世界一高速な RSA 暗号 ICF3 初期検討

2016/05/13 版（初版)

2018/04/17 版 URL の更新のみ、文章の変更はありません

平山　直紀

著者概略

1992 年 早稲田大学理工学部　電気工学科卒

1994 年 早稲田大学理工学研究科 計算機工学専攻卒

1994 年 日立製作所 中央研究所　超高速プロセッサ部　入社

1995 年 日立製作所 汎用コンピュータ事業部に転勤

2005 年 日立製作所　退職

2006 年 株式会社ｉＣａｎａｌ設立、代表取締役社長

注意事項

この資料は報告書ではなくて製品出荷される１年前の検討資料なので、かなりいい加減な内容です。

このファイルを公開しているサイト URL

https://openicf3.idletime.tokyo/

1. はじめに

RSA とは公開鍵暗号のアルゴリズムです。CPU が作りたくて 1994 年に日立製作所に入社してハードウェアを開発することになったが、実際は、IBM が作った LSI の信号線を日立の LSI の信号線に合わせて配線するエクセル作業だった。簡単な作業のように聞こえるが非常に面倒な作業です。データがエクセルで人間にわかりやすい表記になっていたためプログラムで処理するには向かないデータでエクセルで手作業するしかなかった。あるとき手作業では、どうしても作業量が膨大になり、僕にプログラムでやってくれと頼んできた。Perl というテキスト処理に向いた言語でプログラムを作成してデータを変換した。後日、不良が発覚し１０００万円以上の損失をだした。僕とプログラムを依頼した先輩が特別会議室に呼ばれて怒られた。プログラムに誤りはなく、原因は信号線名に 1 文字だけ小文字が混じっていたことだった。エクセルで手作業でしていれば不良になることはなくエクセルのデータに責任を問うことができなかった。そして僕も責任を逃れた。要するに当時のハードウェア開発部が情弱で情報学科で訓練を積んだ僕には苦難が多かった。

1996 年、バイポーラから CMOS にテクノロジが遷移する時代だった。CMOS はバイポーラよりもノイズが多く、その評価は大切だった。当時のテクノロジ開発部は情弱で情報学科で訓練を積んだ僕が活躍する場があった。電気回路シミュレーションによる大量の波形データを Perl 言語で自作したプログラムで処理し gnuplot でノイズを可視化した。従来手法との違いを見せることができて高精度な結果を得ることができた。

この後、アメリカに 3 か月、FAX を受けるだけの仕事をして帰国後、暗号 LSI の開発を開始した。

第 1 作目の ICF1 は、他のメインフレームの暗号装置からの流用だった。暗号鍵の格納装置からの割り込み信号を、ポーリング方式に改造する仕事だった。論理設計の勉強をせずにやったので、不良を作りこんでしまった。後から聞いた話だが、これが一番損害が大きかったらしい。でも流用するのに仕様書も渡されず、全体の流れもわからないまま AND/OR の回路図だけを見て改造するのは至難の業だった。

だから全く怒られることなく選手交代し、他の人が修正した。

第 2 作目の ICF2 は、ICF1 とほぼ同じなのだが、日立独自の論理設計システムから Visual HDL に切り替わった。ICF1 でマスターしたノウハウ(論理シミュレーション含む)が一瞬でなくなり、また 1 から Visual HDL を勉強するハメになった。そして異様なスピードで開発をさせられた。この開発部は東大卒が多く、自分で責任をとれるからできたのだと思う。周囲から、ここだいじょうぶ？と心配された部分が、やっぱり不良だった。異様なスピードで開発したので言われても不良であるかを確認できる状態ではなかった、このため怒られなかった。

第 3 作目の ICF3 は、1999 年に世界一高速な RSA 暗号の LSI となったが、ICF1、ICF2 に RSA 暗号はなく、ICF3 で、僕が 1 から設計、実装した。この PDF は、このときの初期検討資料です。

２．製品化された製品の名称

日立製作所　中型メインフレーム MP5600EX の暗号装置　(1999 年 発売)

LSI 開発コード名　ICF3

ICF3 には、IBM よりも 5 倍以上高速な SHA-1 の演算装置も実装されています。この原案を考えたのも私で、実際に製品化されたものとほぼ同等のものの論理設計図を全部公開しています。

3. おわりに

初期検討資料なので、いい加減な資料ですが、最終的には案 3 が選択されました。急いで開発しなければ ICF3 に RSA 暗号を搭載することができず、モンゴメリ乗算は避けたいと、この資料にハンコを押した東大卒(中央研究所出身)の人に進言したのですが、「案 3 でも、この部分を直せばいける」という名言のもと、案 3 で実装検討に入ることになったのです。

モンゴメリ乗算についてシステム開発研究所から海外の資料をもらったのですが、不十分であった部分を、この資料の 1 ページ目にあるように、とっとと補って、ささっと 1024bit のレジスタ 4 本をもつ CPU を捻出して、あっという間に AND/OR で作ってしまったというのが、僕の成果で、メインフレームの売り上げに貢献し活躍した。

2018 年 4 月 17 日追記

資料の 1 ページ目と 2 ページ目に赤字で追加書きされている部分について、それは東大卒の先輩によるものではないかと、Slack(ビジネスチャット)で聞かれたことがあるのですが「違います。」自分で追加書きしたものです。資料をバージョンアップするタイミングまで赤字で記入しておけば、毎回、印刷しなくても済むからです。

XJ6E ICF3 打合せ資料 11/26

MC-CPU(1)-ICF3-LD015(0) 98.11.25 ヒラヤマ /
0 汎用四則演算器ｹﾞｰﾄ数見直し
b=2ケースのﾓﾝｺﾞﾒﾘ乗算 4ﾋﾞｯﾄ版追加
b=2^{128}ケースのﾓﾝｺﾞﾒﾘ乗算追加
RAS 込みのｹﾞｰﾄ数追加

XJ6E ICF3 RSA 演算方式の検討

1. 目的

XJ6E ICF3 の RSA 演算方式を選択するため、2 つのｱﾙｺﾞﾘｽﾞﾑを挙げ、その解析を行なう。

2. ｱﾙｺﾞﾘｽﾞﾑﾚﾍﾞﾙでの解析

次の 2 つのｱﾙｺﾞﾘｽﾞﾑの解析を行なう。

- 剰余定理による方法
- ﾓﾝｺﾞﾒﾘ乗算による方法

2-1 ｱﾙｺﾞﾘｽﾞﾑ概説

(1) 剰余定理による方法

RSAの暗号演算は、暗号化も復号化も剰余演算($y=g^x \bmod p$: g,x,pは1024bit)の計算を行えばよく、これを高速に行なう方式についてｱﾙｺﾞﾘｽﾞﾑﾚﾍﾞﾙでの解析を行なう。$g^x \bmod p$をそのまま計算すると非常に時間がかかるため剰余の定理を用いて、つぎの方法で一般的には計算される。ここでは剰余定理による方法とする。

$x = j[0]\cdot 2^0 + j[1]\cdot 2^1 + j[2]\cdot 2^2 + \cdots + j[m]\cdot 2^m$

```
y = 1;
for(i = 0 ; i<=m ; i++) {
    if( j[i] == 1) y=(y*g) mod p;
    g = (g*g) mod p;
}
```

(2) ﾓﾝｺﾞﾒﾘ乗算による方法

一般的にmodの計算(余/割算)は計算時間が長いため、通常の空間での余(割)算が乗算で行なえるﾓﾝｺﾞﾒﾘ変換を行ない、ﾓﾝｺﾞﾒﾘ空間でのﾓﾝｺﾞﾒﾘ乗算によって高速に演算を行なう。ここではﾓﾝｺﾞﾒﾘ乗算による方法とする。(下記の方法は、独自に方法を考案)

$x = j[0]\cdot 2^0 + j[1]\cdot 2^1 + j[2]\cdot 2^2 + \cdots + j[m]\cdot 2^m$

前提

```
gcb(p,b) = 1   ← bについては後述する。
R : 基数
G = (g×R) mod p;                    // gをﾓﾝｺﾞﾒﾘ空間へ
y = 1;
Y = (y×R) mod p;                    // yをﾓﾝｺﾞﾒﾘ空間へ
pinv = inv(p);                      // pの逆数p-1をpinvに代入 ← 後述b=2ケースでは不要
for(i = 0 ; i<m ; i++ ) {
    if( j[i]==1) Y = mm(p,pinv,Y,G);   // ﾓﾝｺﾞﾒﾘ乗算
    G = mm(p,pinv,G,G);                // ﾓﾝｺﾞﾒﾘ乗算
}
y = mm(p,pinv,1,Y)                  //  ﾓﾝｺﾞﾒﾘ空間から通常空間へ
```

削除しても結果は同じ

(3) ｱﾙｺﾞﾘｽﾞﾑ比較

ﾓﾝｺﾞﾒﾘ乗算による方法ではy=1,gをﾓﾝｺﾞﾒﾘ空間へ変換し、pの逆数を求める前処理と、求めたYをﾓﾝｺﾞﾒﾘ空間から通常空間へ変換する後処理が必要となる。つぎにﾙｰﾌﾟ内処理を比較すると次ような対応がとれる。

剰余定理による方法	ﾓﾝｺﾞﾒﾘ乗算による方法
y = (y*g) mod p	Y = mm(p,pinv,Y,G)
g = (g*g) mod p	G = mm(p,pinv,G,G)

g=(g*g) mod p は y=(y*g) mod p の演算で y=g のｹｰｽであり、同様に G = mm(p,pinv,G,G)は Y = mm(p,pinv,Y,G)の演算で Y=G のｹｰｽである。従って両ｱﾙｺﾞﾘｽﾞﾑの比較を(y*g) mod p の演算処理量と mm(p,pinv,Y,G)の演算処理量の比較で行なう。

2-2. mm(p,pinv,Y,G)関数(ﾓﾝｺﾞﾒﾘ乗算)

ﾓﾝｺﾞﾒﾘ乗算ｱﾙｺﾞﾘｽﾞﾑ(抜粋)

INPUT: integers $m = (m_{n-1}, \cdots, m_1, m_0)_b$, $x = (x_{n-1}, \cdots, x_1, x_0)_b$, $y = (y_{n-1}, \cdots, y_1, y_0)_b$ with $0 \leqq x,y < m$, $R=b^n$ with $\gcd(m,b)=1$, and $m' = -m^{-1} \bmod b$.

OUTPUT: $xyR^{-1} \bmod m$,

1. A←0 (Notation: $A = (a_n, a_{n-1}, \cdots, a_1, a_0)_b$)
2. For i from 0 to (n−1) do the following:
   2.1 $u_i \leftarrow (a_0 + x_i y_0)m' \bmod b$.
   2.2 $A \leftarrow (A + x_i y + u_i m)/b$.
3. If A≧m then A←A − m.
4. Return (A).

b が gcd(m,b) = 1 となる任意の数である場合、ｱﾙｺﾞﾘｽﾞﾑ中の 2.1,2.2 にある mod b 及び、/b の余(割)算が必要になり、剰余定理による方法での(y*g) mod p の演算よりも遅くなるのは明白である。
そこで $b = 2^d$ (d : 自然数)とし、ｼﾌﾄで行なえるようにする。
次の 2 つのｹｰｽでのｱﾙｺﾞﾘｽﾞﾑのｲﾝﾌﾟﾘﾒﾝﾄを示す。

(1) b = 2 , n = (x のﾋﾞｯﾄ数)の場合

a_i : A の i 番目のﾋﾞｯﾄ値を表わす。

x_i : X の i 番目のﾋﾞｯﾄ値を表わす。

y_i : Y の i 番目のﾋﾞｯﾄ値を表わす。

```
A = 0;
for(i=0 ; i<n ; i++) {
   u=a0 xor (xi and y0);
   A = (A + (xi , xi , ⋯ , xi ) and Y + (u, u, ⋯ ,  u) and m ) >> 1 ;
}
if(A >= m) A = A-m;
```

この演算に必要な演算器は、多倍長加算 2 回×1024 + 減算 1 回。

(2) $b=2^{1024}$, n = 1 の場合

$m' = -m^{-1}$ マイナスを忘れないこと

```
U = (X*Y*m^-1) & 0xffff ⋯ f;      //  0xffff ⋯ f = b -1;
A = (X*Y + U*m) >> 1024;
if(A >= m) A=A-m;
```

第0ビットは'1'固定になるので簡単

この演算に必要な演算は多倍長の乗算 3 回、加算 1 回、減算 1 回。

3. 解析結果まとめ

RSA 暗合演算の key 長を 1024bit とすると次のような表にまとめられる。

多倍長演算	剰余定理による方法	モンゴメリ乗算による方法	
		b = 2	$b = 2^{1024}$
加算	0	2 × 1024	1
減算	0	1	1
乗算	1	0	3
割算(余算)	1	0	0

剰余定理による方法とモンゴメリ乗算による方法($b=2^{1024}$)の比較を行なうため一般的な多倍長演算の処理時間をもちい定性的に評価する。各演算の処理時間は大まかにつぎのように仮定できる。

(割算) ≧ (乗算) >> (加算) = (減算)

ここで加算、減算を無視すると、両方法の優劣は割算 1 回と乗算 2 回の性能比で決まると考える。すなわち、乗算が割算に比べ 2 倍以上高速な場合に、モンゴメリ乗算による方法($b=2^{1024}$)が有利になる。モンゴメリ乗算による方法(b=2)は、専用演算器のカスタマイズにより性能がかなり変動するためここでの結論は避ける。

4. その他

4-1. 逆数の計算方法

FIPS 186 APPENDIX 4.による方法を示す。この方法を使用する場合、FIPS 186 Change Log No.1 1996 December 30 に書かれる内容に注意すること。

n の q に関する逆数計算

```
i=q ; h=n ; v=0 ; d=1 ;
do {
      t = i / h;
      x = h;
      h = i-t*x;
      i=x;
      x=d;
      d=v-t*x;
      v=x;
} while (h>0);
n^-1 = v % q;        ← 逆数
```

この方法には、多倍長の除算が存在し、1024bit 長の演算では収束に 500 回～700 回程度のループが必要である。

付録 A　$g^x \bmod p$ のｱﾙｺﾞﾘｽﾞﾑ計算量比較

前提条件

(1) g,x,p は、演算数が最も多くなる 1024bit の数

(2) 一般的に、乗算や除算は加・減算よりも演算処理時間が長いため演算量として多倍長乗算と多倍長除算について評価する。

(3) 1024bit の逆数演算は、4-1.のﾌﾟﾛｸﾞﾗﾑで 700 回のﾙｰﾌﾟで求められるとする。

(4) 剰余定理による方法　　y*g mod p :　乗算 1 回、除算 1 回

(5) ﾓﾝｺﾞﾒﾘ乗算による方法　mm()　　:　乗算 3 回

ｾｸｼｮﾝ2-1 ｱﾙｺﾞﾘｽﾞﾑ概説で掲載したﾌﾟﾛｸﾞﾗﾑにより前後処理に必要な乗算、除算が求められ、ﾙｰﾌﾟ処理は y*g mod p の演算及び、mm()の演算が最大 2048 回行われることがわかる。

	剰余定理による方法		ﾓﾝｺﾞﾒﾘ乗算による方法($b=2^{1024}$)	
	乗算回数	除算回数	乗算回数	除算回数
前処理	0	0	1400	702
ﾙｰﾌﾟ処理	2048	2048	6144	0
後処理	0	0	3	0
合計	2048	2048	7547	702

ここで、a を

a = (除算処理時間) / (乗算処理時間)　　; 除算器と乗算器の性能比

とすると、ﾓﾝｺﾞﾒﾘ乗算による方法が有利になる条件は、

$$2048 + 2048a > 7547 + 700a$$

$$1348a > 5499$$

$$a > 4.079.....$$

従って、乗算器が 5 倍以上、除算器よりも高速であれば p が毎回変化した場合でもﾓﾝｺﾞﾒﾘ乗算による方法が有利となる。

もし剰余定理による方法を専用演算器により乗算と除算の処理をﾊﾟｲﾌﾟﾗｲﾝ化し乗算処理を 0 とみなせる場合でも

$$2048a > 7547 + 700a$$

$$1348a > 7547$$

$$a > 5.599.....$$

すなわち、乗算器が 6 倍以上、除算器よりも高速であれば p が毎回変化した場合でもﾓﾝｺﾞﾒﾘ乗算による方法が有利となる。

付録 B べき剰余演算(g^x mod p) の性能・ｹﾞｰﾄ見積もり

剰余定理による方法、ﾓﾝｺﾞﾒﾘ乗算による方法(b=2,b=2^{1024})のどちらが ICF3 の RSA 演算方式として有利であるかを定量的に示すため、べき剰余演算(g^x mod p)を行なう論理ﾌﾞﾛｯｸ図を作成し、その性能・ｹﾞｰﾄ数の見積もりを行なった。(付録 C～E 参照)

(前提)

(1) ICF のﾏｼﾝｻｲｸﾙを 1MC = 6.4ns

(2) 1024bit 加算器のﾃﾞｨﾚｲは 2MC = 12.8ns

(3) 性能は最も処理時間がかかるｹｰｽ

(4) 汎用四則演算による性能は付録 A の乗算回数、除算回数にそれぞれの処理時間を乗じて 1.1 倍した値

(5) 汎用四則演算 乗算性能 70MC

(6) 汎用四則演算 除算性能 1100 MC

(7) 汎用四則演算 ｹﾞｰﾄ数見積もり : (42.2+30.7+215+57.3+137.3+33.8+42)×1.1 = 614.1kｹﾞｰﾄ

算出根拠

・ﾚｼﾞｽﾀﾌｧｲﾙ ｹﾞｰﾄ数(1024bitﾚｼﾞｽﾀ用) 42.2kｹﾞｰﾄ
使用予定ﾚｼﾞｽﾀﾌｧｲﾙ FNM1E5 × 32
FNM1E5 の DA 格子面積 : 504 × 1343 = 676872
L4K83A の DA 格子面積 : (40433-2913) × (40433-2913) - 2×156×84 = 1407724192
L4K83A のﾗﾝﾀﾞﾑｹﾞｰﾄ数換算値 : 8230000 × 0.333(実装率) = 2740590 ｹﾞｰﾄ
FNM1E5 のﾗﾝﾀﾞﾑｹﾞｰﾄ数換算値 : 2740590 × 676872 ÷ 1407724192 ≒ 1318 ｹﾞｰﾄ
使用予定ﾚｼﾞｽﾀﾌｧｲﾙｹﾞｰﾄ数合計 : 1318 × 32 = 42176 ｹﾞｰﾄ

・ﾚｼﾞｽﾀﾌｧｲﾙ ｹﾞｰﾄ数(μｺｰﾄﾞ用) 30.7kｹﾞｰﾄ
使用予定ﾚｼﾞｽﾀﾌｧｲﾙ FNM1E9 × 4
FNM1E9 の DA 格子面積 : 3164 × 1247 = 3945508
FNM1E9 のﾗﾝﾀﾞﾑｹﾞｰﾄ数換算値 : 2740590 × 3945508 ÷ 1407724192 ≒ 7682 ｹﾞｰﾄ
使用予定ﾚｼﾞｽﾀﾌｧｲﾙｹﾞｰﾄ数合計 : 7682 × 4 = 30.7kｹﾞｰﾄ

・ﾗｯﾁｹﾞｰﾄ数 30 × 1024 × 7 ≒ 215 k ｹﾞｰﾄ

・ｾﾚｸﾀｹﾞｰﾄ数(4 入力ｾﾚｸﾀ換算) 8× 1024 × 7 ≒ 57.3k ｹﾞｰﾄ

・乗算器ｹﾞｰﾄ数 : 137.3 k ｹﾞｰﾄ
Carry Save Adder ｹﾞｰﾄ数 : 15
CSAｹﾞｰﾄ数 : 15 × (1024 + 16) × 8 = 124.8k ｹﾞｰﾄ
乗算総ｹﾞｰﾄ数 : 124.8× 1.1 = 137.3k ｹﾞｰﾄ

・除算器ｹﾞｰﾄ数 : 33.8k ｹﾞｰﾄ
CSAｹﾞｰﾄ数 : 15 × 1024 × 2 = 30.7k ｹﾞｰﾄ
除算器総ｹﾞｰﾄ数 : 30.7 × 1.1 = 33.8 k ｹﾞｰﾄ

・加算器ｹﾞｰﾄ数 21k × 2 = 42kｹﾞｰﾄ

(8) 剰余専用演算器用四則演算器ｹﾞｰﾄ数見積もり (21.1+15.4+32.8+21) × 1.1 = 90.3kｹﾞｰﾄ

汎用四則演算器を持たない案(案 1,3,5) 90.3kｹﾞｰﾄを追加する。

算出根拠

・ﾚｼﾞｽﾀﾌｧｲﾙ ｹﾞｰﾄ数(1024bitﾚｼﾞｽﾀ用) 21.1kｹﾞｰﾄ
　使用予定ﾚｼﾞｽﾀﾌｧｲﾙ FNM1E5 × 16
・ﾚｼﾞｽﾀﾌｧｲﾙ ｹﾞｰﾄ数(μｺｰﾄﾞ用) 15.4kｹﾞｰﾄ
　使用予定ﾚｼﾞｽﾀﾌｧｲﾙ FNM1E9 × 2
・ﾗｯﾁｹﾞｰﾄ数 0ｋ ｹﾞｰﾄ (専用演算器のものを共有)
・ｾﾚｸﾀｹﾞｰﾄ数(4 入力ｾﾚｸﾀ換算) 8× 1024 × 4 ≒ 32.8k ｹﾞｰﾄ
・乗算器ｹﾞｰﾄ数 : 0k ｹﾞｰﾄ
・除算器ｹﾞｰﾄ数 : 0k ｹﾞｰﾄ
・加算器ｹﾞｰﾄ数 21kｹﾞｰﾄ

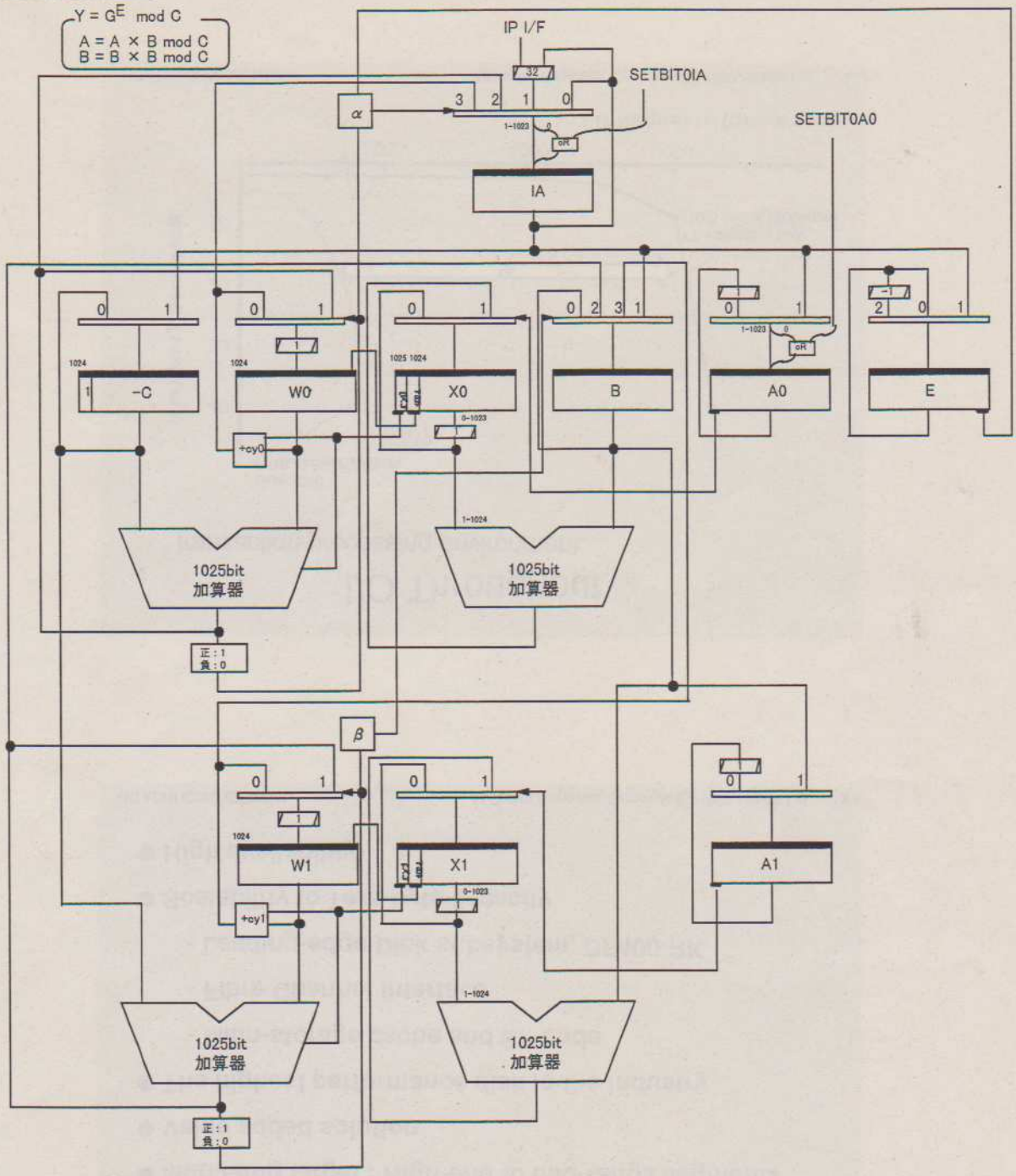
方式検討結果

	案 1	案 2	案 3	案 4	案 5	案 6
ｱﾙｺﾞﾘｽﾞﾑ	剰余定理	剰余定理	ﾓﾝｺﾞﾒﾘ乗算 (b=2) 1ﾋﾞｯﾄ版	ﾓﾝｺﾞﾒﾘ乗算 ($b=2^{1024}$)	ﾓﾝｺﾞﾒﾘ乗算 (b=2) 4ﾋﾞｯﾄ版	ﾓﾝｺﾞﾒﾘ乗算 ($b=2^{128}$)
演算ﾊｰﾄﾞ	剰余専用	汎用四則	ﾓﾝｺﾞﾒﾘ乗算	汎用四則	ﾓﾝｺﾞﾒﾘ乗算	ﾓﾝｺﾞﾒﾘ乗算
ｹﾞｰﾄ数	623.1k	614.1k	652.2k	614.1k	882.2	684.8k
総ｹﾞｰﾄ数(RAS)	1206.1k	1148.0k	1264.3k	1148.0k	1724.3k	1289.4k
逆数なし性能	0.02688 秒	0.01687 秒	0.00337 秒	0.00303 秒	0.00170 秒	0.00097 秒
逆数あり性能	0.02688 秒	0.01687 秒	0.00337 秒	0.00916 秒	0.00170 秒	0.00654 秒
ﾄｰﾀﾙ性能	0.02688 秒	0.01687 秒	0.00337 秒	0.00610 秒	0.00170 秒	0.00376 秒

注) ﾄｰﾀﾙ性能値は、逆数計算なしの処理時間と逆数計算ありの処理時間の平均値。

付録 C

剰余演算器 Rev.0



ｹﾞｰﾄ数見積もり

1024ﾋﾞｯﾄﾚｼﾞｽﾀ : 30 × 1024 × 10 = 307.2kｹﾞｰﾄ
1024ﾋﾞｯﾄ加算器 : 21k × 4 = 84kｹﾞｰﾄ
1024ﾋﾞｯﾄｲﾝｸﾘﾒﾝﾀ : 21k × 2 = 42kｹﾞｰﾄ

4入力ｾﾚｸﾀ : 8 × 1024 × 2 = 16.4kｹﾞｰﾄ
3入力ｾﾚｸﾀ : 6 × 1024 × 1 = 6.1kｹﾞｰﾄ
2入力ｾﾚｸﾀ : 4 × 1024 × 7 = 28.7kｹﾞｰﾄ

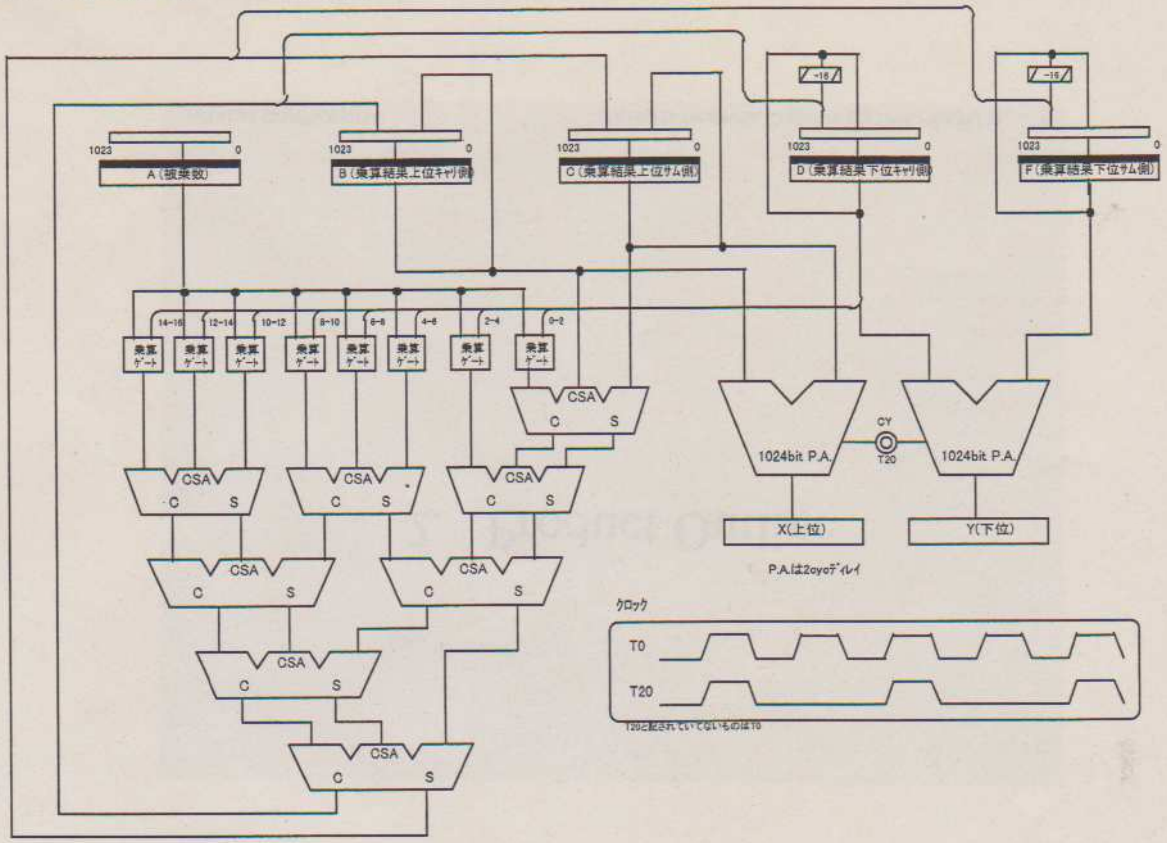
484.4kｹﾞｰﾄ
合計 484.4k × 1.1 = 532.8kｹﾞｰﾄ

性能見積もり

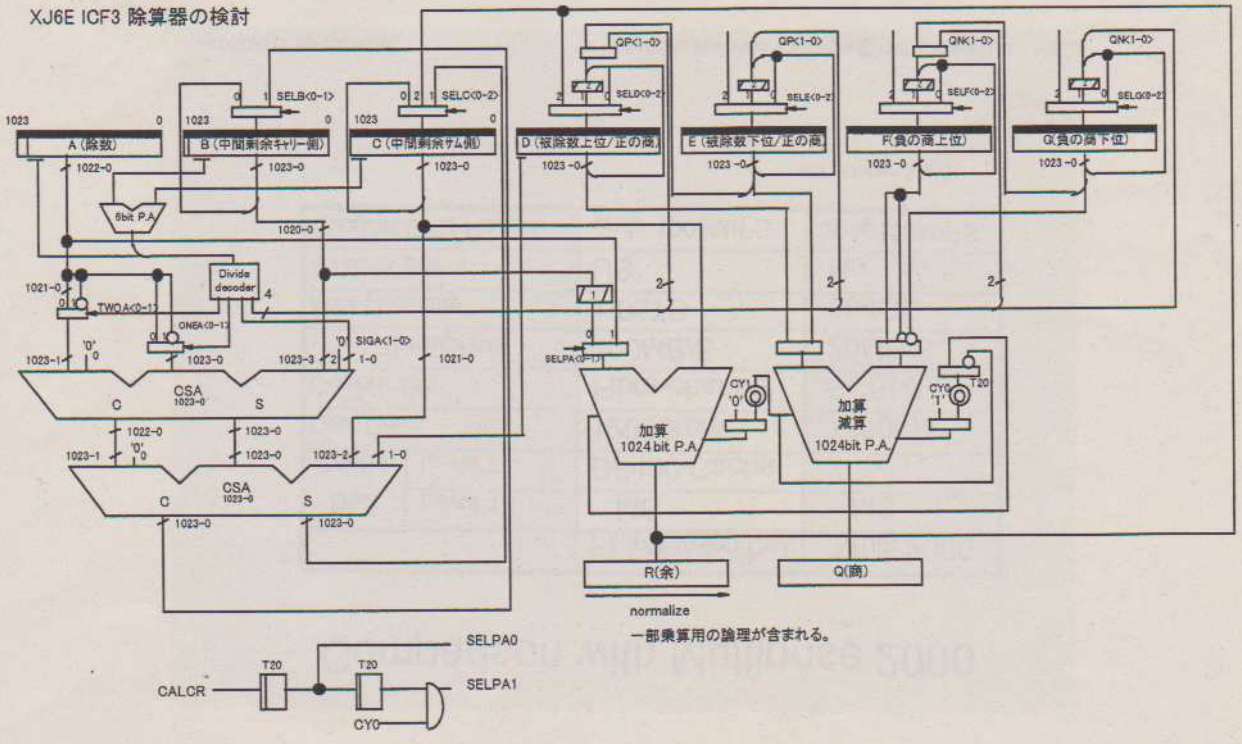
1べき乗演算 : 2051 × 1024 = 2100224 cyc
1cyc = 12.8ns とすると
1べき乗演算時間 = 2100224 × 12.8ns = 0.02688秒

付録 D

XJ6E ICF3 乗算器の検討　　2bit Booth法

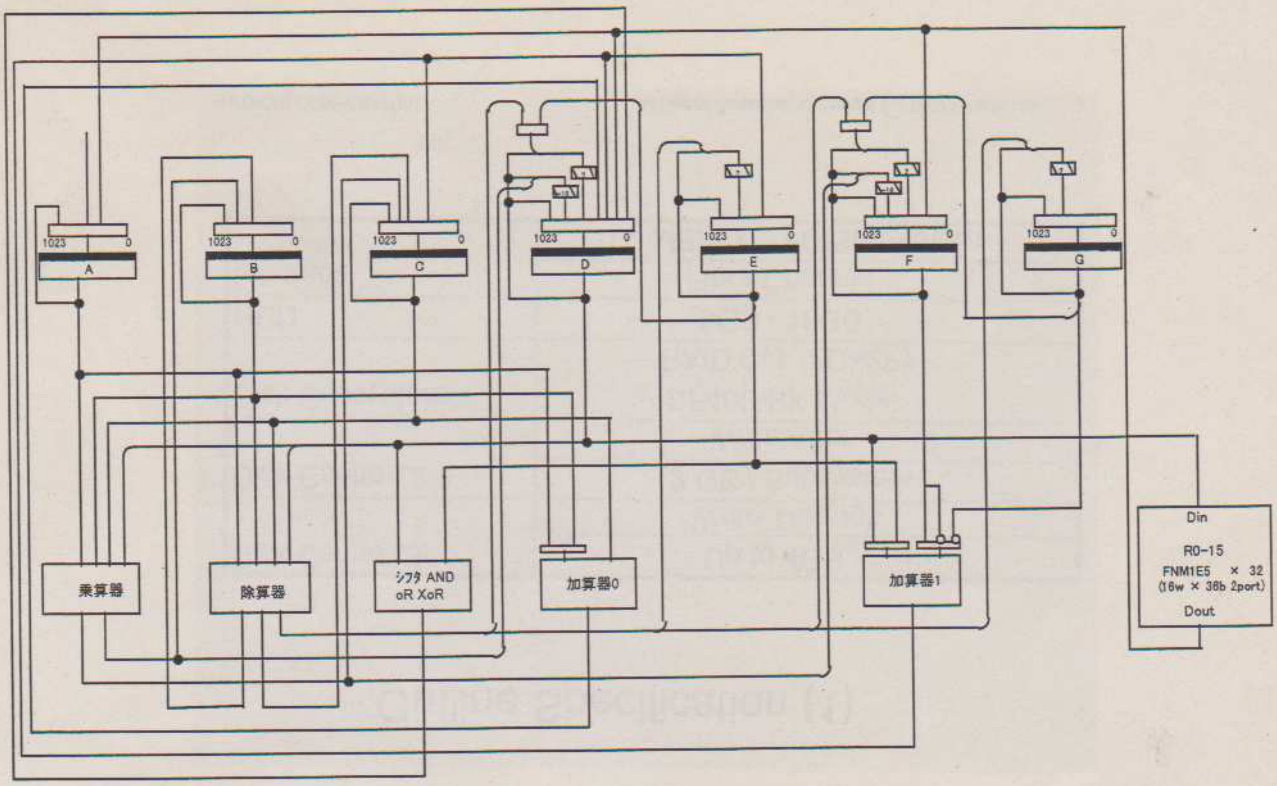


XJ6E ICF3 除算器の検討

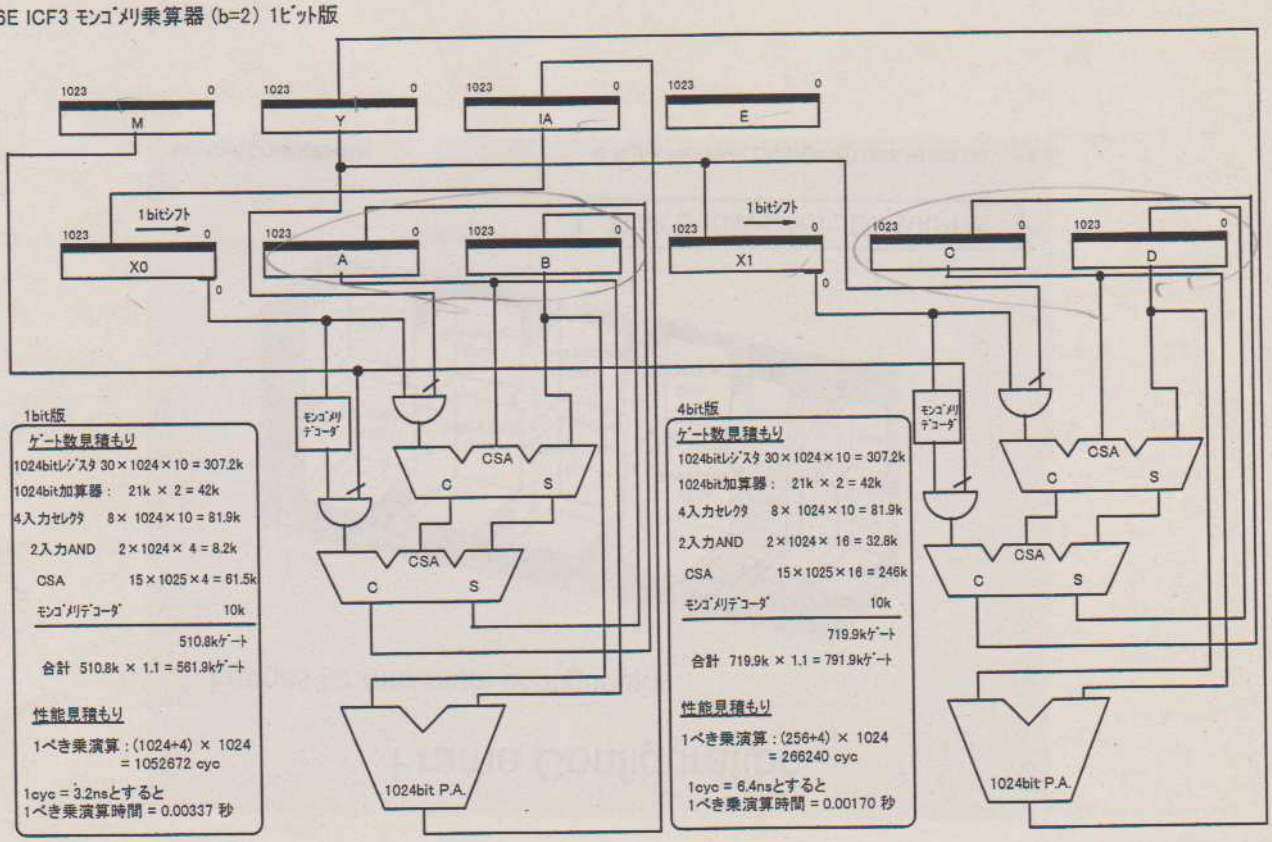


付録 E

XJ6E ICF3 汎用四則演算器



XJ6E ICF3 ﾓﾝｺﾞﾒﾘ乗算器 (b=2) 1ﾋﾞｯﾄ版



1bit版

ｹﾞｰﾄ数見積もり

1024bitﾚｼﾞｽﾀ 30×1024×10 = 307.2k
1024bit加算器： 21k × 2 = 42k
4入力ｾﾚｸﾀ 8× 1024×10 = 81.9k
2入力AND 2×1024× 4 = 8.2k
CSA 15×1025×4 = 61.5k
ﾓﾝｺﾞﾒﾘﾃﾞｺｰﾀﾞ 10k
510.8kｹﾞｰﾄ
合計 510.8k × 1.1 = 561.9kｹﾞｰﾄ

性能見積もり

1ﾍﾞき乗演算：(1024+4) × 1024
= 1052672 cyc

1cyc = 3.2nsとすると
1ﾍﾞき乗演算時間 = 0.00337 秒

4bit版

ｹﾞｰﾄ数見積もり

1024bitﾚｼﾞｽﾀ 30×1024×10 = 307.2k
1024bit加算器： 21k × 2 = 42k
4入力ｾﾚｸﾀ 8× 1024×10 = 81.9k
2入力AND 2×1024× 16 = 32.8k
CSA 15×1025×16 = 246k
ﾓﾝｺﾞﾒﾘﾃﾞｺｰﾀﾞ 10k
719.9kｹﾞｰﾄ
合計 719.9k × 1.1 = 791.9kｹﾞｰﾄ

性能見積もり

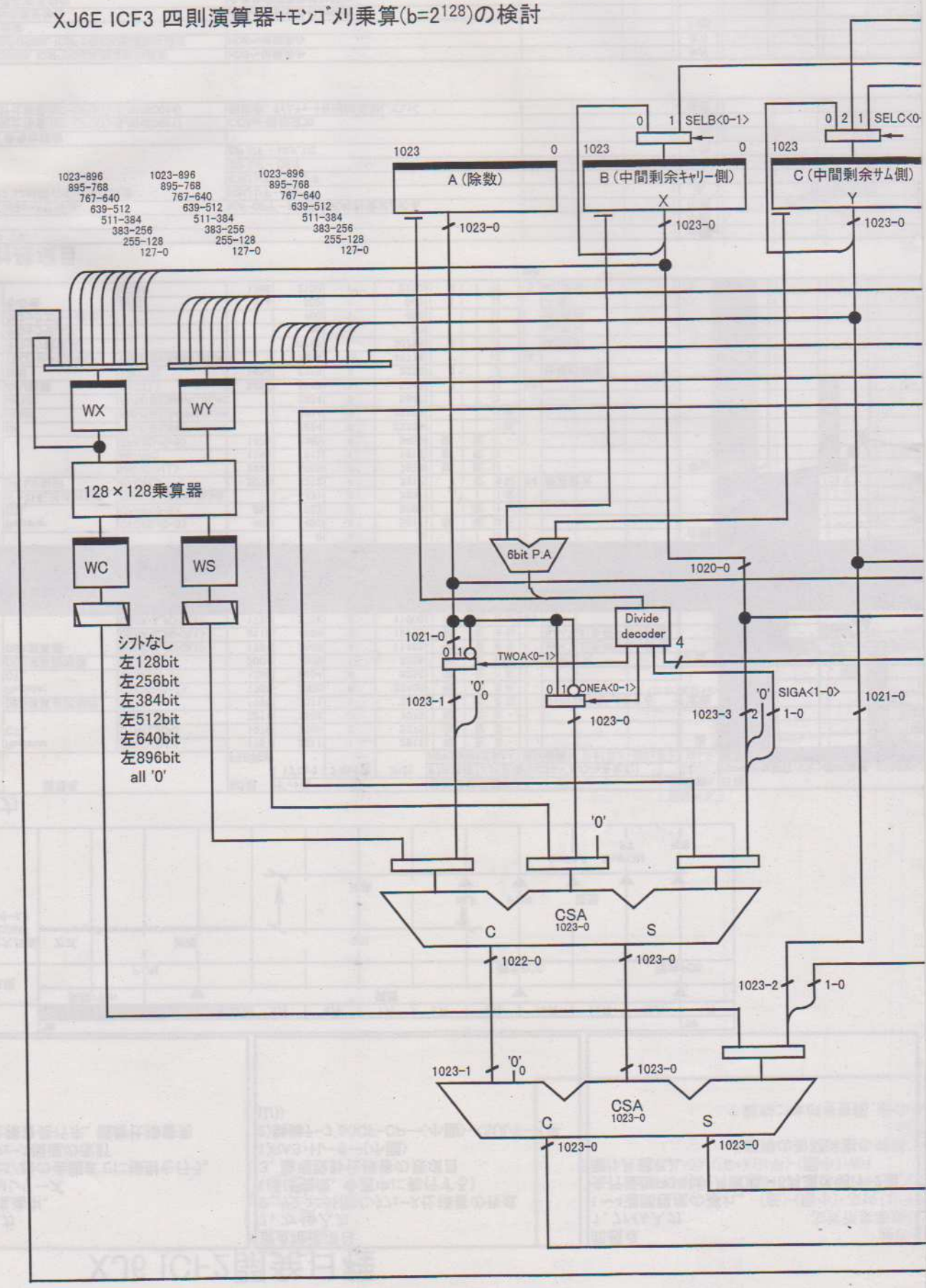
1ﾍﾞき乗演算：(256+4) × 1024
= 266240 cyc

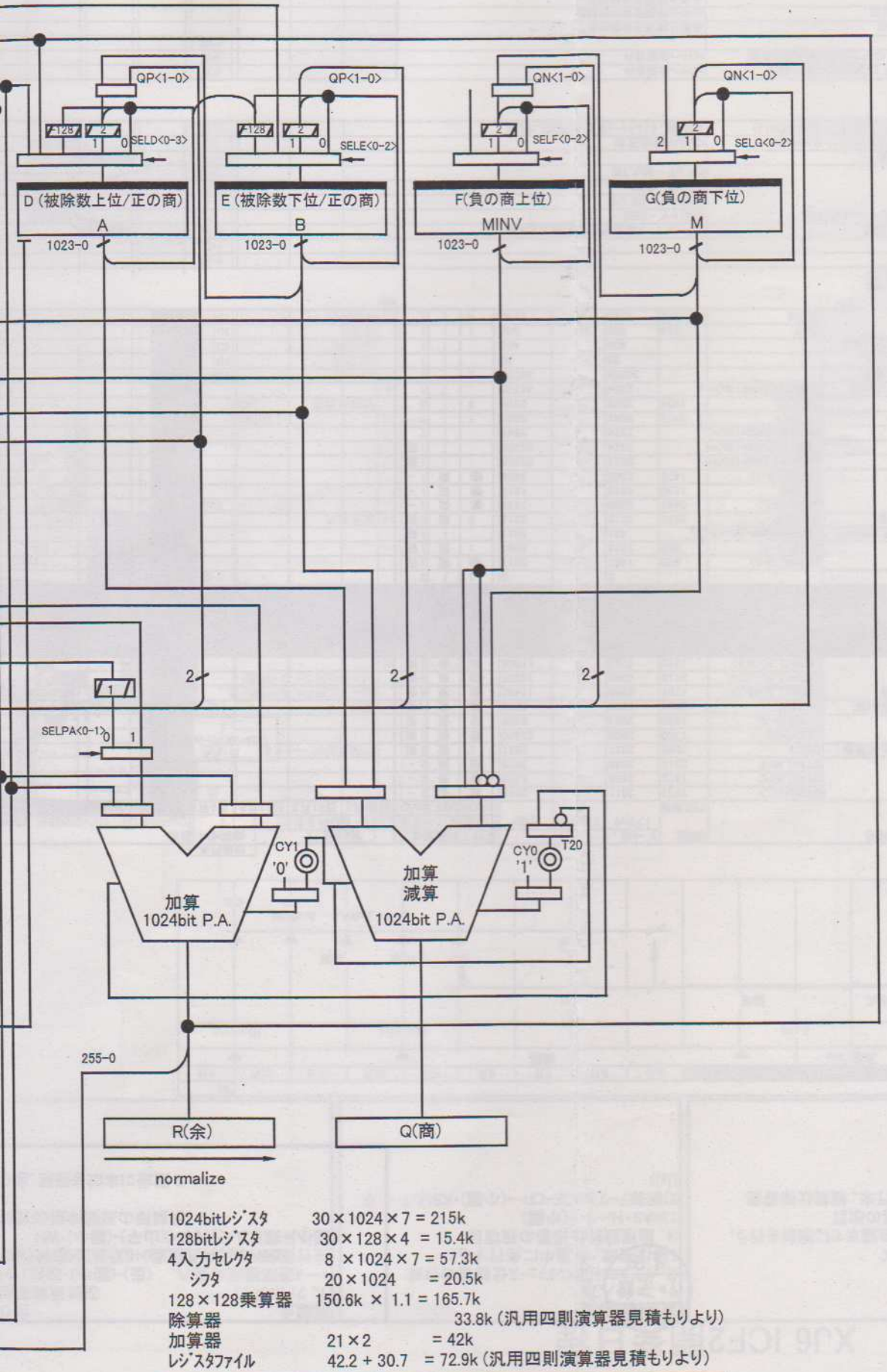
1cyc = 6.4nsとすると
1ﾍﾞき乗演算時間 = 0.00170 秒

0-7L
3.5m-fdy

付録 F

XJ6E ICF3 四則演算器+ﾓﾝｺﾞﾒﾘ乗算(b=2^{128})の検討





1024bitレジスタ	30×1024×7 = 215k	
128bitレジスタ	30×128×4 = 15.4k	
4入力セレクタ	8 ×1024×7 = 57.3k	
シフタ	20×1024 = 20.5k	
128×128乗算器	150.6k × 1.1 = 165.7k	
除算器		33.8k (汎用四則演算器見積もりより)
加算器	21×2 = 42k	
レジスタファイル	42.2 + 30.7 = 72.9k (汎用四則演算器見積もりより)	

622.6kゲート

合計 633.6 × 1.1 = 684.8kゲート

10/E