On the Improved Implementations and Performance Evaluation of Digit-by-Digit Integer Restoring and Non-Restoring Cube Root Algorithms

Yamin Li

Dept. of Computer Science
Hosei University

http://cis.k.hosei.ac.jp/~yamin/

Cube Root Algorithms and Implementations

- Algorithms
  - Digit-by-digit integer restoring
  - Digit-by-digit integer non-restoring
- Improved implementations
  - Basic implementations
  - Eliminating multiplications
  - Calculating $12q^2$ in advance
  - Using carry save adders (CSAs)
- Cost / performance evaluation

Cube Root

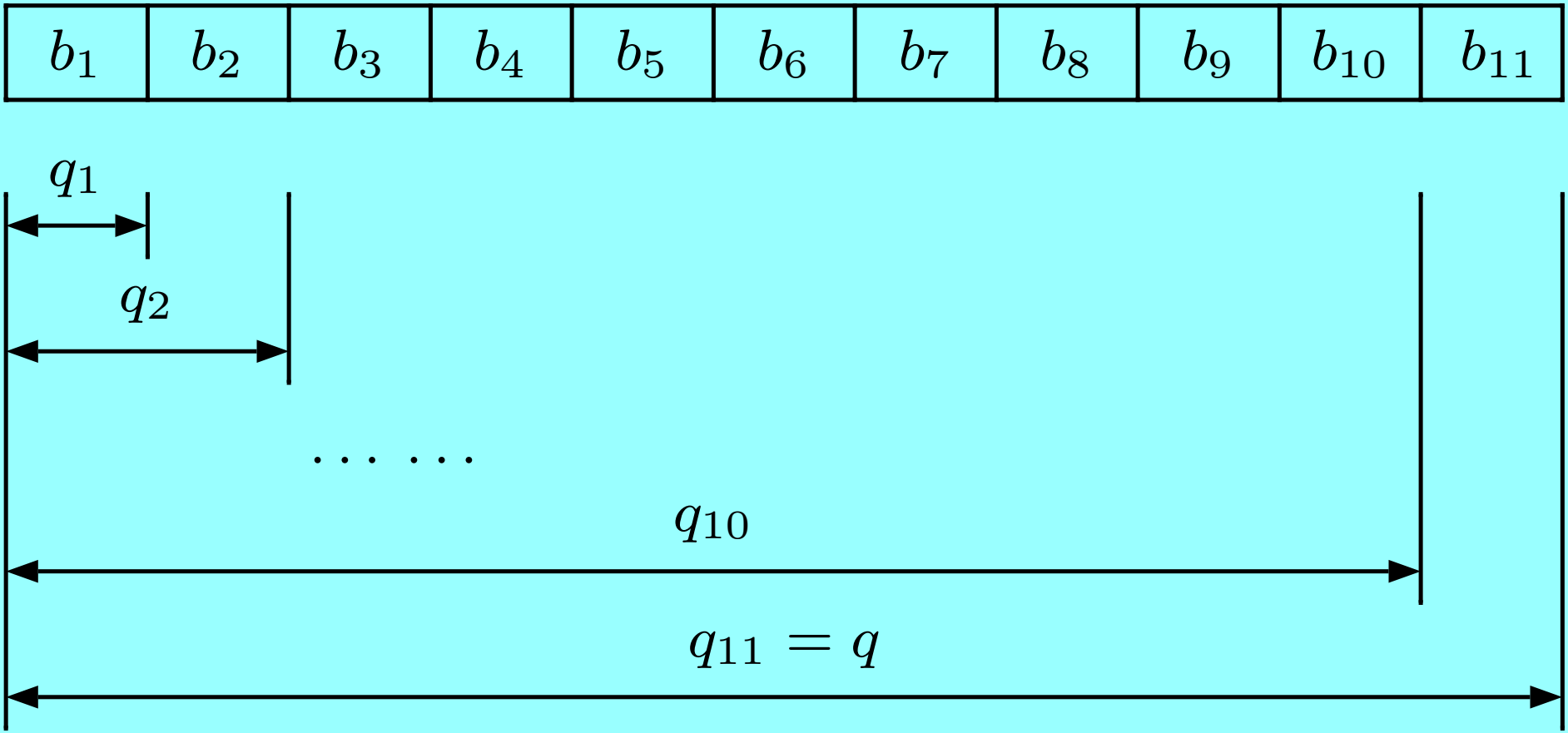
- Given a radicand d, calculate the cube root q so that

$$d = q^3 + r$$

  - Ex. $26 = 2^3 + 18$ (We consider only $d \geq 0$)

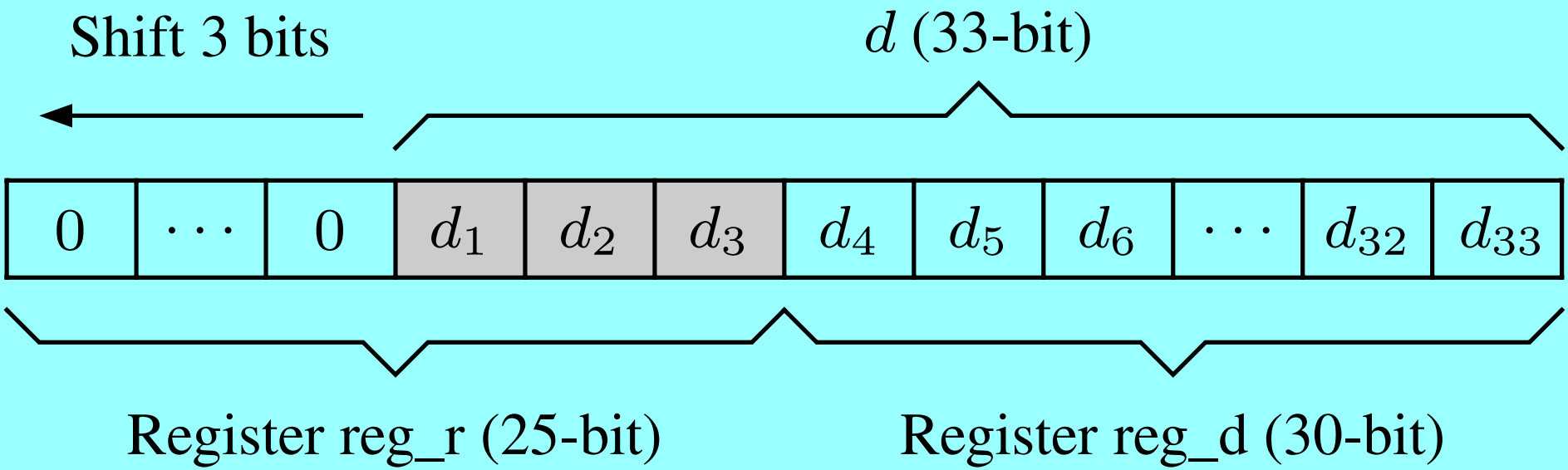
- Assume that d has 33 bits. Then
  - q has 11 bits (33/3 bits)
  - r has 24 bits at most
    - $d = q^3 + r < (q+1)^3 = q^3 + 3q^2 + 3q + 1$
    - $\Rightarrow r \leq 3q^2 + 3q \Rightarrow 2 + 11 \times 2$ bits
    - In order to check the sign of the partial remainder, we use 25 bits during the calculation

Definition of Partial Cube Root q_i



Register reg_q stores cube root

Initial Values of Registers reg_r and reg_d



Register reg_r stores remainder

Register reg_d stores radicand

Basic Restoring Cube Root Algorithm

- Radicand $d = \{d_1, d_2, d_3, \ldots, d_{33}\} = q^3 + r$
- Suppose that we have got a partial cube root q_i and a partial remainder r_i such that the partial radicand

$$\{d_1, d_2, \ldots, d_{3i}\} = q_i^3 + r_i$$

- To calculate b_{i+1} and r_{i+1}, we have

$$\{d_1, d_2, \ldots, d_{3i}, d_{3i+1}, d_{3i+2}, d_{3i+3}\} = q_{i+1}^3 + r_{i+1}$$

where $q_{i+1} = \{q_i, b_{i+1}\}$ or $q_{i+1} = 2q_i + b_{i+1}$

Basic Restoring Cube Root Algorithm

- Let $p_i = \{d_{3i+1}, d_{3i+2}, d_{3i+3}\}$, then

$$\begin{aligned} r_{i+1} &= \{d_1, d_2, \ldots, d_{3i}, p_i\} - q_{i+1}^3 \\ &= 8(\{d_1, d_2, \ldots, d_{3i}\}) + p_i - q_{i+1}^3 \\ &= 8(q_i^3 + r_i) + p_i - (2q_i + b_{i+1})^3 \\ &= 8r_i + p_i - (12q_i^2 b_{i+1} + 6q_i b_{i+1}^2 + b_{i+1}^3) \\ &= \{r_i, p_i\} - (12q_i^2 b_{i+1} + 6q_i b_{i+1}^2 + b_{i+1}^3) \end{aligned}$$

- Assume $b_{i+1} = 1$

$$r_{i+1} = \{r_i, p_i\} - (12q_i^2 + 6q_i + 1)$$

Basic Restoring Cube Root Algorithm

- $r_{i+1} = \{r_i, p_i\} - (12q_i^2 + 6q_i + 1)$
- If $r_{i+1} \geq 0$, then $b_{i+1} = 1$,

$$q_{i+1} = \{q_i, 1\}$$

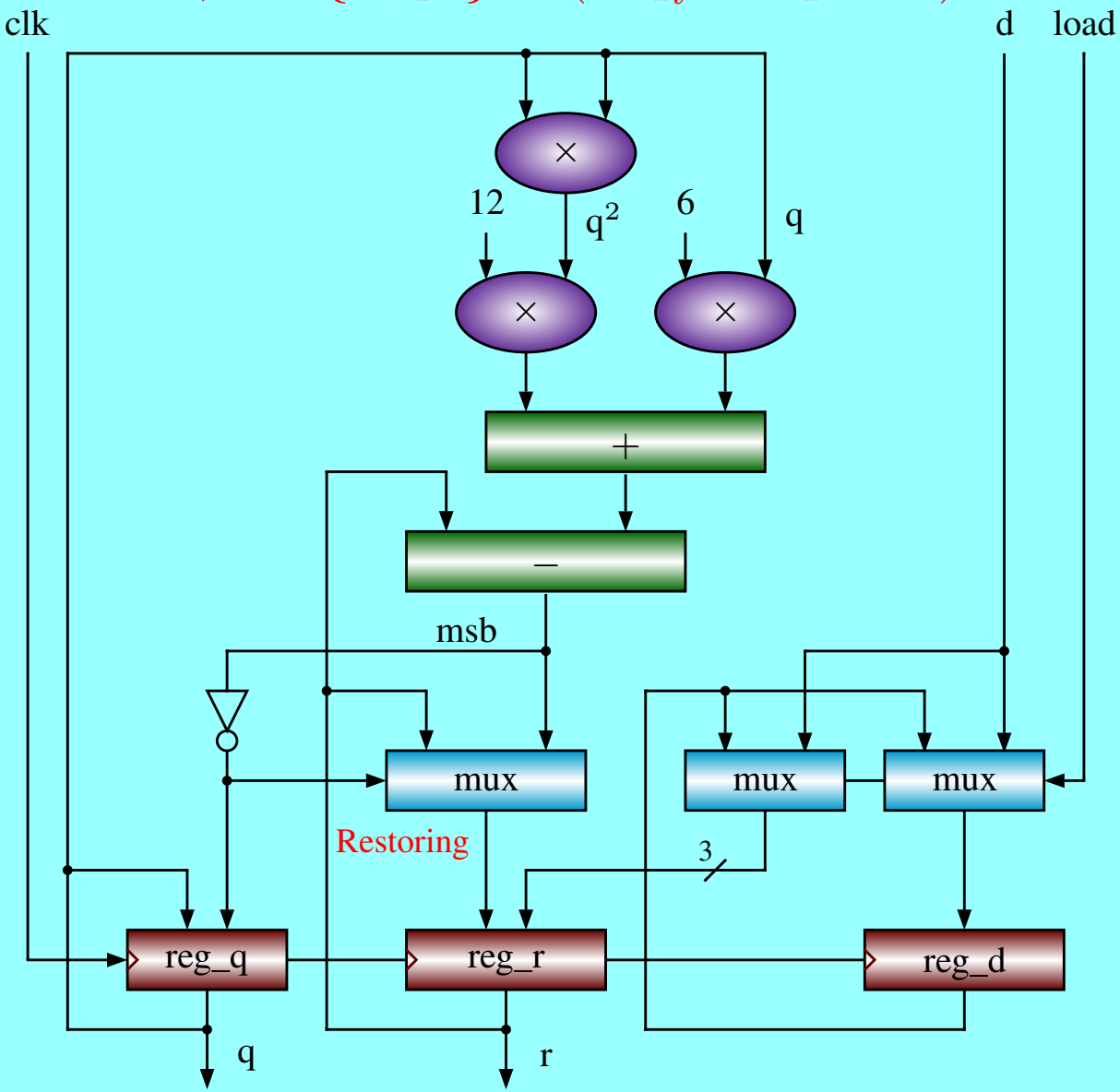
- Otherwise, $b_{i+1} = 0$,

$$q_{i+1} = \{q_i, 0\}$$

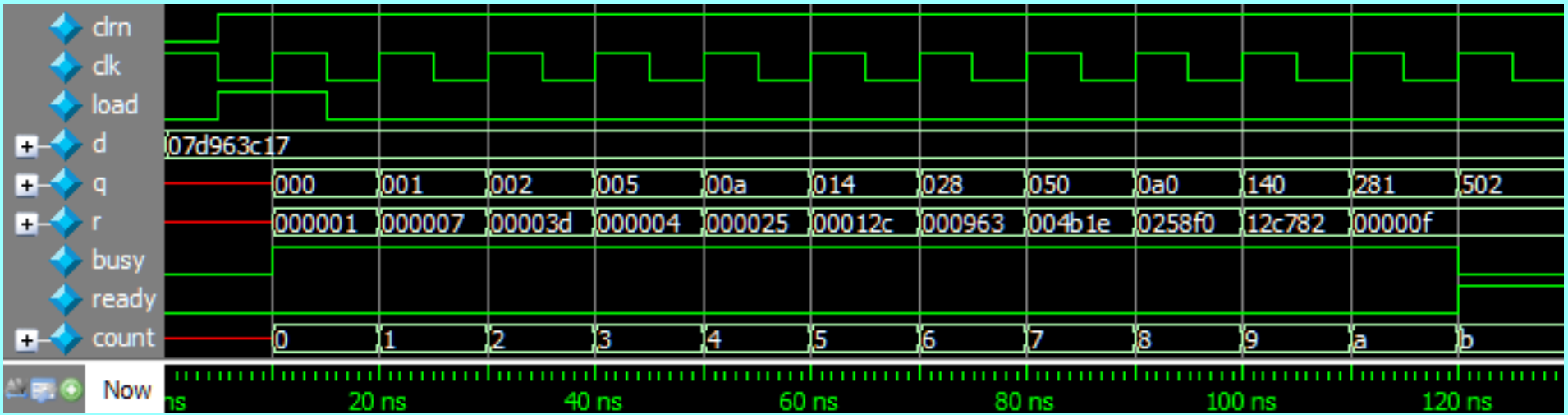
and we restore r_{i+1} by adding $(12q_i^2 + 6q_i + 1)$ to the negative remainder

Circuit of Restoring Cube Root (Using mul)

$$r_{i+1} = \{r_i, p_i\} - (12q_i^2 + 6q_i + 1)$$



Waveform of Restoring Cube Root Algorithm



- Implemented on Altera Cyclone IV E EP4CE115F29C7
- 160 LEs (logic elements)
- 72 DFFs (D flip-plops)
- Two embedded multiplier 9-bit elements
- Maximum clock frequency: 73.87 MHz

Eliminating Multiplications

- Let $s_i = q_i^2$
- For the case of $b_{i+1} = 0$

$$s_{i+1} = q_{i+1}^2 = (2q_i + 0)^2 = 4q_i^2 = 4s_i$$

- For the case of $b_{i+1} = 1$

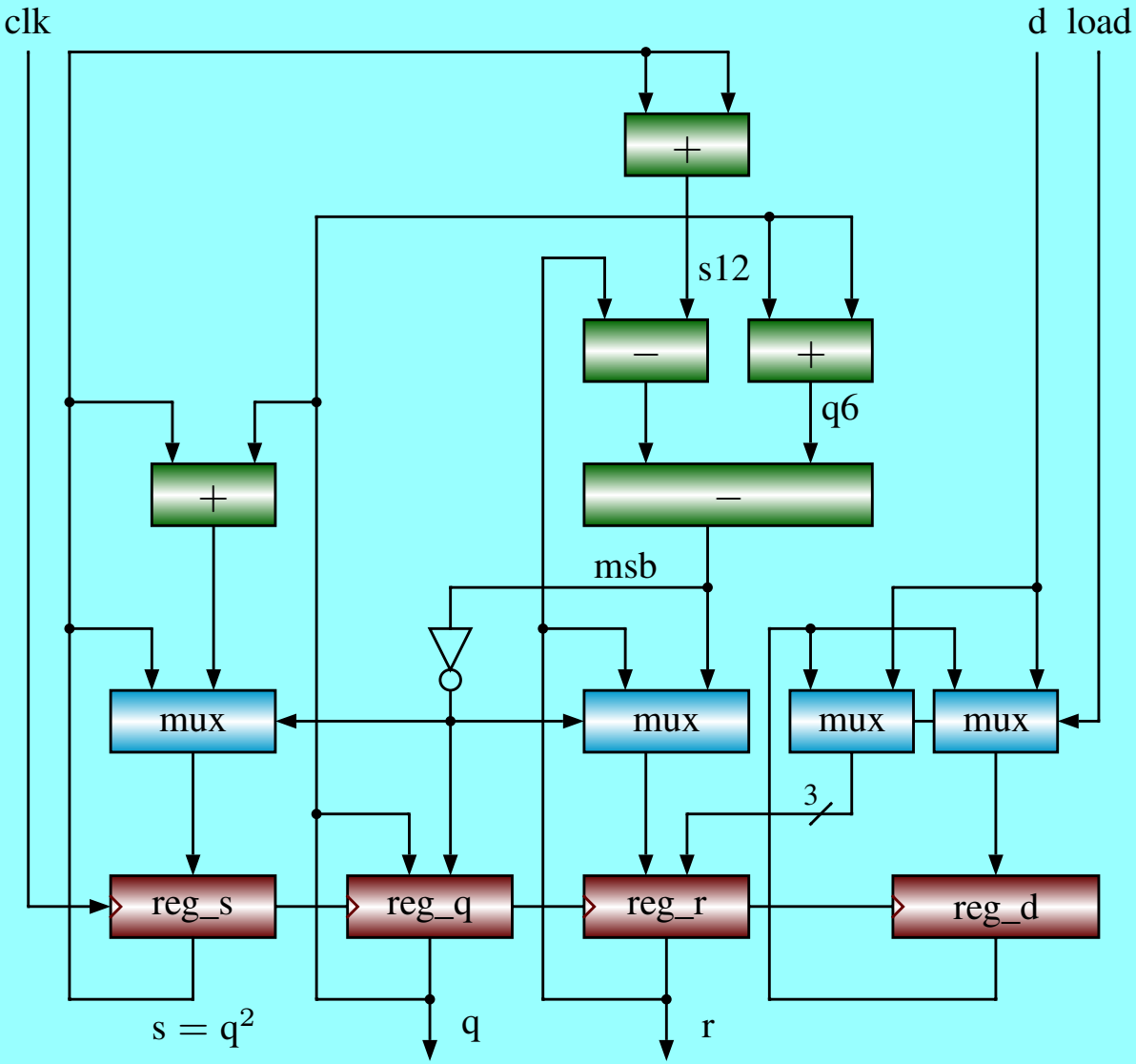
$$s_{i+1} = q_{i+1}^2 = (2q_i + 1)^2 = 4q_i^2 + 4q_i + 1 = 4s_i + 4q_i + 1$$

- Then $r_{i+1} = \{r_i, p_i\} - (12q_i^2 + 6q_i + 1)$ became

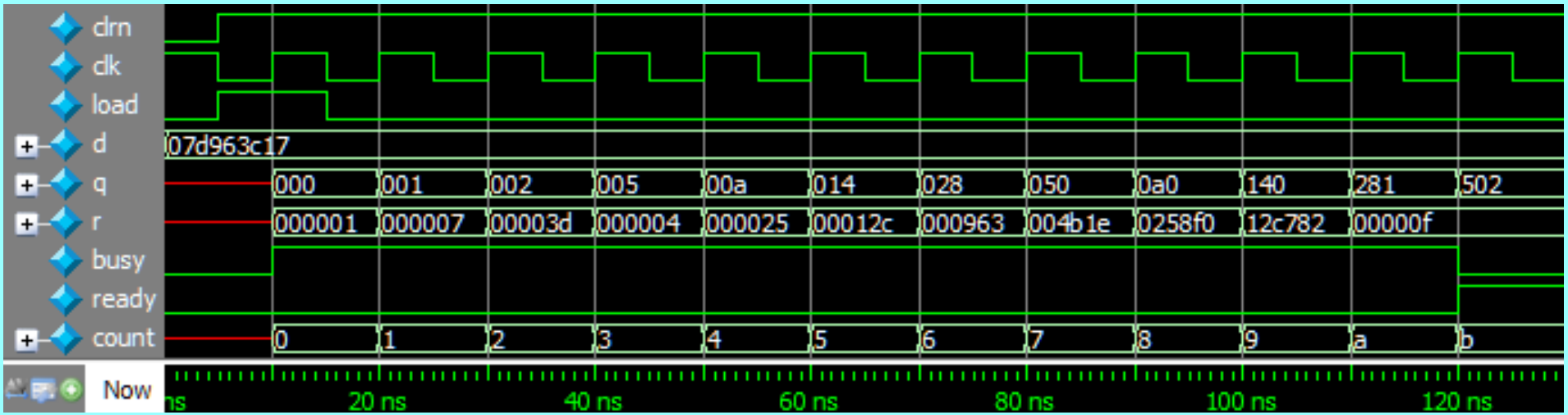
$$r_{i+1} = (\{r_i, p_i\} - (8s_i + 4s_i)) - (4q_i + 2q_i + 1)$$

Circuit of Restoring Cube Root (No mul)

$$r_{i+1} = (\{r_i, p_i\} - (8s_i + 4s_i)) - (4q_i + 2q_i + 1)$$



Waveform of Restoring Cube Root Algorithm



- Implemented on Altera Cyclone IV E EP4CE115F29C7
- 220 LEs (logic elements) (160)
- 92 DFFs (D flip-plops) (72)
- No embedded multiplier is required (2)
- Maximum clock frequency: 126.06 MHz (73.87 MHz)

Calculating $12q^2$ in Advance

- Let $w_{i+1} = 12s_{i+1}$
- For the case of $b_{i+1} = 0$

$$w_{i+1} = 12q_{i+1}^2 = 12(2q_i + 0)^2 = 48s_i$$

- For the case of $b_{i+1} = 1$

$$w_{i+1} = 12q_{i+1}^2 = 12(2q_i + 1)^2 = 48s_i + 48q_i + 12$$

- Then the calculation of r_{i+1} became

$$r_{i+1} = (\{r_i, p_i\} - w_i) - (4q_i + 2q_i + 1)$$

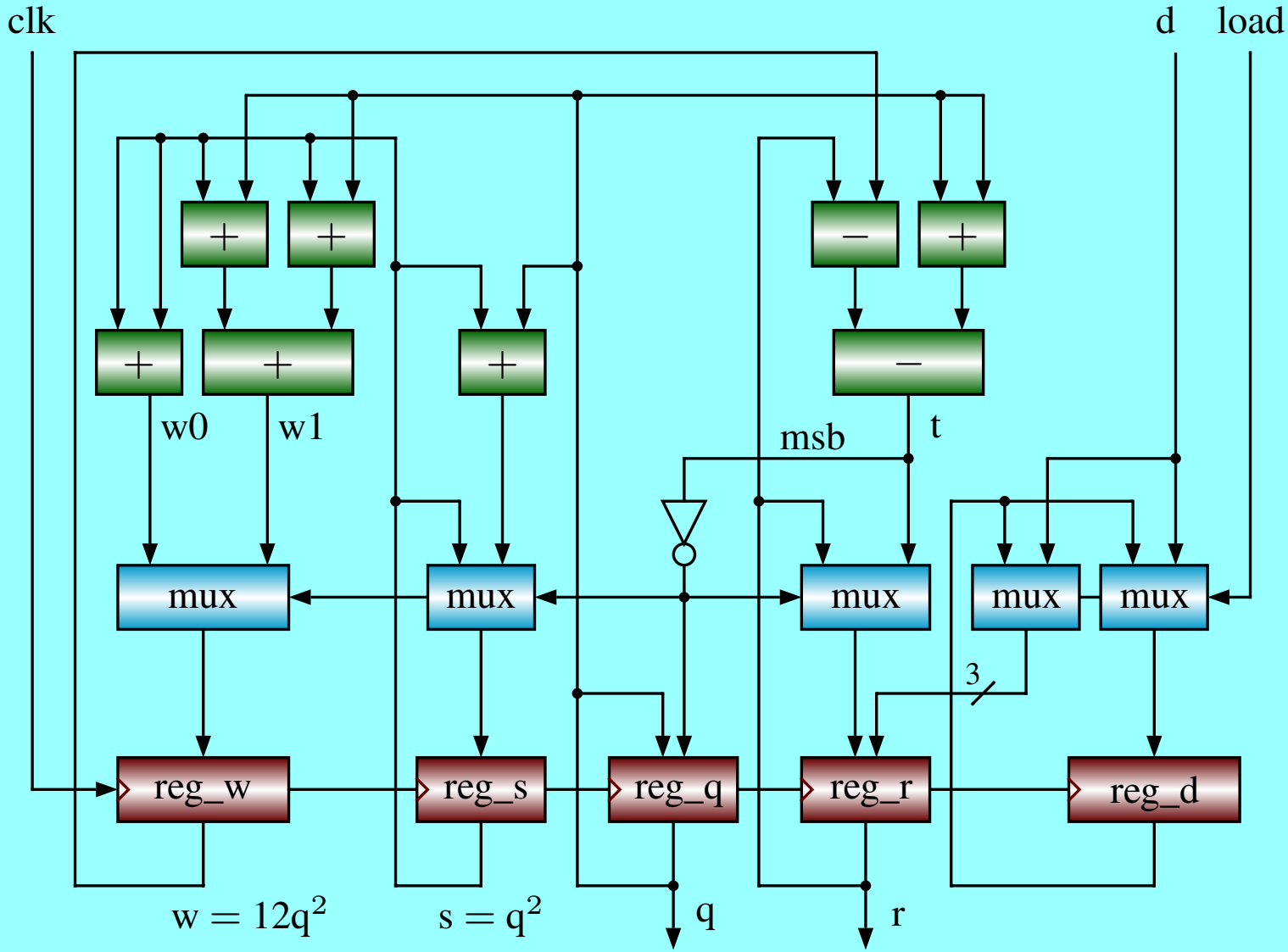
Algorithm 1: cube_root_restoring (d, q, r)

Input: 33-bit radicand $d = \{d_1, d_2, \ldots, d_{33}\}$
Output: 11-bit cube root $q = \{b_1, b_2, \ldots, b_{11}\}$
Output: 24-bit remainder r
begin

$r \leftarrow \{0, \ldots, 0, d_1, d_2, d_3\}$; /* 25 bits */
$d \leftarrow \{d_4, \ldots, d_{33}\}$; /* 30 bits */
$q \leftarrow \{0, \ldots, 0\}$; /* 11 bits */
$s \leftarrow \{0, \ldots, 0\}$; /* $s = q^2$, 22 bits */
$w \leftarrow \{0, \ldots, 0\}$; /* $w = 12q^2$, 25 bits */
for $i \leftarrow 1$ to 11 do /* for bits 1 to 11 */
  $p = d[29:27]$; /* 3 leftmost bits of d */
  $t = (r - w) - (\{q, 0, 1\} + \{q, 0\})$; /* rem */
  if $(t \geq 0)$ /* rem is non-negative */
    $w \leftarrow (\{s, 0, 0, 0, 0, 0\} + \{q, 0, 1, 0, 0, 0\}) +$
      $(\{\ \ s, 0, 0, 0, 0\} + \{\ \ q, 0, 1, 0, 0\})$;
    $s \leftarrow \{s, 0, 0\} + \{q, 0, 1\}$; /* $\{q, 1\}^2$ */
    $q \leftarrow \{q, 1\}$; /* $b_i = 1$ */
    if $(i < 11)$ $r \leftarrow \{t, p\}$; /* new r */
    else $r \leftarrow t[23:0]$; /* final r */
    endif
  else /* rem is negative */
    $w \leftarrow \{s, 0, 0, 0, 0, 0\} + \{s, 0, 0, 0, 0\}$;
    $s \leftarrow \{s, 0, 0\}$; /* $\{q, 0\}^2$ */
    $q \leftarrow \{q, 0\}$; /* $b_i = 0$ */
    if $(i < 11)$ $r \leftarrow \{r, p\}$; /* restoring r */
    else $r \leftarrow r[23:0]$; /* final r */
    endif
  endif
  $d \leftarrow \{d, 0, 0, 0\}$; /* shift 3-bit left */
endfor
end

Circuit of Restoring Cube Root (Balanced)

$$r_{i+1} = (\{r_i, p_i\} - w_i) - (4q_i + 2q_i + 1)$$



cube_root_restoring.v

```
// Copyright by Yamin Li and Wanming Chu, 2016
module cube_root_restoring (
  input  [32:0] d,                    // 33-bit radicand
  input         load,                          // start
  input         clk,                           // clock
  input         clrn,                          // reset
  output [10:0] q,                  // 11-bit cube root
  output [23:0] r,                  // 24-bit remainder
  output reg    busy,                  // circuit busy
  output reg    ready);                // results ready

  reg    [29:0] reg_d;                  // register for d
  reg    [10:0] reg_q;                  // register for q
  reg    [24:0] reg_r;                  // register for r
  reg    [23:0] reg_w;              // register for 12q^2
  reg    [21:0] reg_s;                // register for q^2
  reg     [3:0] count;              // counter for control
  wire   [24:0] rem;                   // signed remainder
```

cube_root_restoring.v

```
wire   [23:0] w0,w1,w;                          // for reg_w
wire   [21:0] rs;                               // for reg_s
wire   [24:0] rr;                               // for reg_r
assign w0  =  {reg_s[18:0],5'd0} +                 // 32q^2
              {reg_s[19:0],4'd0};                  // 16q^2
assign w1  = ({reg_s[18:0],5'd0} +                 // 32q^2
              { 8'd0,reg_q,5'd8})+                 // 32q+8
             ({reg_s[19:0],4'd0} +                 // 16q^2
              { 9'd0,reg_q,4'd4});                 // 16q+4
assign rem = (reg_r - {1'd0,reg_w}) -                 // r-w
             ({12'd0,reg_q,2'd0} +                    // 4q
              {13'd0,reg_q,1'd1});                  // 2q+1
wire   b   = ~rem[24];                          // root bit
assign rs  = b ? {reg_s[19:0],2'd0} +              // 4q^2
                 { 9'd0,reg_q,2'd1} :              // 4q+1
                 {reg_s[19:0],2'd0};               // 4q^2
assign rr  = b ? {  rem[21:0],reg_d[29:27]} :
                 {reg_r[21:0],reg_d[29:27]};
```

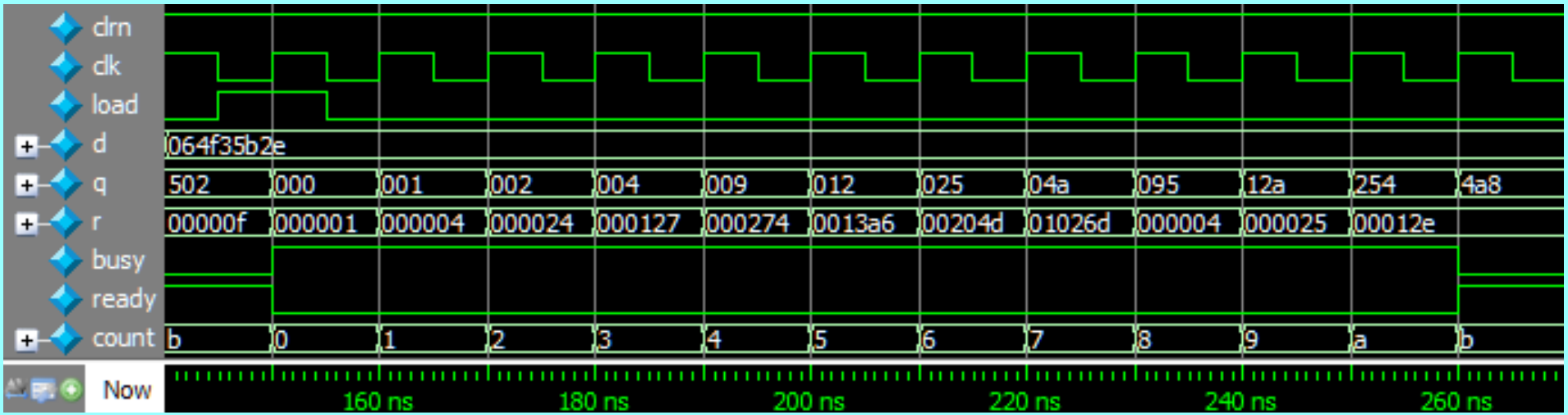
cube_root_restoring.v

```
assign w   = b ? w1 : w0;                          // 12q^2
wire  [3:0] count_plus = count + 4'd1;
wire        counter10  = (count == 4'd10);
always @ ( posedge clk or negedge clrn ) begin
  if (!clrn) begin                  // on reset, active low
    busy  <= 0;                      // circuit is not busy
    ready <= 0;                    // results are not ready
  end else begin                              // not on reset
    if (load) begin                       // load radicand d
      reg_d <= d[29:0];                            // 30 bits
      reg_q <= 0;                                    // q_0=0
      reg_r <= {22'd0,d[32:30]};                      // r_0
      reg_s <= 0;                                     // q^2
      reg_w <= 0;                                   // 12q^2
      busy  <= 1;                           // circuit is busy
      ready <= 0;                    // results are not ready
      count <= 0;                          // clear counter
    end else if (busy) begin                 // calculating
```

cube_root_restoring.v

```
        reg_d <= {reg_d[26:0],3'd0};              // shift
        reg_q <= {reg_q[9:0],b};                  // shift
        reg_s <= rs;                                // q^2
        reg_w <= w;                               // 12q^2
        if (counter10) begin                     // finish
          busy  <= 0;                  // circuit is idle
          ready <= 1;                // results are ready
          if (b) reg_r <= rem;                // remainder
        end else begin                          // not finish
          reg_r <= rr;
        end
        count <= count_plus;                    // counter++
      end
    end
  end
  assign q = reg_q;                          // final cube root
  assign r = reg_r[23:0];                    // final remainder
endmodule
```

Waveform of Restoring Cube Root Algorithm



- Implemented on Altera Cyclone IV E EP4CE115F29C7
- 271 LEs (logic elements) (220)
- 109 DFFs (D flip-plops) (92)
- No embedded multiplier is required (0)
- Maximum clock frequency: 134.44 MHz (126.06 MHz)

Using Carry Save Adders

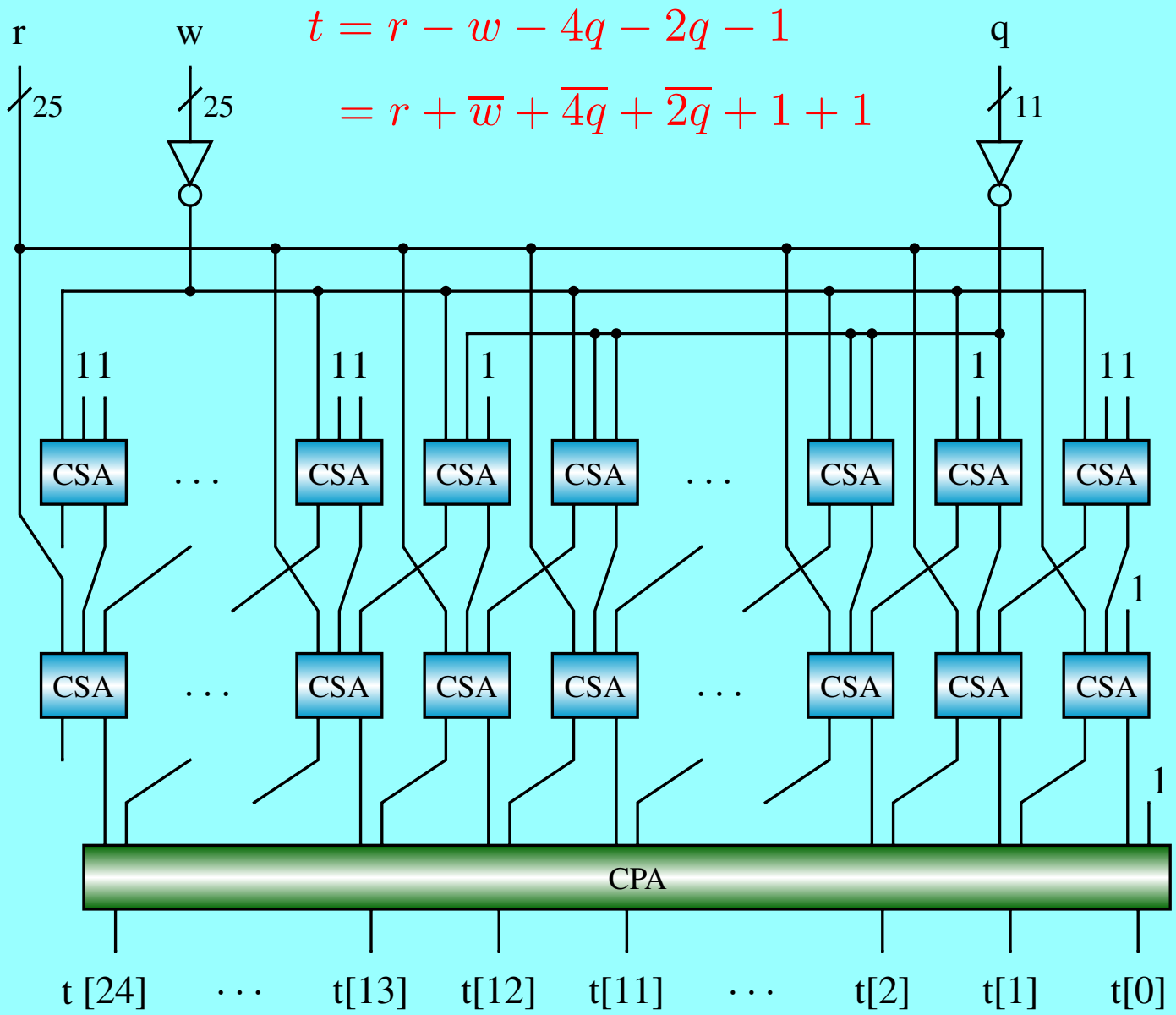
- Two-level carry propagate adders (CPAs) to calculate t
- We can use the carry save adders (CSAs) to speed up it

$$\begin{aligned} t &= \{r, p\} - w - 4q - 2q - 1 \\ &= \{r, p\} + (\overline{w} + 1) + (\overline{4q} + 1) + (\overline{2q} + 1) - 1 \\ &= \{r, p\} + \overline{w} + \overline{4q} + \overline{2q} + 1 + 1 \end{aligned}$$

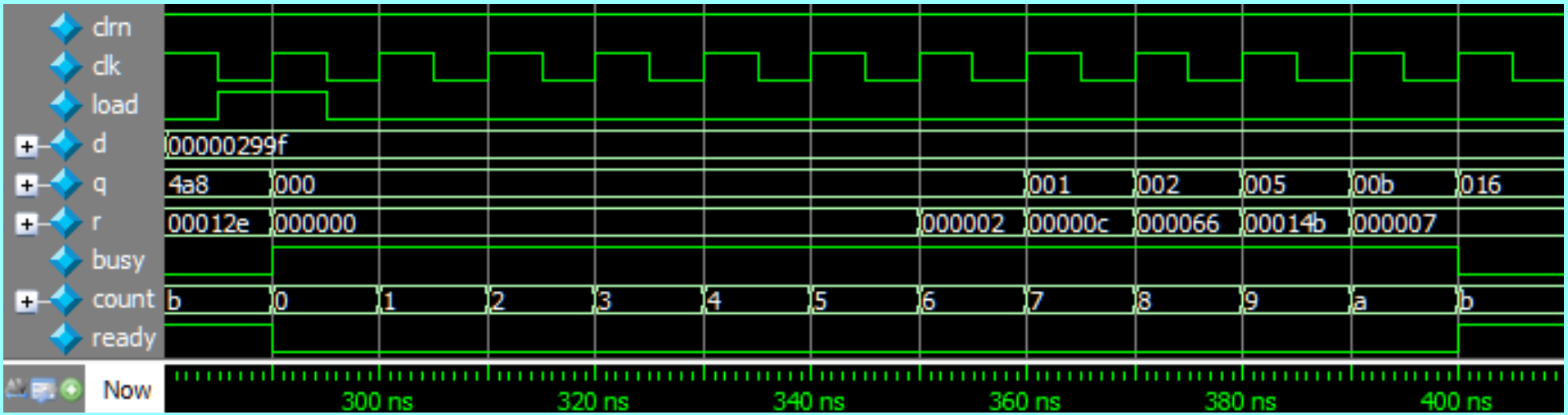
- We use two-level CSAs to perform the additions and get two numbers: carry c and sum s
- Then we use a CPA to get the temporary remainder

$$t = 2c + s$$

Using Carry Save Adders



Waveform of Restoring Cube Root Algorithm



- Implemented on Altera Cyclone IV E EP4CE115F29C7
- 298 LEs (logic elements) (271)
- 109 DFFs (D flip-plops) (109)
- No embedded multiplier is required (0)
- Maximum clock frequency: 147.47 MHz (134.44 MHz)

Non-Restoring Cube Root Algorithms

- In restoring algorithm, if $r_i < 0$, r_i is restored by adding $(12q_{i-1}^2 + 6q_{i-1} + 1)$ to r_i
- Then we perform $r_{i+1} = \{r_i, p_i\} - (12q_i^2 + 6q_i + 1)$
- Because $q_i = \{q_{i-1}, 0\}$ or $q_{i-1} = q_i/2$, we have

$$\begin{aligned} r_{i+1} &= (8(r_i + 12q_{i-1}^2 + 6q_{i-1} + 1) + p_i) - (12q_i^2 + 6q_i + 1) \\ &= (8r_i + 24q_i^2 + 24q_i + 8 + p_i) - (12q_i^2 + 6q_i + 1) \\ &= \{r_i, p_i\} + 12q_i^2 + 18q_i + 7 \end{aligned}$$

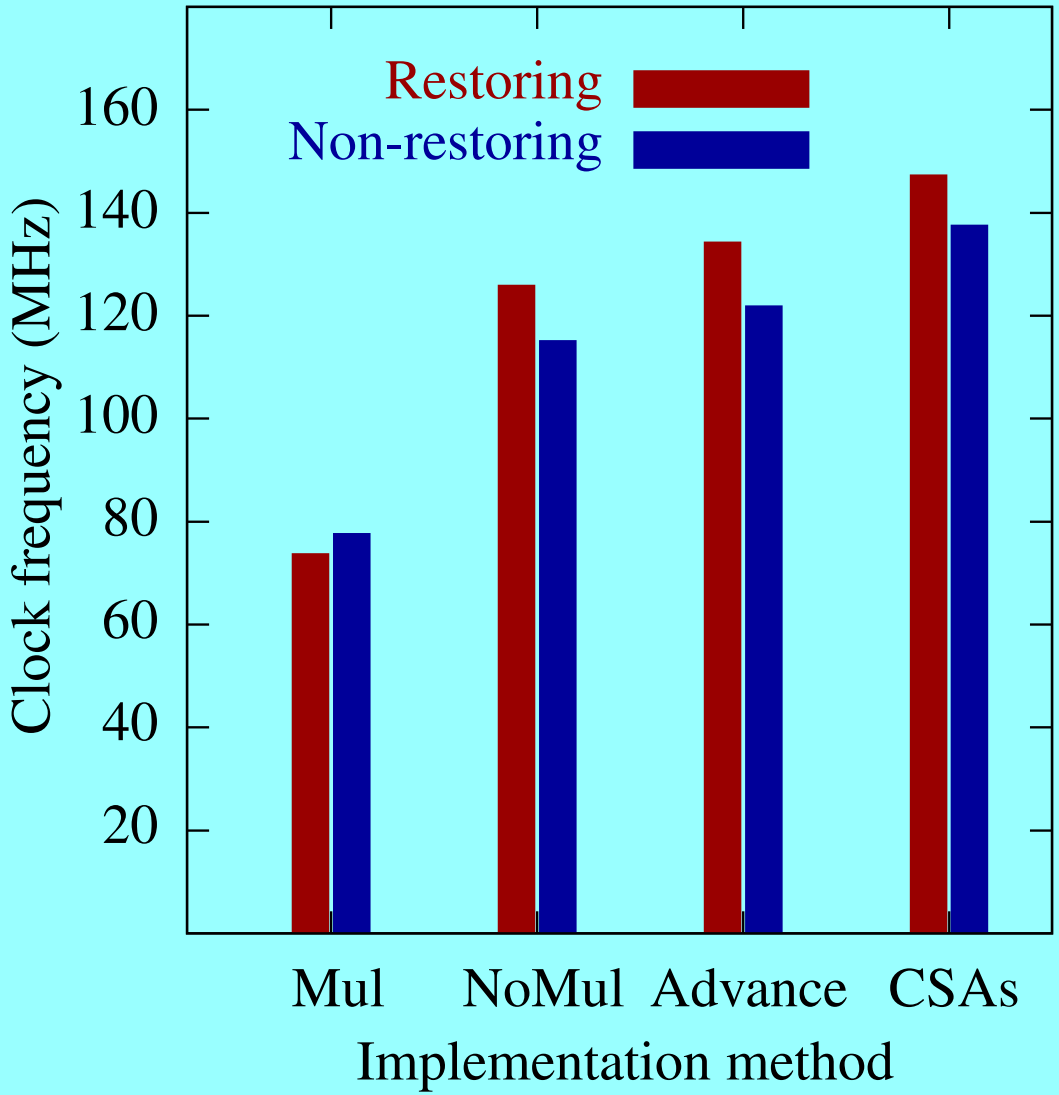
- Non-restoring cube root algorithm

$$r_{i+1} = \{r_i, p_i\} - (12q_i^2 + \ \ 6q_i + 1) \ \text{ if } r_i \geq 0$$
$$r_{i+1} = \{r_i, p_i\} + (12q_i^2 + 18q_i + 7) \ \text{ if } r_i < 0$$

Cost / Performance Evaluation

Implementation	Frequency	Cycle time	Latency	Logic	Reg	Mul
1. Restoring (Mul)	73.87 MHz	13.54 ns	148.91 ns	160	72	2
2. Restoring (NoMul)	126.06 MHz	7.93 ns	87.26 ns	220	92	0
3. Restoring (Advance)	134.44 MHz	7.44 ns	81.82 ns	271	109	0
4. Restoring (CSAs)	147.47 MHz	6.78 ns	74.59 ns	298	109	0
5. Non-rest. (Mul)	77.77 MHz	12.86 ns	141.44 ns	338	73	2
6. Non-rest. (NoMul)	115.23 MHz	8.68 ns	95.46 ns	380	93	0
7. Non-rest. (Advance)	122.03 MHz	8.19 ns	90.14 ns	433	112	0
8. Non-rest. (CSAs)	137.70 MHz	7.26 ns	79.88 ns	492	112	0

Cost / Performance Evaluation



Restoring cube root:

$\text{Speedup}_{CSA} = 147.47/73.87 = 199.63\%$

Non-restoring cube root:

$\text{Speedup}_{CSA} = 137.70/77.77 = 177.06\%$

Both are low at cost

FYI: Square Root Algorithms

- Square Root Algorithms
  - Restoring and non-restoring algorithms
  - Goldschmidt algorithm
  - Newton-Raphson algorithm
- Refer to
  - Yamin Li, “Computer Principles and Design in Verilog HDL”, *John Wiley & Sons*, ISBN 978-1-118-84109-9, 2015, 550 pages
- Thank you very much!