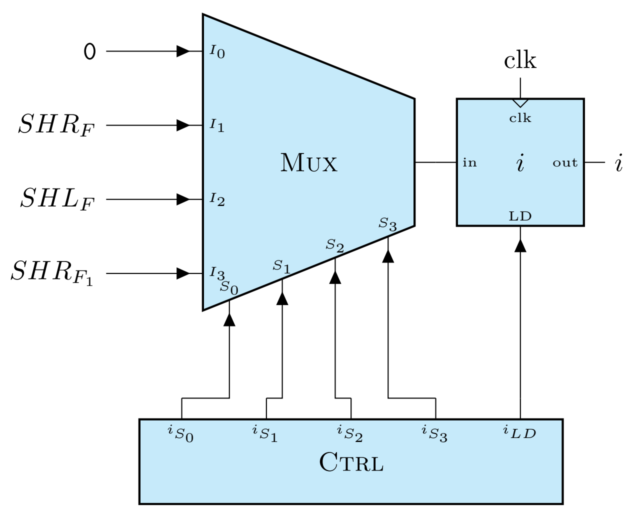

Figure 17: Arithmetic of i (Connected to CTRL))

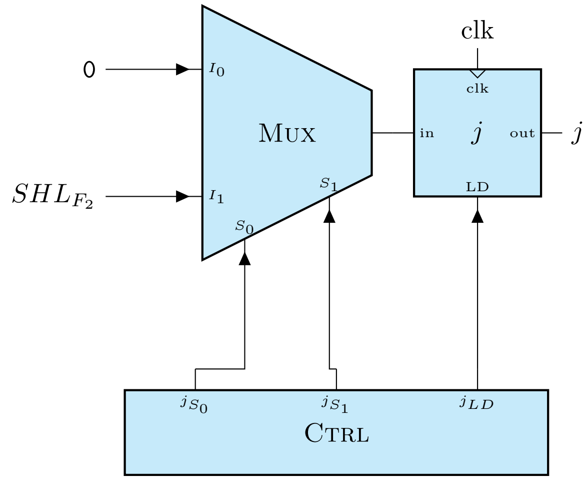

Figure 18: Arithmetic of j (Connected to CTRL))

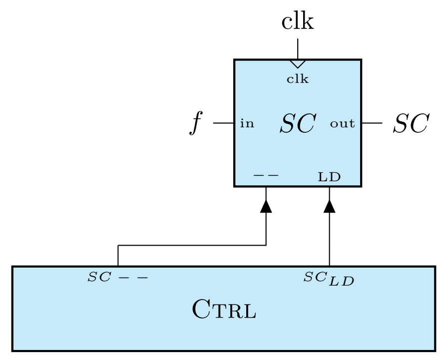

Figure 19: Arithmetic of SC (Connected to CTRL))

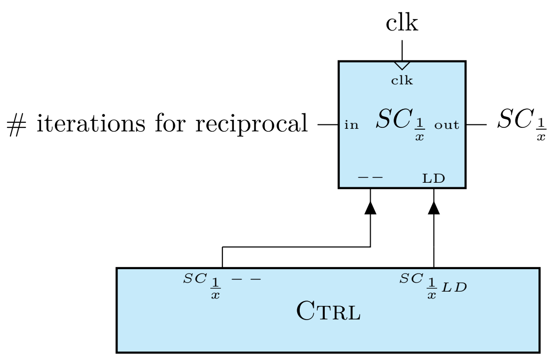

Figure 20: Arithmetic of $SC_{\frac{1}{x}}$ (Connected to CTRL))

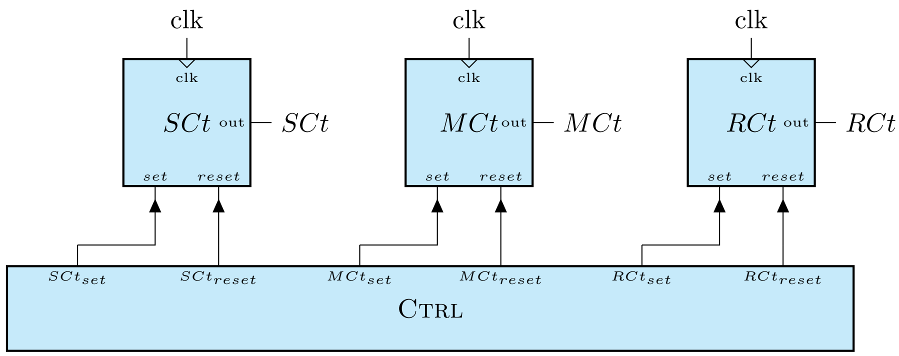

Figure 21: Arithmetic of *ShiftController*, *MultiplyController* and *ReciprocalController* (Connected to CTRL))

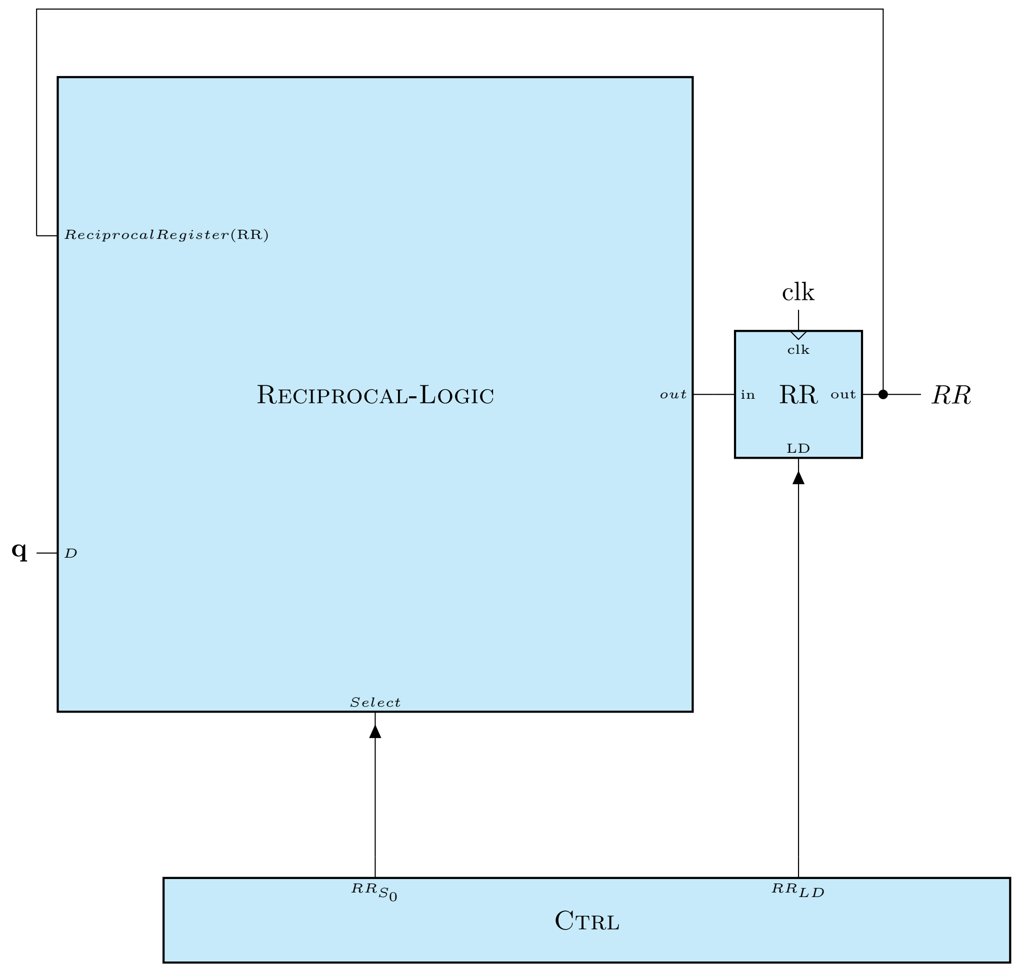

Figure 22: Reciprocal [1]

3.3 Controller

The following logics are for the Ctrl component we saw earlier:

$$
\begin{aligned}
a_{LD} &= a_{LDin} + SCt \cdot (E_{<0} + E_{>f}) + MCt \\
a_{S_0} &= a_{LDin} \\
a_{S_1} &= SCt \cdot E_{>f} + MCt \cdot (E_{>f} + \overline{E_{<0}} \cdot \overline{E_{>f}}) \\
a_{S_2} &= E_{<0} \cdot (SCt + MCt) \\
S_{LD} &= b_{LDin} \\
E_{LD} &= b_{LDin} + Start \\
E-- &= \overline{E_{>f}} \\
E++ &= E_{>f} \\
E_{S_0} &= b_{LDin} \\
E_{S_1} &= Start \\
E_{>fLD} &= Start \\
E_{<0LD} &= Start \\
F_{LD} &= b_{LDin} + MCt \cdot (E_{<0} + E_{>f}) \\
F_{S_0} &= b_{LDin} \\
F_{S_1} &= MCt \cdot E_{<0} \\
F_{S_2} &= MCt \cdot E_{>f} \\
a'_{DL} &= a_{LDin} + MCt \cdot \overline{E_{<0}} \cdot \overline{E_{>f}} \\
a'_{S_0} &= a_{LDin} \\
a'_{S_0} &= MCt \cdot \overline{E_{<0}} \cdot \overline{E_{>f}} \\
q_{LD} &= Start + MCt \cdot (i + j) + RCt \cdot NOR(SC_{\frac{1}{x}}) \\
q_{S_0} &= i \cdot j \\
q_{S_1} &= i \cdot \overline{j} \\
q_{S_2} &= \overline{i} \cdot j \\
q_{S_3} &= MCt \cdot (i + j) \\
q_{S_4} &= RCt \cdot NOR(SC_{\frac{1}{x}}) \\
q_{S_5} &= Start \\
End_{set} &= RCt \cdot NOR(SC_{\frac{1}{x}}) + MCt \cdot NOR(SC) \cdot \overline{S} \\
End_{reset} &= Start
\end{aligned}
$$

$$
\begin{aligned}
F_{1LD} &= Start + \overline{E_{<0}} \cdot \overline{E_{>f}} \cdot (SCt + MCt) \\
F_{2LD} &= Start + \overline{E_{<0}} \cdot \overline{E_{>f}} \cdot (SCt + MCt) \\
F_{1S_0} &= SCr \cdot \overline{E_{<0}} \cdot \overline{E_{>f}} \\
F_{1S_1} &= MCt \cdot \overline{E_{<0}} \cdot \overline{E_{>f}} \\
F_{1S_2} &= Start \\
F_{2S_0} &= \overline{E_{<0}} \cdot \overline{E_{>f}} \cdot (SCt + MCt) \\
F_{2S_1} &= Start \\
i_{LD} &= Start + MCt \\
i_{S_0} &= Start \\
i_{S_1} &= MCt \cdot E_{<0} \\
i_{S_2} &= MCt \cdot E_{>f} \\
i_{S_3} &= MCt \cdot (\overline{E_{<0}} \cdot \overline{E_{>f}}) \\
j_{LD} &= Start + MCt \\
j_{S_0} &= Start + MCt \cdot (E_{<0} + E_{>f}) \\
j_{S_1} &= MCt \cdot \overline{E_{<0}} \cdot \overline{E_{>f}} \\
SC_{LD} &= SCt \cdot NOR(E) \\
SC-- &= MCt \\
SC_{\frac{1}{x}LD} &= MCt \cdot NOR(SC) \cdot \overline{S} \\
SC_{\frac{1}{x}}-- &= RCt \\
SCt_{set} &= Start \\
SCt_{reset} &= SCt \cdot NOR(E) \\
MCt_{set} &= SCt \cdot NOR(E) \\
MCt_{reset} &= MCt \cdot NOR(SC) \cdot \overline{S} \\
RCt_{set} &= MCt \cdot NOR(SC) \cdot \overline{S} \\
RCt_{reset} &= RCt \cdot NOR(SC_{\frac{1}{x}}) \\
RR_{LD} &= RCt \cdot OR(SC_{\frac{1}{x}}) \\
RR_{S_0} &= (SC_{\frac{1}{x}} \overset{?}{\neq} \#\text{reciprocal iterations})
\end{aligned}
$$

References

[1] A. Habegger, A. Stahel, J. Goette, and M. Jacomet, "An efficient hardware implementation for a reciprocal unit," in *2010 Fifth IEEE International Symposium on Electronic Design, Test & Applications*, pp. 183–187, 2010.