卒　業　論　文

擬似乱数発生器WELLのハードウェア化と評価

氏名　　： **新井 惣一朗**
学籍番号： 2260040003-0
指導教員： **山崎 勝弘 教授**

提出日　： 2008 年 2 月 20 日

立命館大学　理工学部　電子情報デザイン学科

内容梗概

本論文では、Verilog-HDL を用いた、擬似乱数発生器 WELL のハードウェア化について述べる。WELL には周期の違うものがいくつかあり、その中で $2^{512}-1$ 周期、$2^{1024}-1$ 周期、$2^{19937}-1$ 周期、$2^{44497}-1$ 周期の 4 つを Verilog-HDL で設計し、シミュレーションを行った。そして 4 つそれぞれのシミュレーション結果が、四捨五入による多少の誤差はあるが、C プログラムで生成した乱数とほぼ同じ値が出ることを確認した。

また、WELL の回路規模を検証し、最大動作周波数から乱数生成時間を求め、C プログラムでの乱数生成時間との比較を行った。WELL をハードウェア化することにより $2^{512}-1$ 周期と $2^{1024}-1$ 周期はソフトウェアの約 5000 倍，$2^{19937}-1$ 周期は $2^{44497}-1$ 周期は約 3350 倍の速度向上が得られた。

目次

図目次

表目次

1　はじめに

乱数は、情報科学や TV ゲーム業界ではもちろん、科学全般、社会学、経済学など、広い分野で利用されている。乱数には擬似乱数と擬似ではない真の乱数が存在する。

真の乱数は原子核崩壊時の雑音・大気中の雑音・電子回路の熱雑音などの外的要因を元のデータとしてビット列を生成するため、次に出現するビット列の予測が不可能であるという利点があるが、再び同じ乱数列を利用することが不可能という欠点もある。

逆に擬似乱数はある特定のアルゴリズムにより次々と計算を行いビット列を生成していくため、同じ乱数列を利用することが可能であり、また動作が高速であることが多く、真の乱数よりも偏りが小さいなどの利点があるが、特定のアルゴリズムにより乱数を生成しているため、生成される乱数列にはある規則性ができてしまうという欠点がある。

世の中で使用されている乱数の殆どが擬似乱数ある。しかしコンピュータ内で擬似乱数を使用すると、メモリの中を見られ、入力値や計算途中の値を知られ、擬似乱数の規則性を解析される可能性がある。よって擬似乱数を暗号やセキュリティにそのまま使うのは危険である。このため、擬似乱数生成アルゴリズムをハードウェア化して得られる、規則性の解析が困難な擬似乱数が必要である。

以上のような背景を踏まえ、本研究では Veirlog-HDL を用いて擬似乱数生成アルゴリズム WELL のハードウェア化を行う。これにより、同じ初期値を与えれば同じ乱数列を得られるという擬似乱数の利点を持ちながら、規則性を解析される可能性がかなり低くなる。さらにハードウェア化することで生成速度が、ソフトウェアで擬似乱数を生成するよりも向上するという利点も持つようになる。

WELL は乱数生成速度が速く、また一般的によく用いられる擬似乱数であるメルセンヌツイスタよりも周期が長く、乱数の偏りも少ないという特徴がある。WELL ののハードウェア化はインターネット上に公開されている WELL の論文[1]と C 言語で記述された WELL のプログラム[2]から行う。またその WELL のプログラムには WELL512a、WELL1024a、WELL19937a、WELL4449a の 4 種類があり、それらは乱数の周期が違うものである。本研究では、それぞれの WELL を Verilog-HDL で記述し、シミュレーションを行い、C 言語のプログラムと同じ出力結果となっていることを確認する。また、ハードウェア化したときの回路規模の大きさ及び、動作周波数を確認する。そして動作周波数と乱数生成に必要なクロック数から、乱数生成速度を求める。その乱数生成速度と C 言語で記述された WELL のプログラムを実行したときの実行時間との比較と評価を行う。

本論文では、第 2 章で WELL の概要とアルゴリズムについて述べる。第 3 章では $2^{512}-1$ 周期、$2^{1024}-1$ 周期、$2^{19937}-1$ 周期、$2^{44497}-1$ 周期の 4 種類の WELL の設計方法を述べる。第 4 章では設計した WELL の論理合成結果の評価、FPGA へ実装、考察をする。第 5 章では現在までの本研究の成果と今後の課題について述べる。

2　擬似乱数発生器 WELL

2.1　WELL の概要

WELL（Well Equidistributed Long-period Linear）は、2004 年に FRANCOIS PANNETON、PIERRE L’ECUYER、松本眞によって発表された擬似乱数生成アルゴリズムである。擬似乱数は生成に規則性を持っているため、できるだけ生成される数値の予測を困難に、かつ生成される数値の偏りを小さくする必要がある。また、全ての擬似乱数は周期を持ち、1 周期ごとに同じ数列を繰り返し生成する性質を持つ。そのため周期は長ければ長いほど優れた擬似乱数と言える。以上のようなことを踏まえて WELL には以下のような長所がある。

・最大周期が $2^{44497}-1$ と非常に長い周期。
・1390 次元超立方体に均等分布する。
・生成される乱数の偏りが小さい。
・計算量は多いが、乗算と除算が殆どないため、乱数の生成速度が速い。
・以前の擬似乱数アルゴリズムは優れたものでも、最初の数十万個は均等に分散をしないという欠点があったが、それも改善されている。

また WELL はワーキングメモリと計算の値を少し変えて周期を小さくすることもでき、$2^{44497}-1$ 周期の他に、$2^{23209}-1$ 周期、$2^{21701}-1$ 周期、$2^{19937}-1$ 周期、$2^{1024}-1$ 周期、$2^{800}-1$ 周期、$2^{607}-1$ 周期、$2^{521}-1$ 周期、$2^{512}-1$ 周期などがある。本研究では $2^{512}-1$ 周期、$2^{1024}-1$ 周期、$2^{19937}-1$ 周期、$2^{44497}-1$ 周期の 4 種類の WELL のハードウェア化を行っている。

2.2　WELL のアルゴリズム

WELL のアルゴリズムのフローチャートを図 1 に示す。

32 ビットの 0 以上の整数 (ベクトル)**v** を r 個持っている状態ベクトル **x** を次のように表す。

$$\mathbf{x}_i = (\mathbf{v}_{i,0}, \mathbf{v}_{i,1}, \mathbf{v}_{i,2}, \ldots, \mathbf{v}_{i,r-2}, \mathbf{v}_{i,r-1})$$

i = 0 の時を $\mathbf{x}_i$ の最初の値とし、 $\mathbf{x}_0$ の初期値として $\mathbf{x}_0$ の中の $\mathbf{v}_{0,0}$〜$\mathbf{v}_{0,r-1}$ に、それぞれ 32 ビットまでの値を入れる。$\mathbf{x}_i$ に行列 **A** の演算を行った値が $\mathbf{x}_{i+1}$ の値となる。そして出力を **y** とすると、状態 i の時の出力は $\mathbf{v}_{i+1,0}\times$ 0.232830643e-10 を出力する。これを乱数の出力したい数だけ繰り返す。図 1 のフローチャートでは乱数を 10000 個出力している。また $\mathbf{v}_{i+1,0}$ は 32 ビットの自然数なので、これに 0.232830643e-10 を掛けることにより乱数は 0〜1 の間で小数が均等に分布して出力される。

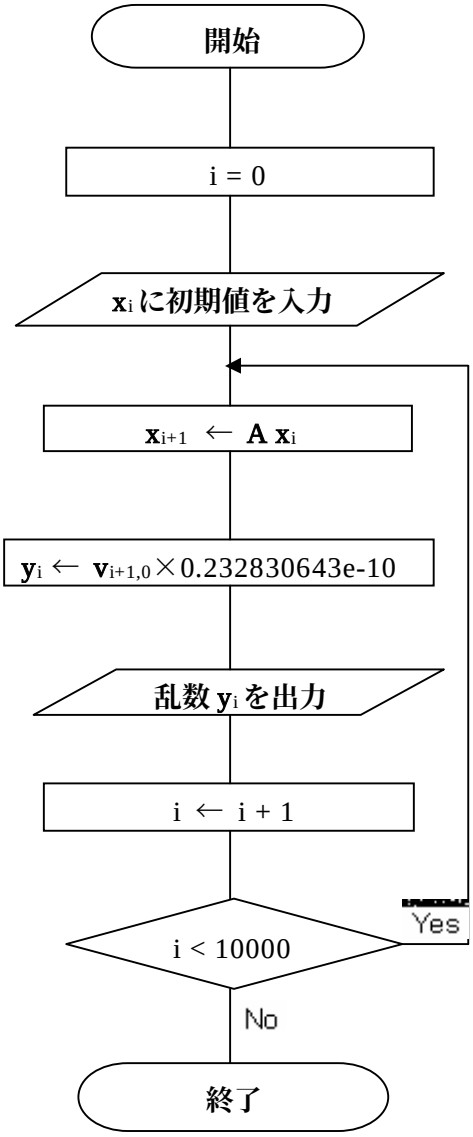
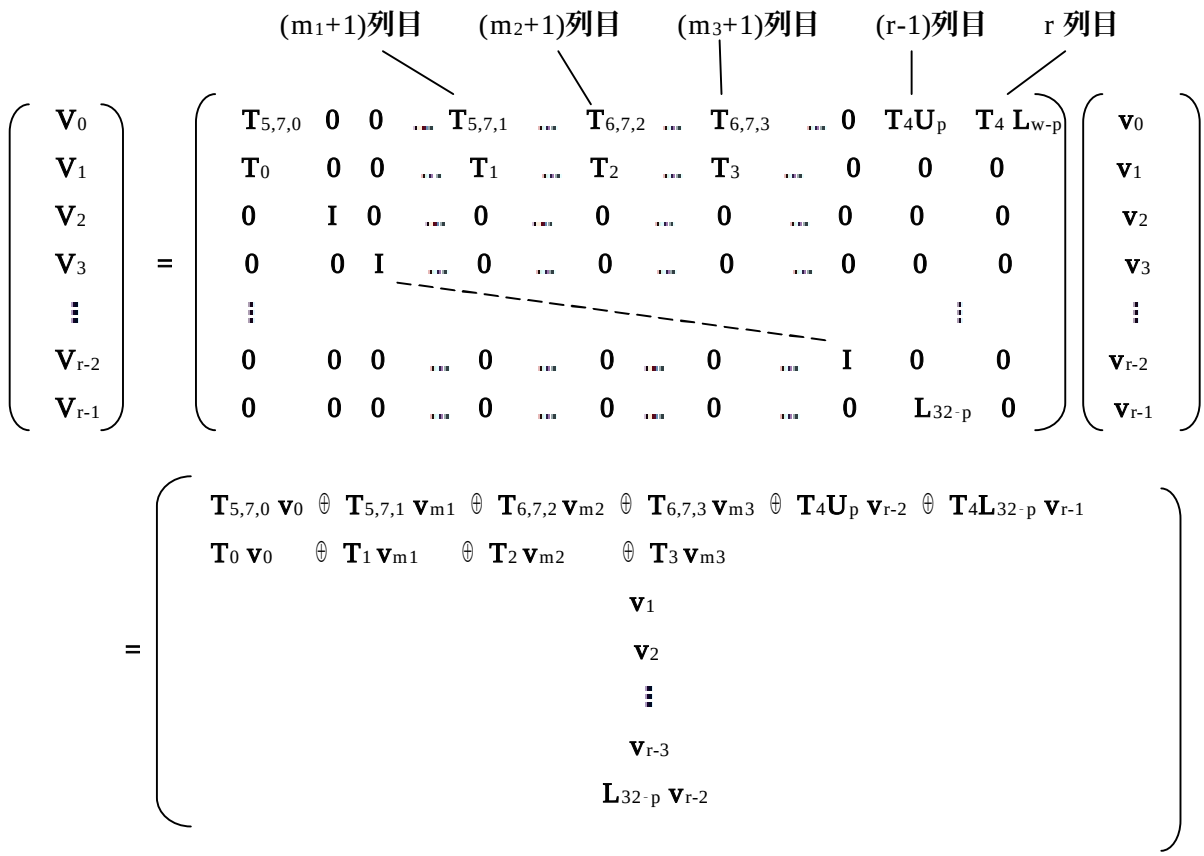


図 1：WELL のフローチャート

- 状態ベクトル $\mathbf{x}_i$　：$\mathbf{x}_i = (\mathbf{v}_{i,0},, \ldots, \mathbf{v}_{i,r-1})$
- $\mathbf{v}_{i,0}$ 〜 $\mathbf{v}_{i,r-1}$　：それぞれ 32 ビット
- r　：$\mathbf{x}_i$ の中の **v**(ベクトル)の数で最大の値は 1391
- $\mathbf{y}_i$　：状態 i の時の出力
- **A**　：32 ビットの行列を r×r 個持っている、変換行列で **x** の中の **v** の値を変化させる。

フローチャートの中の $\mathbf{x}_{i+1} = \mathbf{A}\mathbf{x}_i$ 式の内容を示したのが次の行列の計算式である。$\mathbf{x}_i$ と $\mathbf{x}_{i+1}$ の違いがわかりやすいように $\mathbf{v}_{i,0}$〜$\mathbf{v}_{i,r-1}$ を $\mathbf{v}_0$〜$\mathbf{v}_{r-1}$ と置き、$\mathbf{v}_{i+1,0}$〜$\mathbf{v}_{i+1,r-1}$ を $\mathbf{V}_0$〜$\mathbf{V}_{r-1}$ と置いている。



- p　：**vr-1 の下位 p ビットは使用しないため、常に 0。**
- m1, m2, m3　：**0 < m1, m2, m3 < r の定数**
- **T0 ～T7**　：**32×32 の 2 進数の変換行列で表 1 の M0 ～M6 のどれかの演算を行って v の値を変化させる。WELL の周期によってどの演算を行うかは異なる。**
- **Ta,b,c**　：**Ta Tc ⊕ Tb Tc（例, T5,7,0 = T5 T0 ⊕ T7 T0 ）**
- **Up**　：$\begin{pmatrix} 0 & 0 \\ 0 & I_p \end{pmatrix}$
- **L32-p**　：$\begin{pmatrix} L_{32-p} & 0 \\ 0 & 0 \end{pmatrix}$
- **I**　：32×32 **の単位行列（= E）**
- **Ip**　：p×p **の単位行列**
- **I32-p**　：(32-p)×(32-p) **の単位行列**
- ⊕　：EXOR

前の行列の計算式を簡単に表したものを図 2 に示す。図 2 に演算を行った後に $y_i = V_0 \times 0.232830643$ を出力する。

$z_0 \leftarrow (m_p \,\&\, v_{r-1}) \oplus (!m_p \,\&\, v_{r-2})$

$z_1 \leftarrow T_0 v_0 \oplus T_1 v_{m1};$

$z_2 \leftarrow T_2 v_{m2} \oplus T_3 v_{m3};$

$V_{r-1} \leftarrow v_{r-2} \,\&\, m_p$

for $j = r-2, \ldots, 2,$ do $V_j \leftarrow v_{j-1};$

$V_1 \leftarrow z_1 \oplus z_2;$

$V_0 \leftarrow T_4 z_0 \oplus T_5 z_0 \oplus T_6 z_2 \oplus T_7 (z_1 \oplus z_2).$

- m_p ： v_{r-1} の上位 32-p ビットを取り出す 32 ビットのビットマスク
- $!m_p$ ： m_p の補数で v_{r-2} の下位 p ビットを取り出すのビットマスク
- z_0~z_2 ： 値を一時格納するための変数

図 2：WELL のアルゴリズム[1]

表 1 に T_0 ～T_7 に使われる演算 M_0～M_6 を示す。T_0 ～T_7 に M_0～M_6 のとれを使用するかは周期の違う WELL によって変わり、どの演算にどのような計算値をいれるかは WELL の論文[1]と C プログラム[2]を参照している。

表 1：変換行列 T_0 ～T_7 に使われる演算[1]

変換行列 M	y = Mv の実行　v：(v_0~ v_{31})の 32 ビット
M_0	$y = 0$
M_1	$y = v$
$M_2(t)$ $-32 \leqq t \leqq 32$	$y = (v >> t)$　$t \geqq 0$ $= (v << -t)$　$t < 0$
(t) $-32 \leqq t \leqq 32$	$y = v \oplus (v >> t)$　$t \geqq 0$ $= v \oplus (v << -t)$　$t < 0$
$M_4(a)$ a：32 ビット	$y = (v >> 1) \oplus a$　if　$v_{31} = 1$ $= (v >> 1)$　else
$M_5(t, b)$ $-32 \leqq t \leqq 32$ b：32 ビット	$y = v \oplus ((v << t) \,\&\, b)$　$t \geqq 0$ $= v \oplus ((v >> -t) \,\&\, b)$　$t < 0$
$M_6(q, s, t, a)$ $-32 \leqq q, s, t \leqq 32$ a：32 ビット	$y = (((v << q) \oplus (v >> (w - q))) \,\&\, d_s) \oplus a$　if $v_t = 1$ $= (((v << q) \oplus (v >> (w - q))) \,\&\, d_s)$　else

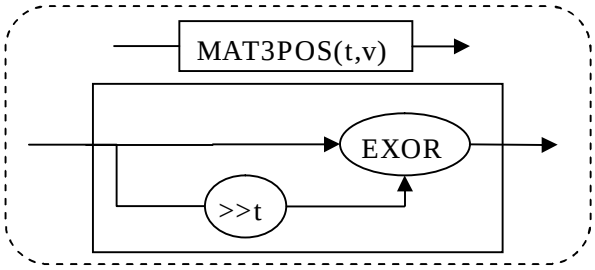
d_s：32 ビットで(s+1)番目の値が 0、他のすべては 1

3　WELL の設計

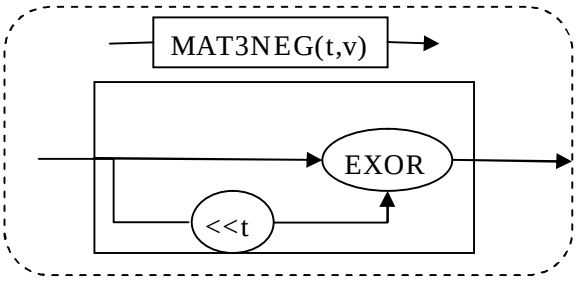
3.1　$2^{512}-1$ 周期 WELL と $2^{1024}-1$ 周期 WELL の設計

ハードウェア化における WELL のモジュール内で使用しているファンクションモジュールを図 3 に示す。図 3 では点線の中の上部にモジュール名を、そしてその構造を下部に示している。また図 3 は表 1 の $\mathbf{M}_2$, $\mathbf{M}_3$, $\mathbf{M}_5$, $\mathbf{M}_6$, 及び乱数出力前の 0. 232830643e-10 を掛けるという演算をモジュール化している。$\mathbf{M}_0$ と $\mathbf{M}_1$ は特に演算が必要ないためモジュール化していない。$\mathbf{M}_4$ は今回使用していない。(h)では 0. 232830643e-10 を掛ける演算の代わりに 232830643 を掛ける演算を行っている。これは Verilog-HDL では小数をそのまま用いることができないため、小数点以下の値を整数に直して用いているからである。また演算は乗算器をを使用せずにビットシフトと加算で行っている。

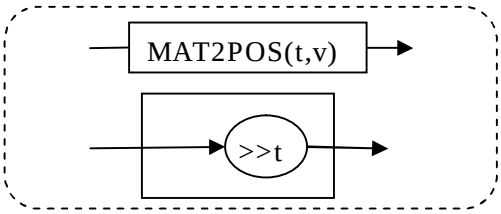
また入力値を v としている。



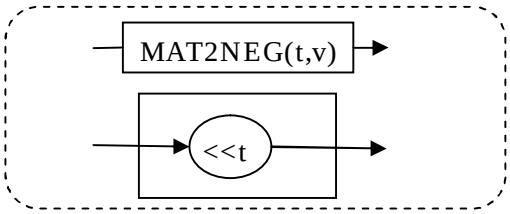
(a)：t >0 の時の $\mathbf{M}_3$



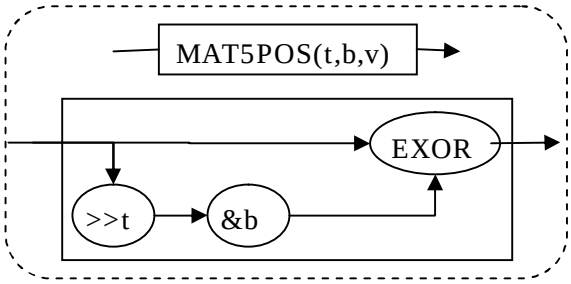
(b)：t<0 の時の $\mathbf{M}_3$



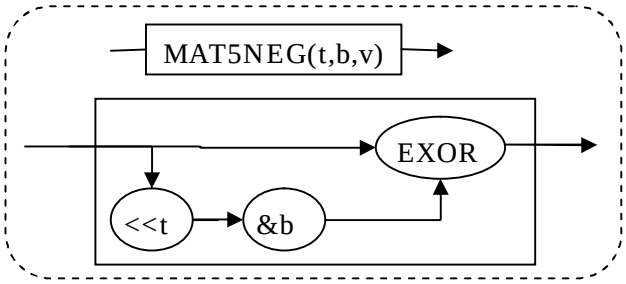
(c)：t >0 の時の $\mathbf{M}_2$



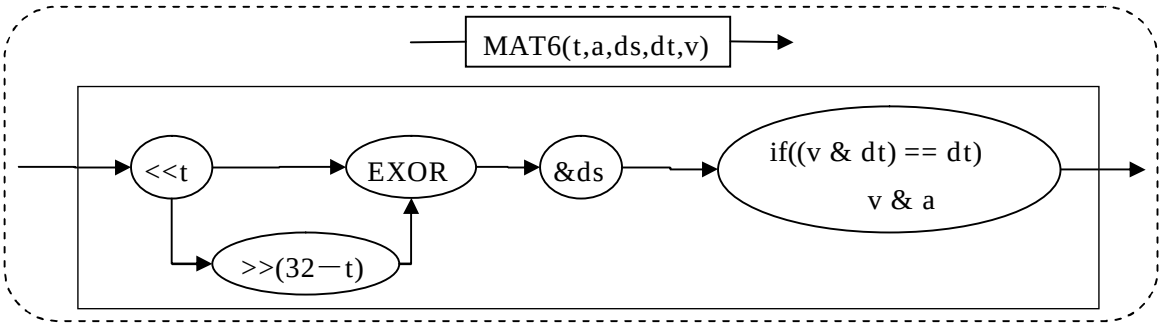
(d)：t<0 の時の $\mathbf{M}_2$



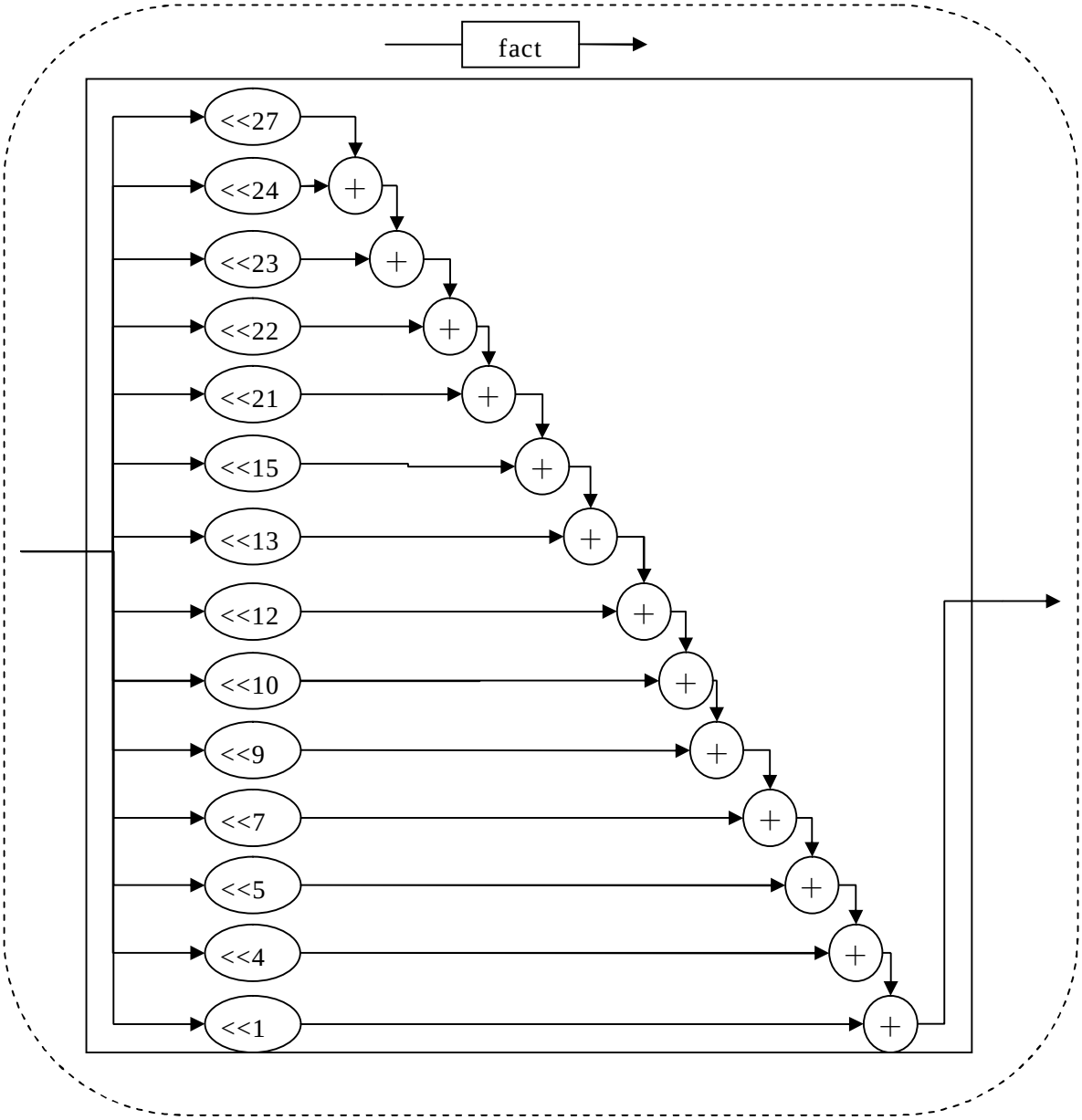
(e)：t >0 の時の $\mathbf{M}_5$



(f)：t<0 の時の $\mathbf{M}_5$



(g)：$\mathbf{M}_6$



(h)：232830643 を掛ける演算

図 3：ファンクションモジュール(a) ～ (h)

次に $2^{512}-1$ 周期 WELL のモジュール構成を図 4 に示す。また演算の種類と計算値は次のようになっている。

v の数 r、v_{r-1} の使わない下位 p ビット、v_{m1}, v_{m2}, v_{m3} の値；

r = 16,　p = 0,　$v_{m1} = v_{13}$,　$v_{m2} = v_9$,　$v_{m3} = v_5$

途中の演算である T_0 ～T_7；

$T_0 = M_3(-16)$,　$T_1 = M_3(-15)$,　$T_2 = M_3(11)$,　$T_3 = M_0$,

$T_4 = M_3(-2)$,　$T_5 = M_3(-18)$,　$T_6 = M_2(28)$,　$T_7 = M_5(-5, 0xda442d24)$,

v_0～v_{15} = STATE[0]～STATE[15]

また周期を 2^k-1 とすると k は $k = 32\times r-p$ **で表される。よって** $k = 32\times 16-0 = 512$ **となり、周期が $2^{512}-1$ となる。この周期の－1 というのは v_0～v_{r-1} 全ての初期値が 0 の時は乱数が 0 にしかならないため、その分を引いたものである。**

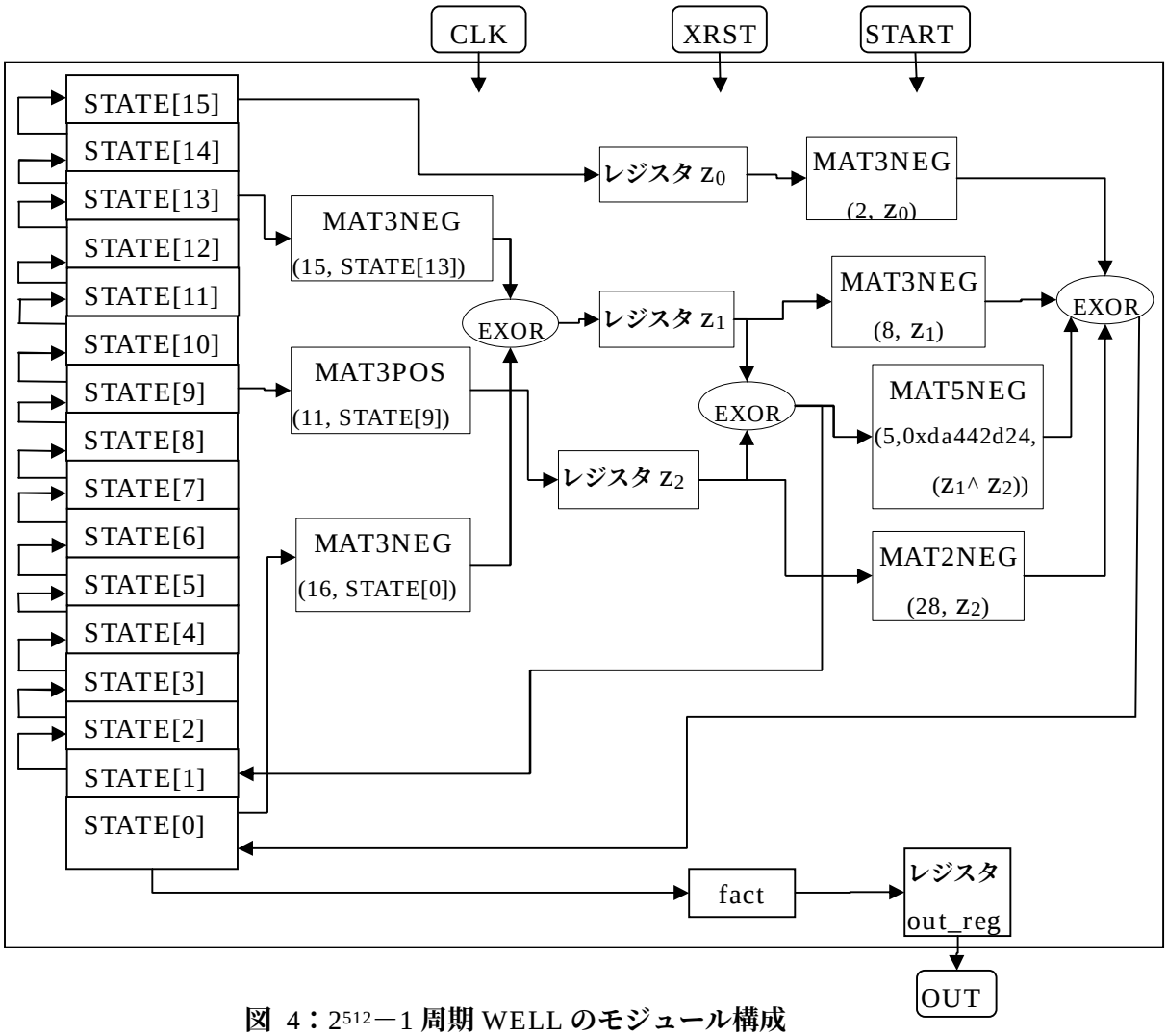


図 4：$2^{512}-1$ 周期 WELL のモジュール構成

$2^{512}-1$ 周期 WELL モジュールの状態遷移を表 2 に示す。

表 2：$2^{512}-1$ 周期 WELL モジュールの状態遷移

状態	処理	次の状態
0	XRST でリセットし、STATE[0]~STATE[15]に初期値を入れる。 START が立ち上がるまで状態 0 で待つ。START が立ち上がると状態 1 へ進む。	1
1	z_0= STATE[15] z_1 = MAT3NEG(16, STATE[0]) ^ MAT3004EEG(15, STATE[13]); z_2 = MAT3POS(11, STATE[9]);	2
2	STATE[15] = STATE[14]; • • • STATE[2] = STATE[1]; STATE[1] = z_1 ^ z_2; STATE[0]=MAT3NEG(2,z_0) ^ MAT3NEG(8,z_1) ^MAT2NEG(28,z_2) ^ MAT5NEG(5,0xda442d24,(z1 ^ z2));	1, 3
3	out_reg = fact(STATE[0]) , out_reg を OUT から出力。	

START が立ち上がった後の 1 クロック目でファンクションモジュールと EXOR で決められた STATE の値に演算を行い、z_0～z_2 に代入する。そして 2 クロック目で STATE[15]～STATE[2]に STATE[14]～STATE[1]を 1 つずつずらして代入し、さらに z_0～z_2 の値を EXOR で演算を行い、その値を STATE[1]と STATE[0]に代入する。そして最後に 3 クロック目で STATE[0]の値を fact モジュールで演算を行い、out_reg に代入して乱数として出力する。またそれと同時に状態 1 の処理も行い、今の状態 1～3 の作業を繰り返す。これにより最初の乱数は生成するのに 3 クロックかかるが、次に乱数からは 2 クロックで生成される。

C のプログラムで乱数を出力した結果と Verilog-HDL で設計した $2^{512}-1$ 周期 WELL のシミュレーション結果を表 3 に示す。シミュレーション結果は最高 18 桁で表示されるが、表 3 では下位の 10 桁を省き、最高 8 桁の整数を示している。

表 3：$2^{512}-1$ 周期 WELL の C プログラムと Model-Sim での乱数出力結果

	C のプログラムの実行結果	Model-Sim のシミュレーション結果
乱数 1 個目	0.56427003	56427003
乱数 2 個目	0.53454621	53454620
乱数 3 個目	0.98663546	98663545
乱数 4 個目	0.87725855	87725855
乱数 5 個目	0.81463648	81463647
乱数 6 個目	0.75354035	75354034

初期値はシンプルに STATE[0] = 0 ～STATE[15] = 15 の順番で代入した。表 3 の結果を見ると、四捨五入による多少の誤差はあるものの、C の結果の小数点以下 8 桁とシミュレーションの結果の最初の 8 桁はほぼ一致していることが分かる。なおシミュレーションは Model-Sim XEⅢ6.2g で行っている。

次に $2^{1024}-1$ 周期 WELL のモジュール構成を図 5 に示す。また演算の種類と計算値は次のようになっている。

v の数 r、$\mathbf{v}_{r-1}$ の使わない下位 p ビット、$\mathbf{v}_{m1}$, $\mathbf{v}_{m2}$, $\mathbf{v}_{m3}$ の値；

r = 31, p = 0, $\mathbf{v}_{m1} = \mathbf{v}_3$, $\mathbf{v}_{m2} = \mathbf{v}_{24}$, $\mathbf{v}_{m3} = \mathbf{v}_{10}$

途中の演算である $\mathbf{T}_0$ ～$\mathbf{T}_7$；

$\mathbf{T}_0 = \mathbf{M}_1$, $\mathbf{T}_1 = \mathbf{M}_3$ (8) , $\mathbf{T}_2 = \mathbf{M}_3$ (-19) , $\mathbf{T}_3 = \mathbf{M}_3$ (-14),

$\mathbf{T}_4 = \mathbf{M}_3$ (-11), $\mathbf{T}_5 = \mathbf{M}_3$ (-7), $\mathbf{T}_6 = \mathbf{M}_3$ (-13), $\mathbf{T}_7 = \mathbf{M}_0$,

$\mathbf{v}_0$～$\mathbf{v}_{31}$ = STATE[0]～STATE[31]

周期の指数 k = 32×32－0 = 1024

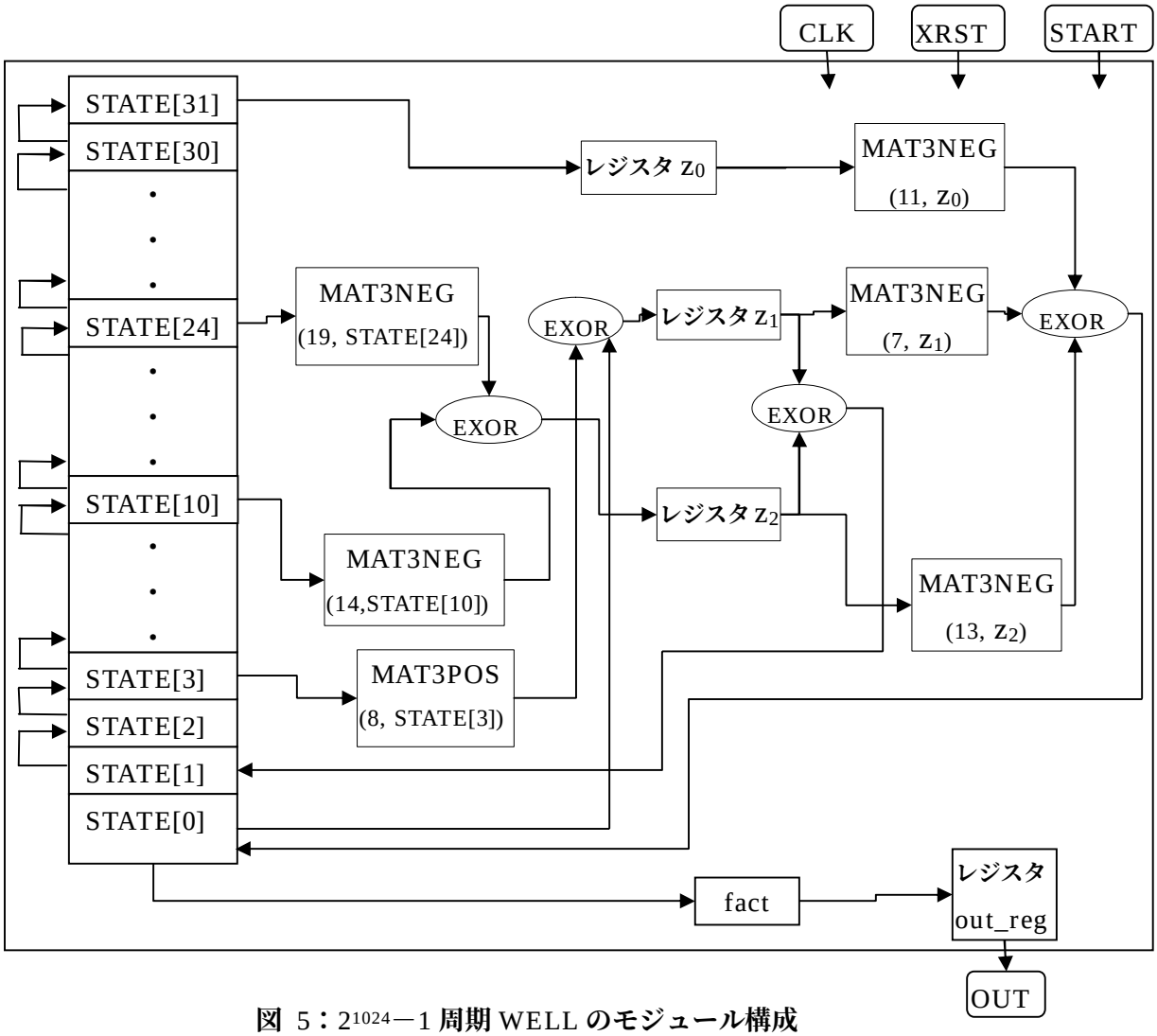


図 5：$2^{1024}-1$ 周期 WELL のモジュール構成

$2^{1024}-1$ 周期 WELL モジュールの状態遷移を表 4 に示す。

表 4：$2^{1024}-1$ 周期 WELL モジュールの状態遷移

状態	処理	次の状態
0	XRST でリセットし、STATE[0]~STATE[31]に初期値を入れる。START が立ち上がるまで状態 0 で待つ。START が立ち上がると状態 1 へ進む。	1
1	z_0 = STATE[31]; z_1 = STATE[0] ^ MAT3POS(8, STATE[3]); z_2=MAT3NEG(19,STATE[24]) ^ MAT3NEG(14, TATE[10]);	2
2	STATE[31] = STATE[30];	1,3

	• • • STATE[2] = STATE[1]; STATE[1] = z_1 ^z_2; STATE[0] = MAT3NEG(11,z_0) ^ MAT3NEG(7,z_1) ^ MAT3NEG(13,z_2);	
3	out_reg = fact(STATE[0]), out_reg を OUT から出力。	

$2^{512}-1$ 周期 WELL と同様の手順で処理行っている。だだし周期を増やしているため、使用する STATE の数が増えている。また処理途中のファンクションモジュールと、演算内容が少し変わっている。

$2^{1024}-1$ 周期 WELL の C のプログラムの実行結果とのハードウェア化によるシミュレーション結果を表 5 に示す。

表 5：$2^{1024}-1$ 周期 WELL の C プログラムと Model-Sim の乱数出力結果

	C のプログラムの実行結果	Model-Sim のシミュレーション結果
乱数 1 個目	0.31543312	31543311
乱数 2 個目	0.47029969	47029692
乱数 3 個目	0.03935820	3935820
乱数 4 個目	0.35515666	35515666
乱数 5 個目	0.35408657	35408657
乱数 6 個目	0.06067022	6067021

初期値は STATE[0] = 0 ～STATE[31] =31 を入力した。表 5 から生成された乱数がほぼ一致していることが分かる。

3.2　$2^{19937}-1$ 周期 WELL と $2^{44497}-1$ 周期 WELL の設計

$2^{19937}-1$ 周期 WELL のモジュール構成を図 6 に示す。また演算の種類と計算値は次のようになっている。

v の数 r、v_{r-1} の使わない下位 p ビット、そして v_{m1}, v_{m2}, v_{m3} の値；

r = 624,　p = 31,　$v_{m1} = v_{70}$,　$v_{m2} = v_{179}$,　$v_{m3} = v_{449}$

途中の演算である T_0 ～T_7；

$T_0 = M_3$ (-25),　$T_1 = M_3$ (27) ,　$T_2 = M_2$ (9) ,　$T_3 = M_3$ (1),

$T_4 = M_1$,　$T_5 = M_3$ (-9),　$T_6 = M_3$ (-21),　$T_7 = M_3$ (21)

初期値 v_0～v_{623} = STATE[0]～STATE[623]

周期の指数 k = 32×624－31 = 19937

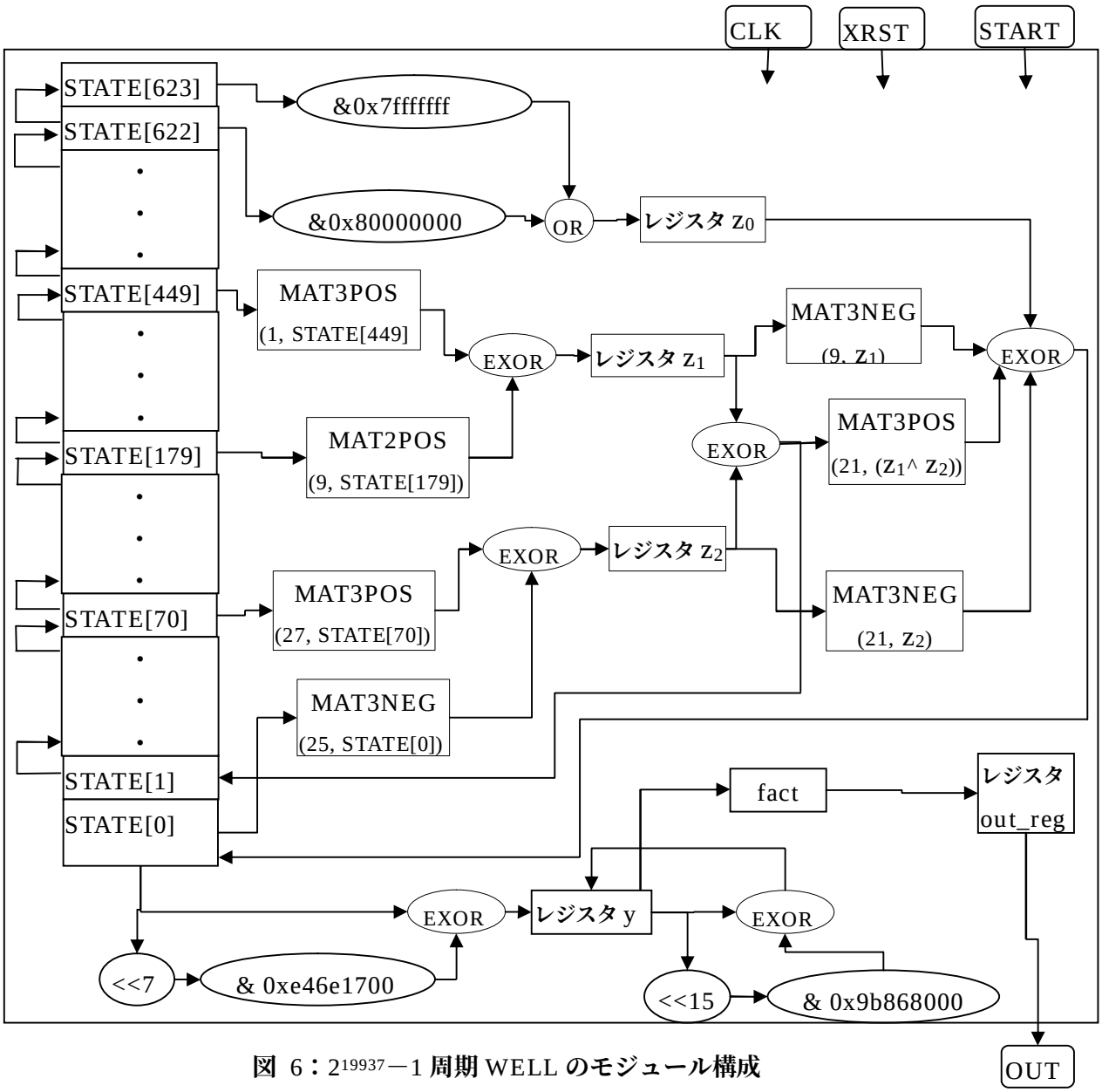


図 6：$2^{19937}-1$ 周期 WELL のモジュール構成

$2^{19937}-1$ 周期 WELL モジュールの状態遷移を表 6 に示す。

表 6：$2^{19937}-1$ 周期 WELL モジュールの状態遷移

状態	処理	次の状態
0	XRST でリセットし、STATE[0]~STATE[623]に初期値を入れる。START が立ち上がるまで状態 0 で待つ。START が立ち上がると状態 1 へ進む。	1
1	z_0 = (STATE[623]&0x7fffffff) | (STATE[622] & 0x80000000); z_1 = MAT3NEG (25, STATE[0]) ^ MAT3POS (27, STATE[70]); z_2=MAT2POS (9, STATE[179]) ^ MAT3POS (1, STATE[449]);	2
2	STATE[623] = STATE[622]; • • • STATE[2] = STATE[1]; STATE[1] = z_1 ^z_2; STATE[0] = z_0 ^ MAT3NEG (9, z_1) ^ MAT3NEG (21, z_2) ^ MAT3POS (21,(z_1 ^ z_2)); 一番最初だけ状態 1 と 3 に移り、それ以降は状態 4 に移る.	1,3,4
3	1 クロック待つ。	1,4
4	y = STATE[0] ^ ((STATE[0] << 7) & 0xe46e1700);	5
5	y = y ^ ((y << 15) & 0x9b868000);	6
6	out_reg = fact(y), out_reg を OUT から出力。	

状態 2 までは $2^{512}-1$ 周期 WELL の処理と同様の手順だが、状態 4 と状態 5 ではさらに値を変化させる処理が行われている。状態 3 ではクロックのタイミングを合わせるために 1 クロック何もしていない。また状態 4 の処理と同時に状態 1 の処理も行い、再び同じ流れで処理を行っているため、最初の乱数の生成には 5 クロックかかるが、それ以降からは 3 クロックで生成される。

$2^{19937}-1$ 周期 WELL の C のプログラムの実行結果とハードウェア化によるシミュレーション結果を表 7 に示す。

表 7：$2^{19937}-1$ 周期 WELL の C プログラムと Model-Sim の乱数出力結果

	C のプログラムの実行結果	Model-Sim のシミュレーション結果
乱数 1 個目	0.17530211	17530211
乱数 2 個目	0.88943897	88943896
乱数 3 個目	0.86056241	86056241
乱数 4 個目	0.13390419	13390419
乱数 5 個目	0.05553595	5553594
乱数 6 個目	0.92855341	92855340

初期値は STATE[0] = 0 ～STATE[623] =623 を入力した。表 7 から生成された乱数がほぼ一致していることが分かる。

次に $2^{44497}-1$ 周期 WELL のモジュール構成を図 7 に示す。また演算の種類と計算値は次のようになっている。

v の数 r、$\mathbf{v}_{r-1}$ の使わない下位 p ビット、そして $\mathbf{v}_{m1}, \mathbf{v}_{m2}, \mathbf{v}_{m3}$ の値；

r = 1391, p = 15, $\mathbf{v}_{m1} = \mathbf{v}_{23}$, $\mathbf{v}_{m2} = \mathbf{v}_{481}$, $\mathbf{v}_{m3} = \mathbf{v}_{229}$

途中の演算である $\mathbf{T}_0$ ～$\mathbf{T}_7$；

$\mathbf{T}_0 = \mathbf{M}_3$ (-24), $\mathbf{T}_1 = \mathbf{M}_3$ (30), $\mathbf{T}_2 = \mathbf{M}_2$ (-10), $\mathbf{T}_3 = \mathbf{M}_2$ (-26),

$\mathbf{T}_4 = \mathbf{M}_1$, $\mathbf{T}_5 = \mathbf{M}_3$ (20), $\mathbf{T}_6 = \mathbf{M}_6$ (9, 0xb729fcec, 0xfbffffff,0xd00020000), $\mathbf{T}_7 = \mathbf{M}_1$

$\mathbf{v}_0$～$\mathbf{v}_{1390}$ = STATE[0]～STATE[1390]

周期の指数 k = 32×1391－31 = 44497

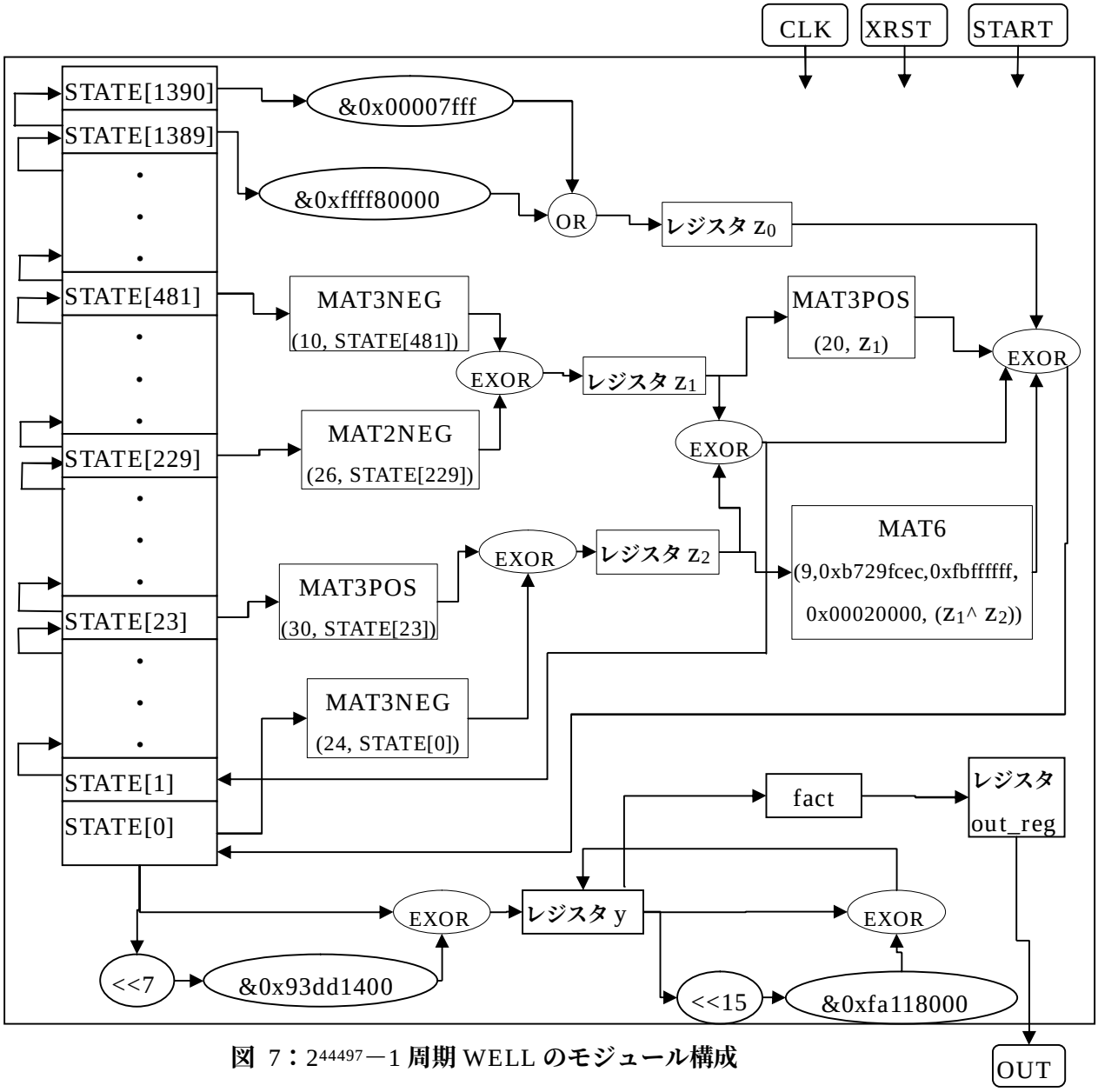


図 7：$2^{44497}-1$ 周期 WELL のモジュール構成

$2^{44497}-1$ 周期 WELL モジュールの状態遷移を表 8 に示す。

表 8：$2^{44497}-1$ 周期 WELL モジュールの状態遷移

状態	処理	次の状態
0	XRST でリセットし、STATE[0]~STATE[1390]に初期値を入れる。START が立ち上がるまで状態 0 で待つ。START が立ち上がると状態 1 へ進む。	1
1	z_0 = (STATE[1390]&0x00007fff) | (STATE[1389] & 0xffff80000); z_1=MAT3NEG(24, STATE[0])^ MAT3POS(30, STATE[23]);	2

	z_2=MAT3NEG(10, STATE[481]) ^　MAT2NEG(26, STATE[229]);	
2	STATE[1390] = STATE[1389]; • • • STATE[2] = STATE[1]; STATE[1] = z_1 ^z_2; STATE[0]=z_0 ^ MAT3POS(20 , z_1) ^ MAT6(9, 0xb729fcec, 0xfbffffff, 0xd00020000, z_2) ^ (z_1 ^ z_2); **一番最初だけ状態** 1 **と** 3 **に移り、それ以降は状態** 4 **に移る**.	1,3,4
3	1 **クロック待つ**.	1,4
4	y = STATE[0] ^ ((STATE[0] << 7) &0x93dd1400);	5
5	y =　y ^ ((y << 15) &0xfa118000);	6
6	out_reg = fact(y) , out_reg **を** OUT **から出力**。	

処理の手順は $2^{19937}-1$ **周期** WELL **と同様だが** STATE **の数、使用しているファンクションモジュール、計算の値などが違う。**

$2^{44497}-1$ **周期** WELL **の** C **のプログラムの実行結果とハードウェア化によるシミュレーション結果を表** 9 **に示す。**

表 9：$2^{44497}-1$ **周期** WELL **の** C **プログラムと** Model-Sim **での乱数出力結果**

	C **のプログラムの実行結果**	Model-Sim **のシミュレーション結果**
乱数 1 **個目**	0.38660784	38660784
乱数 2 **個目**	0.39431377	39431376
乱数 3 **個目**	0.13025174	13025174
乱数 4 **個目**	0.14583805	14583804
乱数 5 **個目**	0.63032099	63032098
乱数 6 **個目**	0.64032099	64581626

初期値は STATE[0] = 0 ～STATE[1390] = 1930 **を入力した。表** 9 **から生成された乱数がほぼ一致していることが分かる。**

4　WELL の評価

4.1　論理合成による WELL の評価

回路規模の検証に使用したデバイスを表 10 に示す。Xilinx 社の提供する ISE9.2i を使用してスライス数、フリップフロップ数、最大動作周波数の検証を行った。ハードウェア化した 4 種類の WELL の回路規模を表 11 に示す。

初めは C プログラムと同じ記述方法である、STATE[ ]の[ ]の中を変数にして、値のシフトを行わず、生成された値を格納する STATE[ ]を変えるという方法（例えば $2^{512}-1$ 周期 WELL 場合、乱数 1 個目の時は $\mathbf{v}_0$ = STATE[0]であるが、2 個目の時は $\mathbf{v}_0$ = STATE[15]、3 個目の時は $\mathbf{v}_0$ = STATE[14]と、他の値をシフトしない代わりに $\mathbf{v}_0$ に相当する STATE[ ]をずらす方法)、で設計していたが、それをシンプルに STATE[0]に乱数を格納し、他の値をシフトして演算を行う方法に変えて設計をすることでスライス数とフリップフロップ数が約半分になるまで回路規模を小さくすることができた。この他にもプログラム中でできる限り無駄な記述を減らすようにして、プログラムを改良し、できるだけ回路規模や最大動作周波数が良くなるようにした。なお、表 11 では $2^{512}-1$ 周期 WELL を WELL512 と表記しており、他の周期のものも同様である。

周期が大きいものほどスライス数やフリップフロップ数が大きくなっていることがわかる。WELL19937 と WELL44497 ではスライス数とフリップフロップ数が 500%以上となり、デバイスの規模を大きく超えてしまっている。また最大動作周波数はどれも同じ 38.339MHz となった。

乱数 1 個生成するのに必要なクロック数は WELL512 と WELL1024 は 1 個目の乱数が 3 クロックで 2 個目以降の乱数は 2 クロック、WELL19937 と WELL44497 は 1 個目の乱数が 5 クロックで 2 個目以降の乱数は 3 クロックである。それぞれの乱数 1 個を生成するのに必要なクロック数を 2 クロックと 3 クロックとすると、乱数 1 個を生成する時間は

$(1/(38.339\times10^{6}))\times2 = 52.17\text{ns}$　　$5.217\times10^{-8}\text{s}$　　(WELL512 と WELL1024)

$(1/(38.339\times10^{6}))\times3 = 78.25\text{ns}$　　$7.825\times10^{-8}\text{s}$　　(WELL19937 と WELL44497)

である。

表 10：使用デバイス

Family	Spartan3
Device	XC3S200
Package	FT256
Speed Grade	－5

表 11：ハードウェア化された WELL の回路規模

	スライス数	フリップフロップ数	最大動作周波数 (MHz)	スライス使用率(%)	フリップフロップ使用率(%)
WELL512	565	694	38.339	29	18
WELL1024	869	1206	38.339	45	31
WELL19937	11717	20151	38.339	610	524
WELL44497	25795	44711	38.339	1343	1164

表 12 に C プログラムでの WELL の実行時間を示す。

WELL のプログラムの計算前と後のそれぞれに現在の時間を測る関数を挿入し、その差分より実行時間を求めている。4 種類の WELL 全てに 10000 個、100000 個、1000000 個の 3 パターンの乱数を 5 回ずつ出力させ、その平均の時間を取っている。さらにその時間を 3 パターンごとの個数で割って、乱数 1 個の生成時間を求め、3 個の時間の平均を乱数 1 個生成した時の時間とている。

またプログラムの実行環境として Intel Xeon 3.00GHz CPU と 2.50GB RAM を使用し、windows の cygwin 上で実行している。

表 12：C 言語による WELL の実行時間

	乱数 10000 個生成した時間(s)	乱数 100000 個生成した時間(s)	乱数 1000000 個生成した時間(s)	乱数 1 個生成した時間(ns)
WELL512	2.632	24.937	263.447	258777
WELL1024	2.632	26.522	263.472	263977
WELL19937	2.634	26.342	263.479	263433
WELL44497	2.633	26.341	263.472	263400

表 11、表 12 の結果から WELL のハードウェア化より $2^{512}-1$ 周期 WELL は 4960 倍、$2^{1024}-1$ 周期は 5060 倍、$2^{19937}-1$ 周期は 3373 倍、$2^{44497}-1$ 周期は 3366 倍の速度向上が得られた。

4.2　FPGA ボードへの実装

表 10 のデバイスを使用して $2^{512}-1$ 周期と $2^{1024}-1$ 周期 WELL を FPGA ボードへ実装した。表 11 にあるように、$2^{19937}-1$ 周期と $2^{44497}-1$ 周期 WELL は回路規模が大きすぎるため、実装していない。

FPGA ボードの動きとして、リセットボタンで初期値が入り、スタートを押すと演算処理を始め、乱数 1 個を生成すると処理が止まり、スタートを押すと再び演算処理を始める、というような設計を行った。

乱数は 16 進数で表示している。乱数は 16 進数表記で最高 16 桁の値を生成するが、FPGA ボードは 4 桁しか表示できないため、16 桁を 4 桁ずつに分け、スイッチを切り替えることで、それぞれ 4 桁を表示できるようにしている。

$2^{512}-1$ 周期 WELL を実装して、1 個目の乱数の上位 4 桁を表示させた FPGA ボードを図 8 に示す。

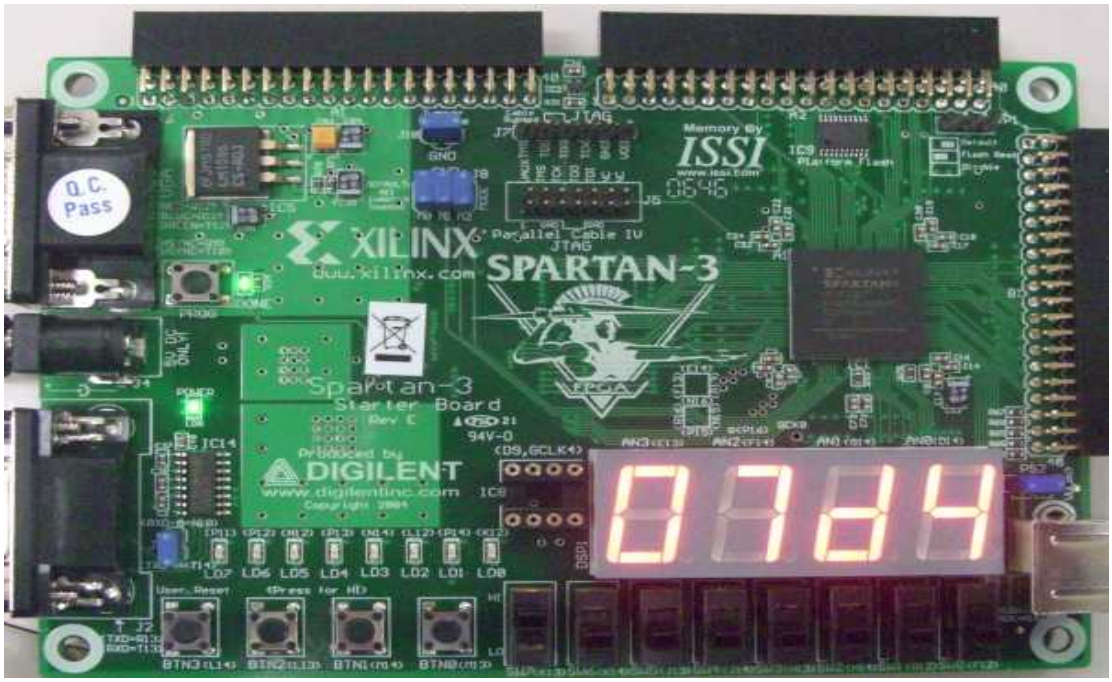


図 8：$2^{512}-1$ 周期 WELL の FPGA への実装

$2^{512}-1$ 周期 WELL のシミュレーションによる 1 個目の乱数は 10 進数表記で 32884952328922254 である。この乱数は 16 進数表記の 16 桁表示にすると 07d4b09a9e05648e となる。この値の上位 4 桁を見ると図 8 と一致していることが分かる。

$2^{1024}-1$ 周期 WELL も同様に実装し、、シミュレーションと同じ値が FPGA ボードに表示されることを確認した。

4.3　考察

表 11 を見ると WELL の周期が大きくなるにつれてスライス数とフリップフロップ数が大きくなっている。これは周期が大きいほど使用しているレジスタの数が多くなるためだと考えられる。レジスタ数は $2^{1024}-1$ 周期 WELL は $2^{512}-1$ 周期 WELL の 2 倍弱、$2^{19937}-1$ 周期 WELL は $2^{1024}-1$ 周期 WELL の 20 倍弱、そして $2^{44497}-1$ 周期 WELL は $2^{19937}-1$ 周期 WELL の 2 倍強である。表 11 でスライス数とフリップフロップ数もそれに近い割合で増加している。

$2^{19937}-1$ 周期と $2^{44497}-1$ 周期 WELL のスライス数とフリップフロップ数は両方とも限度を大きく超えている。おそらくどれだけ回路規模を抑えて設計しようとしても 100%以内にするのは困難だと思われる。この二つを FPGA に実装するにはもっと回路規模の大きいデバイスを使用する必要がある。

最大動作周波数は全ての WELL が 38.399MHz と同じであった。これはそれぞれの WELL の処理内容が多少の演算や計算に使う値が少し違うだけであり、基本的に殆ど同じであるためだと考えられる。

5　おわりに

本論文では、周期の違う 4 種類の擬似乱数発生器 WELL のハードウェア化とそのシミュレーションを Model-Sim XEⅢ6.2g で行い、そして設計した WELL の回路検証を ISE 9.2i で行った。シミュレーションでは C プログラムの実行結果とほぼ同じ値が出ていることを確認した。WELL のハードウェア化による高速化については、C 言語で実行した場合と比較して、$2^{512}-1$ 周期と $2^{1024}-1$ 周期 WELL は約 5000 倍、$2^{19937}-1$ 周期と $2^{44497}-1$ 周期 WELL は約 3350 倍の向上が得られた。

今後の課題として初期値回路の設計が挙げられる。本研究で設計した WELL には初期値が、1 番周期の小さい $2^{512}-1$ 周期 WELL で 16 個、1 番周期が大きい $2^{44497}-1$ 周期 WELL に至っては 1391 個必要とし、初期値の設定が大変である。そのため今回の設計では外部から初期値を入力するのではなくリセットで同じ初期値がセットされるようになっている。しかし、それでは初期値の変更が困難になってしまうため、入力された一つの値(種)から初期値が作られる回路が必要である。

また、種を人が決めるのではなく、現在の時刻や温度などの外部の何かから種を得る回路の設計が挙げられる。

謝辞

本研究の機会を与えてくださり、貴重な助言、ご指導を頂きました山崎勝弘教授に深く感謝いたします。

また、本研究で様々な助言を頂いた松崎裕樹氏、和田智行氏、志水建太氏、Hoang Anh Tuan **氏をはじめ高性能計算研究室の皆様に心より感謝いたします。**

参考文献

[1]FRANCOIS PANNETON、PIERRE L'ECUYER、松本眞：Improved Long-Period Generators Based on Linear Recurrences Modulo 2, 出展 2004.

[2] WELL Random number generator
http://www.iro.umontreal.ca/~panneton/WELLRNG.html

[3]土屋直幸：HDL による乱数発生回路 メルセンヌ・ツイスタの設計と実装, 立命館大学理工学部情報学科卒業論文, 2006

[4]小林優：“改訂　入門 Verilog-HDL 記述”, CQ 出版社, 2004

[5]Verilog-HDL 文法
http://monpe.cliff.jp/design/verilog/verilog_base_01.html#Index02_03

[6]基礎統計解析
http://uca-works.com/Part2/ch2_rand/rng.html