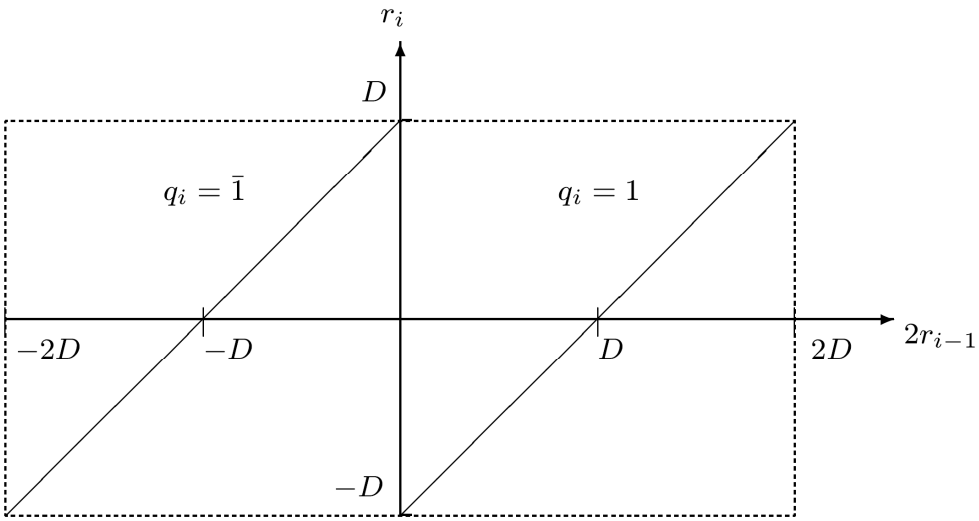
[2010년 디지털 시스템 설계 및 실험 중간고사 2 답안지]

출제: 채수익

1.

(a) (10 pts)

Robertson diagram



Quotient와 remainder의 correction을 뒤로 미루는 것이 non-restoring division이다.

즉,

$$q_i = \begin{cases} 1, & 2r_{i-1} \geq 0 \\ \bar{1}, & 2r_{i-1} < 0 \end{cases}, \quad r_i = 2r_{i-1} - q_i \cdot D$$

만일 연산 결과 remainder와 dividend의 sign이 다를 경우, correction step이 필요하다.

(1) dividend와 divisor가 sign이 같으면, remainder에 D를 더하고, quotient에서 ulp를 뺀다.

(2) divident와 divisor의 sign이 다르면, remainder에서 D를 빼고, quotient에 ulp를 더한다.

또한 만일 중간에 remainder가 0이 된다면, correction이 필요하다.

이 경우, remainder에 D를 더하고, quotient에서 ulp를 뺀다.

Robertson diagram을 정확히 그린 경우 (5 pts)

- x, y축의 값들을 정확히 기술하지 않았을 경우 (-2 pts)
- x, y축이 의미하는 바를 정확히 기술하지 않았을 경우 (-2 pts)
- q_i가 해당 영역에서 어떤 의미를 갖는지 정확히 기술하지 않았을 경우 (-1 pts)

Non-restoring division algorithm에 대한 설명을 정확히 한 경우 (5 pts)

- q_i와 r_i가 어떻게 결정되는지 수식 혹은 말로 설명하지 않았거나 틀렸을 경우 (-1 pts)
- 연산 결과의 부호 correction step에 대해서 설명하지 않았거나 틀렸을 경우 (-1 pts)
- Zero remainder의 경우에 대한 correction을 설명하지 않았거나 틀렸을 경우 (-1 pts)

(b) (10 pts)

$\bar{1}$, 1을 각각 0, 1로 표기한다고 가정하면,

Step 1. Shift the given number one bit position to the left

Step 2. Complement the most significant bit

Step 3. Shift a 1 into the least significant position

2's complement number로 바꾸는 알고리듬을 기술한 경우 (10 pts)

$1\bar{1}$을 01로 바꾼다고 쓴 경우를 비롯해 일반적이지 않은 경우 (5 pts)

2.

(1) (4 pts)

S	8 bits - biased exponent E	23 bits - unsigned fraction f

Base는 2고, hidden bit를 사용한다.

E = 0는, f = 0인 경우 zero에 해당하고, f≠0인 경우 denormalized number에 해당한다.

E = 255는, f = 0인 경우 ±∞에 해당하고, f≠0인 경우 NaN에 해당한다.

1 < E < 254인 경우에는

$$F = (-1)^S \, 1.f \, 2^{E-127}.$$

E가 8 bit이고, f가 23 bit이며, S가 1 bit이라고 기술하고, base는 2, hidden bit를 사용한다고 기술한 경우 (1 pts)

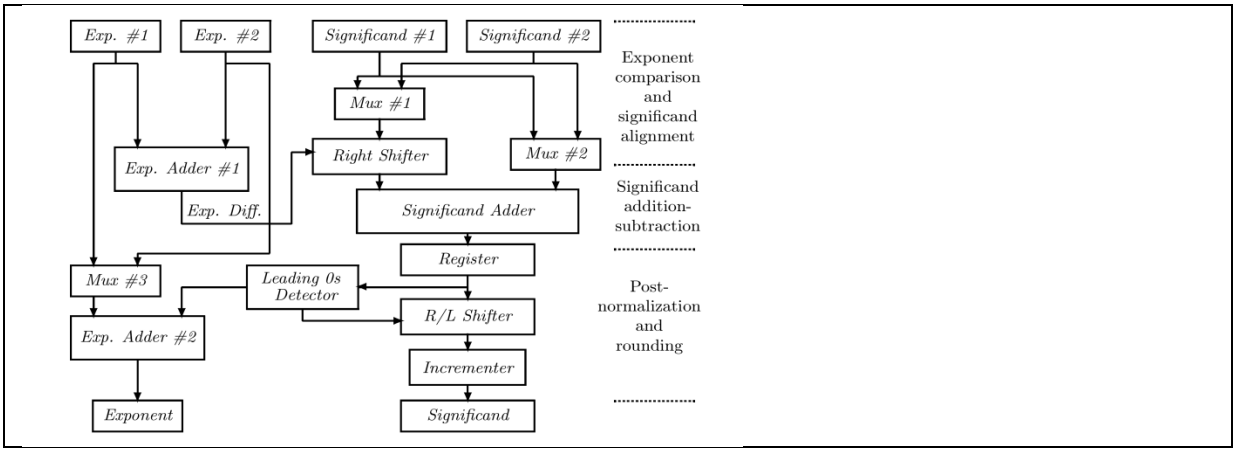
E = 0인 경우에 대해 기술한 경우 (1 pts)

E = 255인 경우에 대해 기술한 경우 (1 pts)

1 < E < 254인 경우에 대해 기술한 경우 (1 pts)

각 경우에 대해 하나라도 틀리게 쓴 경우, 해당 경우에 대해 0 pts 부여

(2) (4 pts)



빠져있는 block이 있거나 잘못된 경우 -1 pts씩

4개 이상 잘못된 경우 0 pts

(3) (4 pts)

Exponenet comparison and significand alignment significand addition/subtraction post-normalization and rounding 에 대한 내용을 block에 대한 설명과 함께 기술

설명이 일부 빠져있는 경우 (-1 pts)

(4) (8 pts)

Shifter를 barrel shifter로 구현 leading 0s detect를 미리 수행

두 가지 이상 적당한 아이디어들을 기술한 경우 (8 pts)
한 가지만 기술한 경우 (6 pts)

3.

(1) (10 pts)

$$\delta(x) \leq \frac{\frac{1}{2}\, ulp\ \beta^E}{M\beta^E} = \frac{1}{2}\,\frac{ulp}{M} \leq \frac{1}{2}\frac{ulp}{\frac{1}{\beta}} = \frac{1}{2}\, ulp \cdot \beta$$ (MRRE)

$$\mathrm{ARRE} = \int_{\frac{1}{\beta}}^{1} \frac{1}{M \ln \beta}\,\frac{ulp}{4M}\, dM = \frac{\beta - 1}{\ln \beta}\,\frac{ulp}{4}$$

각 경우에 대해 5 pts씩
사소한 실수 (-1 pts씩)

(2) (5 pts)

$0.36 \times 2^{-24} = \frac{1}{4\ln 2} \times 2^{-24}$
(주의 : hidden bit의 존재로 ulp가 2^{-24}이다)

사소한 실수 (-1 pts씩)

4.

(1) (5 pts)

	i	7	6	5	4	3	2	1	0
	x_i	1	0	1	1	0	1	1	0
	y_i	0	0	1	0	1	1	0	1
Step 1	s_i^0	1	0	0	1	1	0	1	1
	c_{i+1}^0	0	0	1	0	0	1	0	0
	s_i^1	0	1	1	0	0	1	0	
	c_{i+1}^1	1	0	1	1	1	1	1	
Step 2	s_i^0	1	0	0	1	0	0	1	1
	c_{i+1}^0	0		1		1		0	
	s_i^1	1	1	1	0	0	1		
	c_{i+1}^1	0		1		1			
Step 3	s_i^0	1	1	0	1	0	0	1	1
	c_{i+1}^0	0				1			
	s_i^1	1	1	1	0				
	c_{i+1}^1	0							
Result		1	1	1	0	0	0	1	1

위는 8-bit adder의 관한 경우로, 16-bit adder의 경우, 한 step이 더 있게 된다. 또한 최초의 step에서 LSB의 경우, full adder가 1개만 필요한 것에 유의한다.

Step 0 to Step 1. Full adder 31개

Step 1 to Step 2. 1-bit 2to1 Mux 30개 (=2*2*7+2)

- 예를 들어, i=6인 경우에 c_in이 0, 1인 경우 각각에 대해 i=7의 sum과 carry out을 각각 선택해야 한다. LSB를 제외하면 반복되므로, 2*2*7이며, LSB에 해당하는 경우는 i=0가 이미 결정되어 있기 때문에 i=1의 sum과 carry out을 선택하기 위한 mux가 각각 1개씩 필요하다.

Step 2 to Step 3. 1-bit 2to1 Mux 21개

Step 3 to Step 4. 1-bit 2to1 Mux 15개

Step 4 to result. 1-bit 2to1 Mux 9개

총 FA 31개, 1-bit 2to1 mux 75개

Block diagram이나 말로 위의 구현 방법을 설명 (3 pts)

- 16-bit carry save adder를 쓴 경우 (1 pts)
- Multi step으로 구성하지 않고 최초의 stage에 대해서만 conditional sum을 한 경우 (1 pts)

Full adder와 1-bit 2to1 mux의 개수 (각각 1 pts) (주의 : 1-bit 2to1 mux의 개수임)

(2) (10 pts)

```
module mux_2to1(x, a, b, select);
parameter n = 8;
input select;
input [n-1:0] a, b;
output [n-1:0] x;

wire [n-1:0] a, b, x;
assign x = (select == 1'b0) ? a : b;
```

```
endmodule
```

```
module full_adder(sum, c_out, a, b, c_in);
input a, b, c_in;
output sum, c_out;

wire sum, c_out, tmp1, tmp2, tmp3;
assign sum = a ^ b ^ c_in;
assign tmp1 = a & b;
assign tmp2 = b & c_in;
assign tmp3 = a & c_in;
assign c_out = tmp1 | tmp2 | tmp3;

endmodule
```

```
module conditional_sum_adder(sum, c_out, a, b, c_in)
input [15:0] a, b;
input c_in;
output [15:0] sum;
output c_out;

... block diagram에 맞게끔 코딩 ...

endmodule
```

Full adder, mux module 각각 3 pts

Conditional sum adder를 맞게 기술한 경우 (4 pts)

(1)에서 conditional sum adder를 잘못 기술하고 이를 verilog로 쓴 경우 (2 pts)

5. (15 pts)

associative (5 pts)

증명은 아래와 같다.

$$\big((P_1, G_1)\circ(P_2, G_2)\big)\circ(P_3, G_3) = (P_1 \cdot P_2, G_1 + P_1 \cdot G_2)\circ(P_3, G_3) = (P_1 \cdot P_2 \cdot P_3, G_1 + P_1 \cdot G_2 + P_1 \cdot P_2 \cdot G_3)$$
$$= \big(P_1 \cdot (P_2 \cdot P_3), G_1 + P_1 \cdot (G_2 + P_2 \cdot G_3)\big) = (P_1, G_1)\circ(P_2 \cdot P_3, G_2 + P_2 \cdot G_3) = (P_1, G_1)\circ((P_2, G_2)\circ(P_3, G_3))$$

commutative (5 pts)

성립하지 않는다.

반례 :

$(0, 0)\circ(1, 1) = (0, 0) \neq (0, 1) = (1, 1)\circ(0, 0)$

idempotent (5 pts)

증명은 아래와 같다.

$(P, G)\circ(P, G) = (P{\cdot}P, G{+}P{\cdot}G) = (P, G)$

associative, commutative, idempotent가 무엇인지 기술할 경우, 각 항목당 2 pts

proof, 혹은 disproof를 맞게 제시한 경우, 각 항목당 5 pts
Proof나 disproof 없이 fundamental carry operator만 기술한 경우 (2 pts)
그 외 경우, (0 pts)

6.

Number of operands	Number of levels using (3,2)	Number of levels using (4;2)	Equivalent delay
3	1	1	1.5
4	2	1	1.5
5 – 6	3	2	3
7 – 8	4	2	3
9	4	3	4.5
10 – 13	5	3	4.5
14 – 16	6	3	4.5
17 – 19	6	4	6
20 – 28	7	4	6
29 – 32	8	4	6
33 – 42	8	5	7.5

위의 표를 참고하여 문제를 푼다. n을 2의 power로 표현되는 4 이상의 정수로 제한했으므로, 위의 표에서는 4, 8, 16, 32에 대한 경우를 참고하면 된다.

(1) (5 pts)

$$2 \times (\log_2(n) - 1)$$

맞을 경우 (5 pts)
$n\left(\frac{2}{3}\right)^{l} \leq 2$를 이용한 경우 (3 pts) (각 level 사이가 integer라는 조건이 추가로 있기 때문에 approximation이 계산될 뿐, 정확한 값이 도출되지는 않는다)
n이 고정된 한 경우(예를 들어, n = 4인 경우)에 대해서만 쓴 경우 (1 pts)
그 외 (0 pts)

(2) (5 pts)

$$\log_2(n) - 1$$

맞을 경우 (5 pts)
식의 정리가 덜 된 경우 (3 pts) (ceiling function 등이 남아 있는 경우)
n이 고정된 한 경우(예를 들어, n = 4인 경우)에 대해서만 쓴 경우 (1 pts)
그 외 (0 pts)

(3) (5 pts)

위의 level에 2D와 3D를 곱해서 비교해보면,

$$4D \times (\log_2(n) - 1) > 3D \times (\log_2(n) - 1)$$

(3,2) counter의 delay가 더 크다.
즉, 주어진 조건에서 항상 (4,2) compressor를 사용한 곱셈기가 항상 빠르다.

맞을 경우 (5 pts)
틀릴 경우 (0 pts)

7.

(1) (5 pts)

$$k \cdot (\frac{2}{3})^l \leq 2$$

$$\mathrm{l} = \frac{\log(\mathrm{k}/2)}{\log(3/2)}$$

단, 위의 경우는 approximation으로 정확한 값은 (2)의 table을 참고한다.

위의 수식 중 하나를 쓰고 approximation이라는 말을 쓴 경우 (5 pts)

approximation이라는 말이 없는 경우 (-1 pts)

Level 수를 직접 얻지는 못 하지만, 관련된 수식을 쓴 경우 (3 pts)

(2) (5 pts)

Number of operands	Number of levels
3	1
4	2
$5 \leq k \leq 6$	3
$7 \leq k \leq 9$	4
$10 \leq k \leq 13$	5
$14 \leq k \leq 19$	6
$20 \leq k \leq 28$	7
$29 \leq k \leq 42$	8
$43 \leq k \leq 63$	9

위의 table과 정확히 일치하는 경우 (5 pts)

사소한 실수 (-1 pts)

(1)의 수식을 사용하여 table을 만든 경우 (-3 pts)

8.

(1) (5 pts)

```
sequential_multiplier (A, X, n)
  P <- 0;
  for (i = 0; i < n; i++) {
    if (X[i] == 1) {
      P = P + A;
    }
    P >>> 1; // arithmetic shift
  }
  if (X < 0 && P < 0) {
    P = P – A;
  }
  return P
```

위와 같이 pseudo-code 형태로 sequential multiplier를 기술한 경우 (5 pts)

- Booth algorithm 등 다른 형태로 기술한 경우 (-1 pts)

마지막의 correction step을 쓰지 않았거나 틀린 경우 (-2 pts)

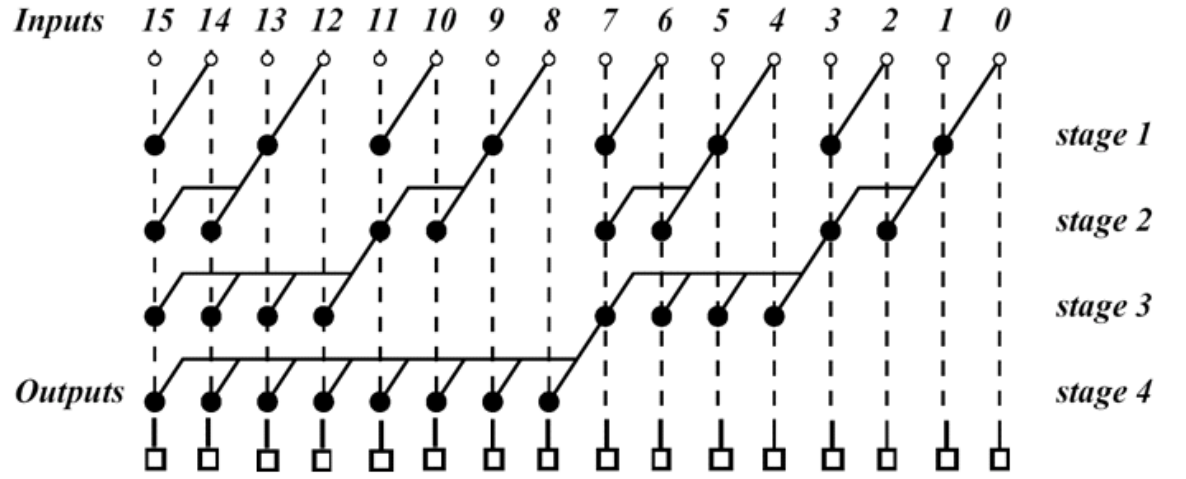
(2) (10 pts)
(1)에서 제시한 알고리즘과 유사한 형태로 기술한 경우 (10 pts) (multi-cycle도 상관없음)
adder만 기술한 경우 (5 pts)
마지막의 correction step을 쓰지 않은 경우 (-3 pts)
Verilog code의 합성가능성은 무관함

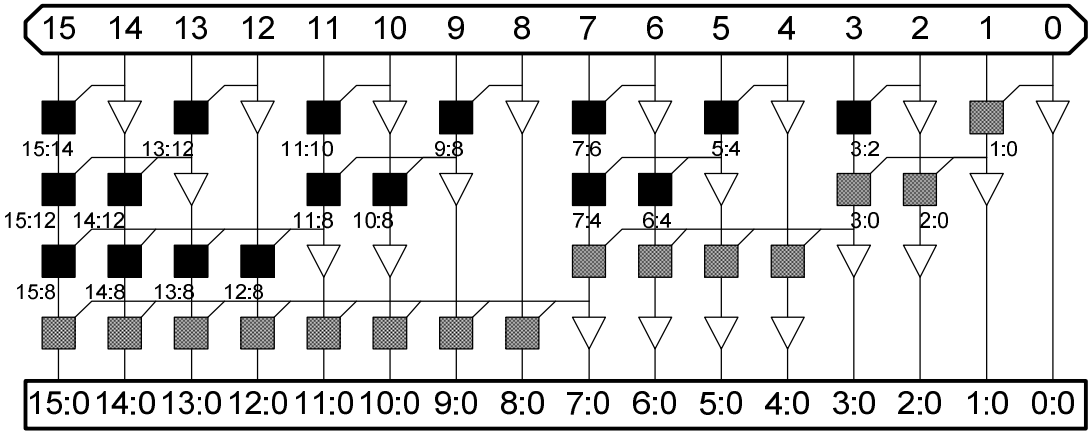
9.
(1) (6 pts)

Logic levels : L + l (L = $\log_2 n$)
Fanout : $2^f + 1$
Wiring tracks : 2^t

Logic level, fanout, wiring track이라는 단어를 기술한 경우 (각 1 pts)
수식을 제대로 기술한 경우 (각 1 pts)

(2) (5 + 2 pts)





맨 위의 그림을 문제의 조건에 맞게 변형한다. 단, black 사각형과 gray 사각형의 구분은 아래의 그림을 따른다.

P, G generator / fundamental carry operator / sum generator 각각 1 pts (다 맞은 경우 +2 pts)
사각형의 경우, black과 gray를 정확히 맞춘 경우에만 2 pts

(3) (5 pts)

Maximum delay는 MSB에서 발생한다. (2)의 그림을 참고하면,
1D+4D+2D = 7D

정확히 수치가 맞은 경우에만 5 pts, 틀리면 0 pts

(4) (10 pts)

P, G generator / fundamental carry operator / sum generator 각각에 대해 2 pts씩
이를 이용해 Sklansky adder를 기술한 경우 (4 pts)

10.

(1) (5 pts)

x_i	x_{i-1}	x_{i-2}	x_{i-3}	y_i	y_{i-1}	y_{i-2}	operation	comments
0	0	0	0	0	0	0	+0	string of zeros
0	0	1	0	0	0	1	+A	a single 1
0	1	0	0	0	1	0	+2A	a single 1
0	1	1	0	0	1	1	+3A	two 1's
1	0	0	0	$\bar{1}$	0	0	-4A	beginning of 1's
1	0	1	0	0	$\bar{1}$	$\bar{1}$	-3A	alternating
1	1	0	0	0	$\bar{1}$	0	-2A	beginning of 1's
1	1	1	0	0	0	$\bar{1}$	-A	beginning of 1's
0	0	0	1	0	0	1	+A	end of 1's
0	0	1	1	0	1	0	+2A	end of 1's

0	1	0	1	0	1	1	+3A	alternating
0	1	1	1	1	0	0	+4A	end of 1's
1	0	0	1	0	$\bar{1}$	$\bar{1}$	-3A	two 0's
1	0	1	1	0	$\bar{1}$	0	-2A	a single 0
1	1	0	1	0	0	$\bar{1}$	-A	a single 0
1	1	1	1	0	0	0	+0	string of 1's

Recoding table이 정확한 경우 (comments는 상관없음) (5 pts)

일부 실수가 있는 경우 (-1 pts씩)

(2) (5 pts)

Partial products의 개수가 줄어든다.

장점을 정확하게 쓴 것이 한 가지 이상 있는 경우 (5 pts)

single 0나 single 1에 대한 처리를 쓴 경우 (3 pts) (radix-2에서 radix-4로 갈 때의 장점)

(3) (10 pts)

Unsigned를 곱할 경우, MSB에 0을 추가해줘야 제대로 된 결과가 나온다.

Radix 8 modified Booth's algorithm을 사용하면, partial product의 개수가 3개로 줄어든다. 또한 중간에 3-bit shift를 해주어야 하며, 최대 4A까지 더하거나 빼기 때문에 아래와 같이 덧셈이 이루어진다.

	2 bit	3 bit	3 bit	3 bit
2 bit	3 bit	3 bit	3 bit	

가장 아랫부분의 덧셈에서 half adder를 사용하고 이후부터는 full adder를 사용한다고 하자. 따라서 delay는 $10\times \Delta FA + 1 \times \Delta HA$가 된다. 이와 같은 덧셈이 partial product의 개수인 3회만큼 일어나므로,

delay = $30\times \Delta FA + 3 \times \Delta HA$

(단, shift 및 recoding table을 읽어오는데 걸리는 delay는 무시한다고 가정한다)

MSB에 0을 추가해줘야 한다는 내용이 있음 (3 pts)

덧셈이 몇 회 이루어지는가(partial product의 개수)에 대한 설명 (3 pts)

Delay가 맞을 경우 (4 pts)

- Carry save adder 형태로 구현한 경우 $17\times \Delta FA + 1 \times \Delta HA$로 쓴 경우 (4 pts)
- 4A, 3A 등에 대한 경우를 생각하지 않고 한 경우 (2 pts)