Шашков Алексей

Отчёт по проектированию процессора шифрования RSA

Определение задачи

Задача: спроектировать процессор зашифрования и расшифрования данных по алгоритму RSA с параметризуемыми размерами входных/выходных данных: изучить алгоритм RSA, изучить возможные эффективные методы реализации процессора RSA, получить описание быстродействующего процессора RSA на языке System Verilog, осуществить верификацию полученного описания процессора.

Алгоритм шифрования RSA

RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) — криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел; создан в 1977 году. На сегодня RSA является открытым для использования во всех странах (патент в США истёк в 2000 году, в других странах не патентовался).

Зашифрование RSA:

$c = m^e \bmod n$, где m – шифруемая часть сообщения, такая, что $m < n$,
$n = pq$ (p , q – произвольные простые числа) и e – число, взаимно простое с $(p-1)(q-1)$.

n и q составляют *открытый ключ*. p , q не должны быть раскрыты.

Расшифрование RSA:

$m = c^d \bmod n$, где d – *закрытый ключ*, $d = e^{-1} mod((p-1)(q-1))$.

Таким образом, алгоритм шифрования RSA представляет собой вычислительную задачу вида $b^p \bmod m$.

Анализ вычислительной сложности $b^p \bmod m$

Алгоритм RSA как для зашифрования, так и для расшифрования представляет собой возведение основания b в степень p с последующим взятием результата операции по модулю m (т. е. нахождением остатка от деления числа b^p на m для $b^p \bmod m$).

Эффективным алгоритмом нахождения степени числа является бинарный алгоритм «справа-налево», в котором степень p представляется в двоичном виде:

$p = \sum_{i=0}^{n-1} a_i \cdot 2^i$, где n – число двоичных разрядов, необходимых для представления p .

Таким образом,

$$b^p = b^{\sum_{i=0}^{n-1} a_i \cdot 2^i} = \prod_{i=0}^{n-1} \left(b^{2^i}\right)^{a_i} .$$

Так, 3^{10} можно представить как

$$3^{10} = 3^{(1010)_2} = (3)^0 \cdot (3^2)^1 \cdot (3^4)^0 \cdot (3^8)^1 .$$

С другой стороны, существует свойство

$$(a \cdot b) \bmod m = (a \bmod m \cdot b \bmod m) \bmod m \quad ,$$

согласно которому выражение $b^p \bmod m = b^{\sum_{i=0}^{n-1} a_i \cdot 2^i} \bmod m = \prod_{i=0}^{n-1} (b^{2^i})^{a_i} \bmod m$ можно представить в виде следующего Алгоритма 0 (взято из B. Schneier, Applied Cryptography).

Алгоритм 0: возведение в степень по модулю по бинарному методу «справа налево» ModExp(base, exponent, modulus) для (base^{exponent}) (mod modulus).

```
result = 1;
base = base (mod modulus);
while exponent > 0
   if (exponent mod 2 == 1) result = (result * base) (mod modulus);
   exponent = exponent >> 1;
   base = (base * base) (mod modulus);
return result;
```

В Алгоритме 0 выражение (exponent mod 2 == 1) определяет значение текущего разряда степени exponent, а (exponent >> 1) сдвигает следующий разряд экспоненты.

Можно увидеть, что Алгоритм 0 сводится к множественным вычислениям выражений вида x * y (mod m). Существует классическая схема умножения по модулю, которая сводится к Алгоритму 0.2.

Алгоритм 0.2: классическая схема умножения по модулю z = x * y (mod m).

```
z = 0;
for i in n-1 downto 0 begin  // n – число двоичных разрядов в y
        Z = (2*Z + y_i*x) (mod m);  // y_i – текущий двоичный разряд y
end for;
return z;
```

Существенным недостатком Алгоритма 0.2 является само выражение (2*Z + y_i*X) (mod m), для вычисления которого требуется существенно затратная операция взятия по модулю, сводящаяся, например (в одном из вариантов), к итеративным процедурам вычитания и сравнения.

Для решения этой проблемы был разработан алгоритм Монтгомери, в котором операции по mod m заменяются на операции по mod 2^k, при использовании которых, как будет показано далее, вычисления существенно упрощаются.

Алгоритм Монтгомери для вычисления $b^p \bmod m$

Алгоритм Монтгомери – алгоритм, позволяющий ускорить выполнение операций умножения и возведения в квадрат по модулю, необходимых при возведении числа в степень по модулю, когда модуль велик. Был предложен в 1985 году Питером Монтгомери в статье «Modular Multiplication Without Trial Division».

Пусть a, b < n, n – k-разрядное число. Тогда A = a*r (mod n) – *n-представление* числа a по r, где $r=2^k$, а НОД(r, n) = 1 (n *обязательно* нечётное). B = b*r (mod n) – *n-представление* числа b по r.

Тогда Z = A*B*r^{-1} (mod n) – произведение Монтгомери, где r*r^{-1}= 1 (mod n).

Важным свойством умножения Монтгомери является то, что

$$Z = A*B*r^{-1} \text{ (mod n)} = a * r * b * r * r^{-1} \text{ (mod n)} = (a * b) * r \text{ (mod n)},$$

то есть Z – это *n-представление* произведения (a * b) по r.

Произведение Монтгомери для $Z * 1 * r^{-1} \pmod n$ позволяет получить $a * b \pmod n$:

$$Z * 1 * r^{-1} \pmod n = a * b * r * r^{-1} \pmod n = a * b \pmod n.$$

Также полезным свойством для оптимизации вычислений является

$$A * b * r^{-1} \pmod n = a * r * b * r^{-1} \pmod n = a * b \pmod n.$$

Алгоритм умножения Монтгомери описывает вычисление выражения $A*B*r^{-1} \pmod n$ без использования затратных операций вычисления остатка от деления на n. Не считая затрат на получение n-представлений операндов и получения реальных выходных чисел из их n-представлений, используя умножения Монтгомери можно существенно ускорить вычисления вида $M=C^d \pmod n$ (в частности, RSA), так как такие вычисления содержат большое количество промежуточных умножений по модулю n.

Алгоритм 1 описывает умножение Монтгомери без конечного вычитания для $r = 2^k$.

Алгоритм 1: умножение Монтгомери MontMult(A, B, n) без конечного вычитания для $r = 2^k$.

```
S[0] = 0;
for i in 0 to k-1 loop  // k – число разрядов в A, B, n
        q_i = (S[i]_0 + A_i * B_0) mod 2;  // mod 2 эквивалентно взятию (S[i]_0 + A_i * B_0)_0
        S[i+1] = (S[i] + A_i * B + q_i * n) / 2;  // для всех S: 0 ≤ S < M+B
end loop;
return S[k];  // S = (A * B + Q * M) / 2^k = (A * B + Q * M) / r
```

В Алгоритме 1 двоичный разряд q_i подобран таким образом, чтобы выражение $S+a_i*B+q_i*M$ всегда делилось на 2 без остатка. Если определить $A_i = \sum_{j=0}^{i} a_j \cdot 2^j$ и $Q_i = \sum_{j=0}^{i} q_j \cdot 2^j$, тогда $A_i = A_{i-1} + a_i*2^i$ и $A_k = A$. Тогда, раскрывая Алгоритм 1, $2^{i+1} * S[i] = A_i*B+Q_i*n$, а по окончании работы алгоритма $2^k * S[k] = A*B+Q*n$, или $S[k] = (A*B+Q*n) / 2^k = (A*B+Q*n) / r$, причём $(A*B+Q*n)$ делится на r без остатка.

Можно показать, что выражение $(A*B+Q*M) / r \equiv A*B*r^{-1} \pmod n$, где $A*B*r^{-1} \pmod n$ является умножением Монтгомери по определению. Так, $(A*B+Q*n) / r \pmod n = (A*B+Q*n) * r * r^{-1} / r \pmod n = (A*B+Q*M) * r^{-1} \pmod n$.

В статье «Montgomery Exponentiation Needs No Final Subtractions» К. Уолтер показал, что при условиях $(A < 2n)$, $(B < 2n)$ и $(2^{k-1} > 2n)$ является верным неравенство $(S[k] < 2n)$, то есть результатом *полного* умножения Монтгомери $Z = A*B*r^{-1} \pmod n$ будет являться либо S[k], либо $(S[k] - n)$. При $(S[k] < 2n)$ ввиду свойства $(a-n) \bmod n = a \bmod n$ при последовательном умножении чисел по модулю можно довольствоваться передачей значения S[k] в качестве входных данных (т. е. новых A или B) без потери корректности алгоритма. Также можно показать, что при получении искомого выходного числа из его n-представления на конечном этапе последовательности умножений (MontMult(Z, 1, n)) верно неравенство $(S[k] \le n)$, а значит S[k] или 0 является искомым результатом и нет необходимости находить $(S[k] – n)$. Таким образом, перейдя к Монтгомери-образам входных чисел и имея на каждом последующем этапе лишь неполный результат умножения Монтгомери S[k] с оконечным MontMult(Z, 1, n) можно безошибочно вычислить любую последовательность умножений по модулю, такую как, например, возведение в степень по модулю.

Важно отметить, что условие $(2^{k-1} > 2n)$ требует выбирать число разрядов k как минимум на 2 большим, чем эффективное число разрядов в n: $k \ge 2+min\_no\_of\_bits(n)$.

Пример: вычисление MontMult(2, 1, 511).
$A = 2 = (0\ 0000\ 0010)_2$;
$B = 1 = (0\ 0000\ 0001)_2$;

$n = 511 = (1\ 1111\ 1111)_2$, значит: ($2^{k-1} > 2n$), ($2^{k-1} > 11\ 1111\ 1110$), ($2^{10} > 11\ 1111\ 1110$), ($k - 1 = 10$), $k = 11$, где k – число разрядов для A, B, n.
Отсюда $r = 2^k = 2^{11} = 2048$.
$r*r^{-1} = 1 \pmod{511}$, $r^{-1} = 128$ т. к. $2048 * 128 = 1 \pmod{511}$.
0. S[0] = 0.
От i = 0 до i = k-1 = 10:
1. i = 0.
   $q_0 = (0 + 0*1) \bmod 2 = 0$;
   $S[1] = (0 + 0*1 + 0*511) / 2 = 0$;
2. i = 1.
   $q_1 = (0 + 1*1) \bmod 2 = 1$;
   $S[2] = (0 + 1*1 + 1*511) / 2 = 512 / 2 = 256$;
3. i = 2.
   $q_2 = (0 + 0*1) \bmod 2 = 0$;
   $S[3] = (256 + 0*1 + 0*511) / 2 = 256 / 2 = 128$;
4. i = 3.
   $q_3 = (0 + 0*1) \bmod 2 = 0$;
   $S[4] = (128 + 0*1 + 0*511) / 2 = 128 / 2 = 64$;
5. i = 4. $q_4 = 0$; $S[5] = 32$;
6. i = 5. $q_5 = 0$; $S[6] = 16$;
7. i = 6. $q_6 = 0$; $S[7] = 8$;
8. i = 7. $q_7 = 0$; $S[8] = 4$;
9. i = 8. $q_8 = 0$; $S[9] = 2$;
10. i = 9.
    $q_9 = (0 + 0*1) \bmod 2 = 0$;
    $S[10] = (2 + 0*1 + 0*511) / 2 = 1$;
11. i = 10.
    $q_{10} = (1 + 0*1) \bmod 2 = 1$;
    $S[11] = (1 + 0*1 + 1*511) / 2 = 512 / 2 = 256$;
12. return S[11];
Результат: MontMult(2, 1, 511) = S[11] = 256.
С другой стороны, $\text{MontMult}(2, 1, 511) \equiv A*B*r^{-1} \pmod{511}$, $(A*B)*r^{-1} \pmod{511} = (2*1)*128 \pmod{511} = 256$.

Алгоритм 2 описывает возведение в степень с использованием операции умножения Монтгомери и позволяет получить значение $m = c^d \pmod n$. Алгоритм 2 реализует вариант бинарной схемы возведения в степень по модулю «справа налево» (Алгоритм 0), в котором классические умножения и возведения в квадрат по модулю заменяются на неполные умножения Монтгомери.

Алгоритм 2: возведение в степень по модулю с использованием операции умножения Монтгомери
MontExp(c, d, n).
$K = 2^{2k} \pmod n = r^2 \pmod n$; *// Вычисляется внешне.*
R[0] = MontMult(K, 1, n);
P[0] = MontMult(K, c, n);
for i in 0 to (t-1) loop // t – *эффективное число разрядов в* d.
    if (d[i] = 1) then
        R[i+1] = MontMult(R[i], P[i], n);
    end if;
    if (i != t-1) then
        P[i+1] = MontMult(P[i], P[i], n);
    end if;
end loop;
R[t+1] = MontMult(R[t], 1, n);
if (R[t+1] == n) then
    R[t+1] = 0;
end if;
return m = R[t+1];

Раскрывая Алгоритм 2 аналитически, например, для $m = c^3 \pmod n = c^{(11)_2} \pmod n$, t = 2, можно убедиться в правильности алгоритма.

$K = 2^{2k} \pmod n = r^2 \pmod n$;
1. $R[0] = MontMult(K, 1, n) = r^2 * 1 * r^{-1} \pmod n = r \pmod n$;
   $P[0] = MontMult(K, c, n) = MontMult(r^2, c, n) = r^2 * c * r^{-1} \pmod n = c * r \pmod n = C$.
От i = 0 до i = 2 – 1 = 1:
2. i = 0, $e_0 = 1$;
   $R[1] = MontMult(R[0], P[0], n) = r * C * r^{-1} \pmod n = C \pmod n$;
   $P[1] = MontMult(P[0], P[0], n) = C * C * r^{-1} = C^2 * r^{-1} \pmod n$;
3. i = 1, $e_1 = 1$;
   $R[2] = MontMult(R[1], P[1], n) = C * C^2 * r^{-1} * r^{-1} \pmod n = C^3 * r^{-2} \pmod n$;
5. $R[3] = MontMult(R[2], 1, n) = C^3 * r^{-2} * 1 * r^{-1} = C^3 * r^{-3} \pmod n$;
6. return $m = R[3] = C^3 * r^{-3} \pmod n$.
Результат $m = C^3 * r^{-3} \pmod n$, но $C^3 * r^{-3} \pmod n = (C * r^{-1})^3 \pmod n = c^3 \pmod n$. т. к $C * r^{-1} \pmod n = c * r * r^{-1} \pmod n = c \pmod n$. Таким образом $m = c^3 \pmod n$.

Пример: вычисление $2^8 \pmod{511}$ с применением MontExp(2, 8, 511).
c = 2;
$d = 8 = (1000)_2$; отсюда t = 4;
$n = 511 = (1\ 1111\ 1111)_2$, значит: ($2^{k-1} > 2n$), ($2^{k-1} > 11\ 1111\ 1110$), ($2^{k-1} > 11\ 1111\ 1110$), ($2^{10} > 11\ 1111\ 1110$), (k – 1 = 10), k = 11, где k – число разрядов для c, d, n.
Отсюда $r = 2^k = 2^{11} = 2048$.
$r*r^{-1} = 1 \pmod{511}$, $r^{-1} = 128$ т. к. $2048 * 128 = 1 \pmod{511}$.
$K = 2^{2k} \pmod n = 2^{2*11} \pmod n = 2^{22} \pmod{511} = 16$;

0. $R[0] = MontMult(16, 1, 511) = 16 * 1 * 128 \pmod{511} = 256 \pmod{511} = 4$;
   $P[0] = MontMult(16, 2, 511) = 16 * 2 * 128 \pmod{511} = 8$;
От i = 0 до i = 4 - 1 = 3
1. i = 0, $d_0 = 0$;
   $P[1] = MontMult(8, 8, 511) = 8 * 8 * 128 \pmod{511} = 16$;
2. i = 1, $d_1 = 0$;
   $P[2] = MontMult(16, 16, 511) = 16 * 16 * 128 \pmod{511} = 64$;
3. i = 2, $d_2 = 0$;
   $P[3] = MontMult(64, 64, 511) = 64 * 64 * 128 \pmod{511} = 2$;
4. i = 3, $d_3 = 1$;
   $R[4] = MontMult(4, 2, 511) = 4 * 2 * 128 \pmod{511} = 2$;
5. $R[5] = MontMult(2, 1, 511) = 2 * 1 * 128 \pmod{511} = 256$;
6. return M = R[5] = 256;
Результат: MontExp(2, 8, 511) = R[5] = 256.
С другой стороны, $MontExp(2, 8, 511) = 2^8 \pmod{511} = 256 \pmod{511} = 256$.

Реализация процессора шифрования RSA с использованием алгоритма Монтгомери

По Алгоритму 2 очевидно, что основными вычислительными блоками процессора шифрования RSA будут являться умножители Монтгомери, реализующие Алгоритм 1 (MontMult).

Рассматривая Алгоритм 1 (MontMult) можно заключить, что при реализации умножителя Монтгомери в аппаратуре основная схемная задержка будет представлена в логике, реализующей выражение $S[i+1] = (S[i] + A_i * B + q_i * n) / 2$. Вычисление данного выражения требует суммирования чисел очень большой разрядности, и если это вычисление реализовать с помощью обычного сумматора, то задержка прихода переноса к старшему разряду сумматора будет очень существенной, учитывая то, что разрядность операнда может составлять k = 1024 разряда и более.

Однако существует вид сумматора, в котором задержка переноса отсутствует как класс. Таким сумматором является сумматор с сохранением переноса, или ССП (carry-save adder или CSA). В ССП есть три входа x1, x2, x3 с одинаковой разрядностью k, и два выхода с разрядностью (k+1) – s и c, причём s + c = x1 + x2 + x3. В ССП выход суммы s = {0, x1 xor x2 xor x2}, а выход переноса c = {(x1 and x2) or (x1 and x3) or (x2 and x3), 0}.

Учитывая, что результирующей суммой в ССП является два числа s и c, а не одно, и что в алгоритме MontExp результаты двух вычислений MontMult попадают на вход одного MontMult, то одним из оптимальных вариантов представления выражения $S[i+1] = (S[i] + A_i * B + q_i * n) / 2$ будут являться схемы, показанные на рис. 1 и на рис. 2. В этом представлении $S = S0 + S1$, $A = A0 + A1$, $B = B0 + B1$. Так как для A_i необходимо знать знать полную сумму $A = A0 + A1$, то одноразрядный полный сумматор, показанный на рис. 2, будет предоставлять необходимое в текущем такте значение A_i.

Таким образом, используя схему с каскадом ССП (рис. 1) и схему сумматора (рис. 2) можно представить Алгоритм 1 MontMult как показано в Алгоритме 3.

Алгоритм 3: умножение Монтгомери с использованием ССП «5 к 2»

```
MontMult(A0, A1, B0, B1, n)
S1[0] = 0;
S2[0] = 0;
for i in 0 to k-1 loop  // k – число разрядов в A, B, n
        q_i = ((S0[i]_0 + S1[i]_0 ) + (A_i * (B0_0 + B1_0)) mod 2;
S0[i+1], S1[i+1] = CSA_5_2(S0[i] + S1[i] + A_i * (B0 + B1) + q_i * n) / 2;
end loop;
return S0[k], S1[k];
```

Число тактов для выполнения Алгоритма 3 можно записать как N = (k+1), где k – полное число разрядов в представлении n.

Нетрудно увидеть, что в Алгоритме 2 MontExp выражения R[i+1] = MontMult(R[i], P[i], n) и P[i+1] = MontMult(P[i], P[i], n) полностью независимы и могут быть вычислены параллельно. Поэтому для ускорения вычислений в процессоре RSA будет рациональным использовать два умножителя Монтгомери.

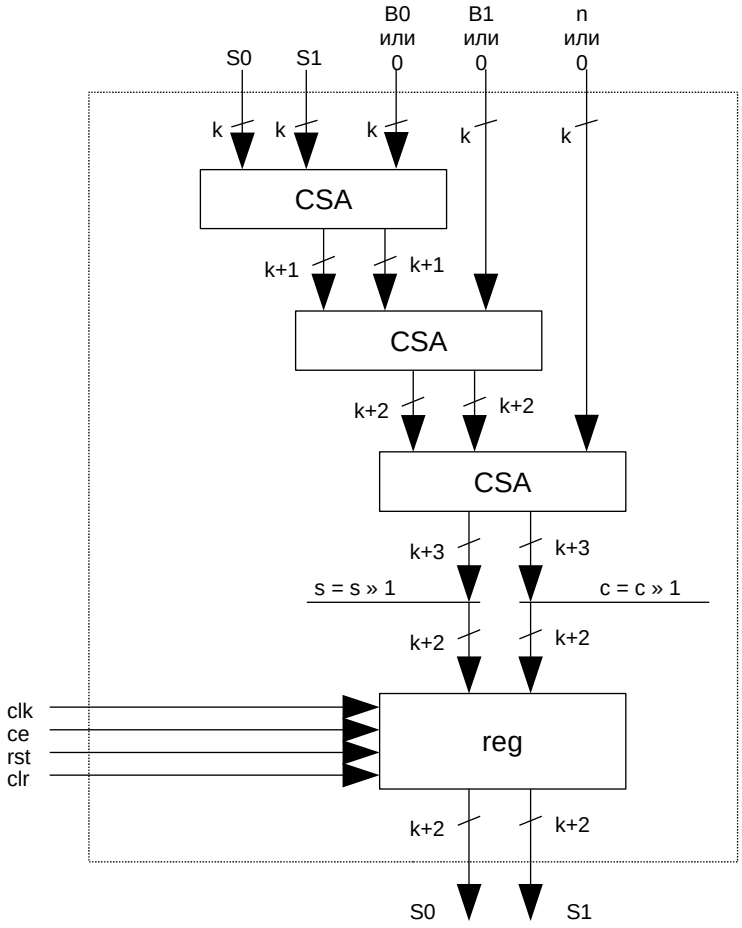


Рис. 1 – схема ССП «5 к 2» / 2 для реализации MontMult

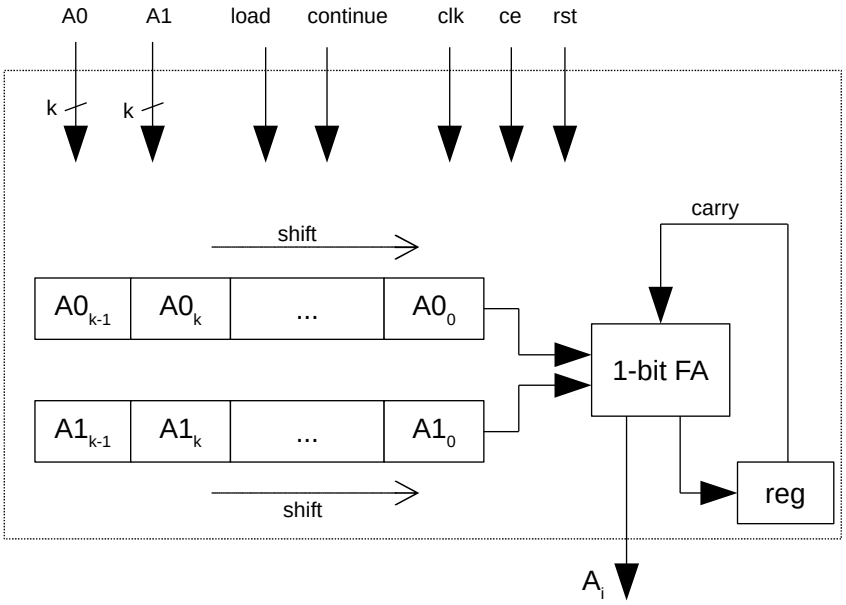


Рис. 2 – схема поразрядного сумматора для реализации MontMult

В Алгоритме 4 показан модифицированный вариант MontExp с использованием MontMult на базе ССП «5 к 2», описанного в Алгоритме 3.

Алгоритм 4: возведение в степень по модулю с использованием операции умножения Монтгомери с использованием ССП «5 к 2» MontExp(c, d, n).

```
K = 2^2k (mod n);  // Вычисляется внешне.
P0[0], P1[0] = MontMult(K, 0, c, 0, n);
R0[0], R1[0] = MontMult(K, 0, 1, 0, n);
for i in 0 to t-1 loop   // t – эффективное число разрядов в d.
        if (d[i] = 1) then
                R0[i+1], R1[i+1] = MontMult(R0[i], R1[i], P0[i], P1[i], n);
        end if;
        if (i != t-1) then
                P0[i+1], P1[i+1] = MontMult(P0[i], P1[i], P0[i], P1[i], n);
        end if;
end loop;
R0[t+1], R1[t+1] = MontMult(R0[t], R1[t], 1, 0, n)
m = R0[t+1] + R1[t+1];
if (m == n) then
        m = 0;
end if;
return m;
```

Ввиду того, что в Алгоритме 4 используется одна и та же функция MontMult на разных этапах алгоритма, и того, что выражения внутри цикла независимы и могут быть вычислены параллельно, то Алгоритм 4 рационально реализовать как два модуля MontMult с мультиплексорами (коммутационной логикой), настраиваемыми в соответствии с этапом алгоритма (рис. 3).

Последний этап алгоритма реализуется как поразрядный сумматор (как на рис. 2), независимый по данным от модулей MontMult, при этом модули MontMult остаются свободны. Значит, данный этап можно реализовать одновременно с этапом поступления новых данных на модули MontMult, получив двухступенчатый конвейер. Таким образом, время обработки данных при непрерывном потоке данных будет определяться временем работы модулей MontMult.

Число тактов для выполнения Алгоритма 4 можно записать как N = (k+1)(t+3), где k – полное число разрядов в представлении n, а t – эффективное число разрядов в d.*

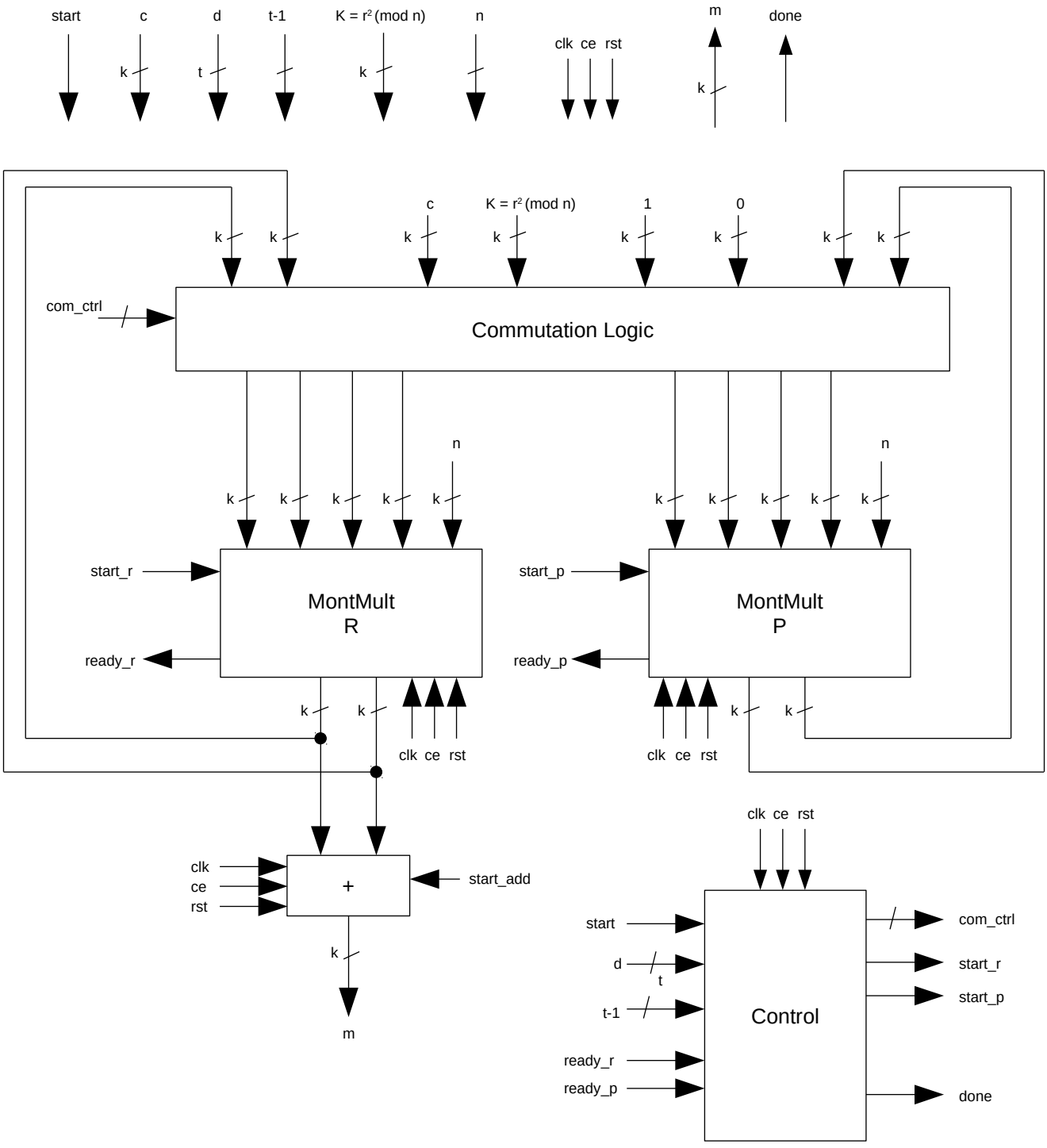


Рис. 3 – укрупнённая схема процессора RSA (mod_exp)

Возможность построения быстрого процессора расшифрования алгоритма RSA на основе Китайской теоремы об остатках

Рассмотрим Алгоритм 4, который реализуется в mod_exp. Как было указано ранее, Алгоритм 4 может вычислить как зашифрование RSA $c = m^e \bmod n$, так и расшифрование RSA $m = c^d \bmod n$ (e и d здесь и далее называются экспонентами). Однако экспонента зашифрования e обычно выбирается с небольшой эффективной разрядностью, например e = 3 или e = 65537 (типовые значения e). При этом для процесса зашифрования RSA, как видно из записи Алгоритма 4, число внутренних циклов для i от 0 до t-1 (где t – эффективное число разрядов в экспоненте) будет, соответственно, небольшим. В случае же небольших экспонент зашифрования e их соответствующие экспоненты расшифрования d, напротив, получаются с очень большой эффективной разрядностью.

Таким образом, Алгоритм 4 хоть и способен осуществлять как зашифрование, так и расшифрование, но скорость выполнения расшифрования будет на столько же медленнее, на сколько больше эффективных разрядов в d по сравнению с e. Как было указано ранее, для выполнения Алгоритма 4 необходимо число тактов N = (k+1)*(t+3), где k – число разрядов в модуле n, а t – эффективное число разрядов в d. Для одного из вариантов конфигурации RSA выберем k = 512, p = 137, q = 131, n = p*q = 137*131 = 17947, e = 3, d = 11787. Число двоичных разрядов в e для зашифрования равно 2, значит $N_{заш}$ = (k+1)*(t+3) = (512+1)*(2+3) = 2565. Число двоичных разрядов в d для расшифрования равно 14, значит $N_{расш}$ = (k+1)*(t+3) = (512+1)*(14+3) = 8721. $N_{расш} / N_{заш}$ = 3,4. То есть для процедуры расшифрования в конкретном случае мы имеем в 3,4 раза меньшую производительность, чем для зашифрования.

Таким образом, если иметь в распоряжении по одному модулю mod_exp как для зашифрования, так и для расшифрования, и при этом оба модуля работают на одинаковых частотах, то невозможно обеспечить одинаковую пропускную способность для зашифрования и для расшифрования, так как скорость расшифрования будет существенно ниже.

Однако существует способ, позволяющий существенно ускорить процесс расшифрования RSA, основанный на использовании обеих экспонент (e и d) в модуле расшифрователя. Зашифрователь RSA лишён возможности использовать этот способ, так как зашифрователь (в общем случае) не может содержать закрытый ключ d, в то время как расшифрователь может содержать как d, так и открытую экспоненту e.

В данном методе быстрого расшифрования RSA используется один из вариантов Китайской теоремы об остатках. Как известно, для RSA модуль n = p*q. Для метода быстрого расшифрования предварительно необходимо вычислить значения dP, dQ и qInv:

$$dP = d \bmod (p-1);$$
$$dQ = d \bmod (q-1);$$
$$qInv = q^{-1} \bmod p.$$

dP, dQ и qInv являются, по сути, формой представления d и могут быть предварительно рассчитаны для d. Далее необходимо осуществить следующие вычисления:

$$m_1 = c^{dP} \bmod p;$$
$$m_2 = c^{dQ} \bmod q;$$
$$h = qInv * (m_1 - m_2) \bmod p;$$
$$m = m_2 + (h * q).$$

В случае, если $(m_1 < m_2)$, $h = qInv * (m_1 + ceil(q/p)*p - m_2) \bmod p$ (для получения сопоставимого по модулю, но положительного, а не отрицательного числа).

Можно увидеть, что вычисления m_1 и m_2 независимы, а значит могут быть выполнены параллельно. Можно заметить, что вычисления m_1 и m_2 можно реализовать на модуле mod_exp ($m = c^d \bmod n$), представленном ранее, причём в данном случае p, q < n, а экспоненты dP, dQ < d, значит m_1 и m_2 можно вычислить гораздо быстрее, чем прямое $m = c^d \bmod n$ с помощью mod_exp. Фактически вычисления m_1 и m_2 являются вычислениями половинной сложности относительно $m = c^d \bmod n$.

С учётом конечных операций по вычислению h и затем m число тактов, необходимое для выполнения расшифрования, можно записать как N = (k/2+1)(t/2+3), где k – число разрядов в модуле n, а t – эффективное число разрядов в d. Для ранее указанной конфигурации RSA (k = 512, p = 137, q = 131, n = p*q = 137*131 = 17947, e = 3, d = 11787), как было найдено ранее для процесса зашифрования, $N_{заш}$ = 2565. Число двоичных разрядов в d для расшифрования равно 14, значит для быстрого расшифрования RSA с использованием Китайской теоремы об остатках можно найти $N_{расш.быстр.}$ = (k/2+1)(t/2+3) = (512/2+1)*(14/2+3) = 2570. $N_{расш.быстр.}$ / $N_{заш}$ = 2570 / 2565 = 1,002.

Таким образом, используя метод быстрого расшифрования на основе Китайской теоремы об остатках, можно существенно ускорить процесс расшифрования, и приблизить пропускную способность расшифрователя RSA к пропускной способности зашифрователя RSA, в котором использование Китайской теоремы об остатках невозможно.

Интерфейс компонента возведения в степень по модулю mod_exp

Ниже представлено описание интерфейса mod_exp на Verilog / System Verilog.

```
//% mod_exp is designated for RSA encryption or decryption class m = (c ^ d) mod n, where
//% c < n and n is odd. Odd n is the limitation of mod_exp due to the underlying fast
//% Montgomery multiplication algorithm.
module mod_exp #(parameter DATA_WIDTH = (`CONFIG_DATA_WIDTH))
(
    input clk,                                  //% Clock.
    input ce,                                   //% Clock enable.
    input rst,                                  //% Reset.
    input start,                                //% Starts encryption/decryption
                                                //% process (only if in ready state).
    input [(DATA_WIDTH-1):0] c,                 //% Input to encrypt/decrypt.
    input [(DATA_WIDTH-1):0] d,                 //% Encrypt/decrypt exponent. d > 0 only.
    input [($clog2(DATA_WIDTH)-1):0] t_sub_1,   //% Effective number of bits in
                                                //% d, but minus 1.
    input [(DATA_WIDTH-1):0] r2_mod_n,          //% r2_mod_n = (r^2) mod n =
                                                //% = ((2^(2+DATA_WIDTH))^2) mod n.
    input [(DATA_WIDTH-1):0] n,                 //% Modulus n: n must be odd.
    output ready,                               //% Denotes readiness for new
                                                //% data input (the new c,
                                                //% d, t_sub_1, r2_mod_n, and n).
    output [(DATA_WIDTH-1):0] m,                //% Output of
                                                //% encryption/decryption.
    output done                                 //% Denotes m as m = (c ^ d) mod n
                                                //% (completion of operation).
);
```

Компонент mod_exp реализует процесс возведения в степень по модулю (схема на рис. 3), или, с другой стороны, процесс зашифрования или же расшифрования RSA. Модуль mod_exp является параметризуемым и поддерживает значения DATA_WIDTH от 4 и выше. Таким образом, на базе mod_exp можно задать процессор RSA для входного операнда input [(DATA_WIDTH-1):0] c, n размерностью от 4 разрядов и выше. На практике же для RSA ширина input [(DATA_WIDTH-1):0] c, n составляет как минимум несколько сотен бит.

Под эффективной разрядностью числа здесь и далее понимается минимальное число

разрядов, необходимых для представления числа. Например, эффективной разрядностью числа $(0\ 0000\ 1010)_2$ является 4, хотя в представлении используется 9 разрядов.

Важно отметить, что для зашифрования RSA $m = c^d \bmod n$, где c – шифруемый текст, *такой, что* $c < n$. Это значит, что для корректной работы процесса зашифрования необходимо обеспечить (c<n) для любых c. Так как входные данные для зашифрования в общем случае являются случайными и не любой блок текста input [(DATA_WIDTH-1):0] c будет меньше n, то наиболее рациональным способом обеспечить (c<n) является сокращение максимального эффективного числа разрядов input [(DATA_WIDTH-1):0] c на единицу относительно максимального эффективного числа разрядов input [(DATA_WIDTH-1):0] n (желательно, чтобы эффективная разрядность n совпадала с максимальной разрядностью input [(DATA_WIDTH-1):0] n). Другими словами, для зашифрования RSA средствами mod_exp для любых input [(DATA_WIDTH-1):0] c должно быть всегда справедливым равенство (0 == c[DATA_WIDTH-1]), а c[(K-4):0] должно содержать входной текст. Тем не менее, при такой входной схеме максимальная эффективная разрядность выходного шифротекста output [(DATA_WIDTH-1):0] m будет на единицу больше максимальной эффективной разрядности входного текста input [(DATA_WIDTH-1):0] c. Этот факт создаёт определённые особенности при построении интерфейсов сопряжения в системе с шифрованием потока данных: после зашифрования выходной шифротекст будет иметь на один разряд больше, чем входной текст.

Для процесса расшифрования RSA $c = m^e \bmod n$ показанная выше схема сокращения на один разряд входных данных относительно выходных данных уже не нужна, так как m является результатом вычисления по mod n ($m = c^d \bmod n$), а значит m<n по определению, даже если эффективная разрядность m совпадает с эффективной разрядностью n. Однако для mod_exp для выходного значения текста m всегда будет справедливо равенство (0 == m[DATA_WIDTH-1]), если при шифровании использовалась схема с меньшей на разряд шириной входных данных. В таком случае при расшифровании RSA средствами mod_exp m[(K-4):0] будет содержать исходный незашифрованный текст, а это значит, что входной шифротекст на разряд больше выходного текста.

Модуль mod_exp устанавливает output ready = 1, когда первая ступень конвейера свободна для получения новых входных данных. Регистрируя output ready = 1, внешний управляющий модуль на один такт подаёт сигнал input start = 1 и одновременно выставляет новые значения на шинах input c, input d, input t_sub_1 и input r2_mod_n, после чего output ready 1 → 0, а mod_exp начинает работу. После того, как первая ступень конвейера заканчивает работу, внутренние данные передаются на вторую ступень конвейера, а первая ступень конвейера обозначается свободной с помощью output ready 0 → 1.

Вторая (заключительная) ступень конвейера поэтапно подсчитывает конечную сумму output m, причём готовое значение output m обозначается как результат $m = c^d \bmod n$ с выставлением output done = 1 на один такт.

Значение r2_mod_n – это предрассчитанное значение $(r^2) \bmod n = ((2^K)^2) \bmod n$. Если подразумевать, что вычисление параметров RSA (p, q, n = p*q, e, d) будет производиться на центральном процессоре (ЦП), и считать, что в ходе работы mod_exp значение n меняется не часто, то такое же редкое дополнительное затратное вычисление r2_mod_n на ЦП также не может представить сложности.

Выбор оптимальной разрядности входных данных для процессора шифрования алгоритма RSA

Согласно Alfred J. Menezes и др. из «Handbook of Applied Cryptography, **1996**»: "Recommended size of modulus. Given the latest progress in algorithms for factoring integers, a ***512-bit modulus n provides only marginal security*** from concerted attack. As of 1996, in order to foil the powerful quadratic sieve and number field sieve factoring algorithms, ***a modulus n of at least 768 bits is recommended***. *For long-term security, 1024-bit or larger moduli should be used*."

Некоторые современные IP-ядра с RSA-процессорами:

EXP-T6421 - High Performance PKA/RSA/ECC Security Processor, April **2015**.
http://www.design-reuse.com/sip/high-performance-pka-rsa-ecc-security-processor-ip-30128/
"Capable of over 12,000 RSA-1024 and over 5,000 RSA-2048 operations per second (at >566 MHz)".

BA414EP - Public Key crypto engine for RSA, ECC, ECDSA and ECDH, **2015**.
http://www.barco-silex.com/ip-cores/encryption-engine/Public-Key-Crypto-Engine
"RSA, CRT, DH, DSA (Digital Signature Algorithm) up to 4096 bits".

IPX-RSA - RSA Public Key Cryptography Exponentiation Accelerator, **2015**.
http://www.design-reuse.com/sip/rsa-public-key-cryptography-exponentiation-accelerator-ip-15804/
"2048-bit length inputs".

eSi-RSA - RSA public key cryptography APB peripheral, **2015**.
http://www.ensilica.com/wp-content/uploads/eSi-RSA.pdf
"The peripheral can be configured for between 512 and 4096-bit maximum key size to keep the resource requirements as low as possible".

RSA1-E - RSA Public Key Exponentiation Accelerator, **2015**.
http://www.ipcores.com/rsa_ip_core
"Support for RSA binary fields of configurable bit sizes up to 2048".

Некоторые статьи:

Researcher: RSA 1024-bit Encryption not Enough, **2011**.
http://www.pcworld.com/article/132184/article.html
"Arjen Lenstra, a cryptology professor at the École polytechnique fédérale de Lausanne (EPFL) in Switzerland, said the distributed computation project, conducted over 11 months, achieved the equivalent in difficulty of cracking a 700-bit RSA encryption key, so it doesn't mean transactions are at risk, yet".

Researchers: 307-digit key crack endangers 1024-bit RSA, **2007**.
http://arstechnica.com/uncategorized/2007/05/researchers-307-digit-key-crack-endangers-1024-bit-rsa/
"[...] experienced attackers might not have such a hard time. Getting the computing power to crack a 1024-bit key could be as easy as employing a decent-sized botnet or two".

Таким образом, относительно безопасным размером модуля n для RSA сейчас является 1024 разряда, то есть RSA-1024 можно считать минимально безопасным для любых применений.

Насколько сильно можно уменьшить разрядность ключа для шифрования потока данных по радиоканалу остаётся открытым вопросом.

Предварительная верификация компонента mod_exp средствами ModelSim/Questa

Для моделирования и верификации компонента mod_exp была использована САПР Mentor Graphics Questa Sim -64 10.2c Revision: 2013.07, являющаяся версией продукта Model Sim -64 10.2c с поддержкой ряда новых технологий (в частности, UPF), причём в Model Sim поддержка таких технологий изначально не планируется разработчиком.

Модуль mod_exp является процессором, производящим вычисления вида $m = c^d \bmod n$, где c<n и n *обязательно* нечётное. В частности, mod_exp может вычислять как зашифрование RSA, так и расшифрование RSA, если для выбранной конфигурации RSA модуль n является нечётным. Очевидно, что для проверки правильности работы модуля mod_exp вообще нет необходимости проектировать криптосистему RSA, выбирая p, q, n = p*q, e и d, а достаточно лишь подготовить некоторое количество тестовых наборов данных {n, c, d, t_sub_1, r2_mod_n}, таких, что: *n нечётное, c<n, d ≥ 1, (t_sub_1+1) — эффективное число разрядов в d, а r2_mod_n = (r^2) mod n = ((2^K)^2) mod n*, где K – выбранная полная разрядность модуля mod_exp (K = 2 + DATA_WIDTH).

Также ввиду того, что модуль mod_exp является параметризуемым, было необходимо убедиться, что HDL-описание модуля корректно для различных актуальных значений K.

Начальная отладка mod_exp осуществлялась для DATA_WIDTH = 9, K = 11 для шести наборов входных данных (файл mod_exp_tb.sv). В ходе теста были проверены вычисления (255^4) mod 511 = 32, (56^5) mod 509 = 393, (56^1) mod 509 = 56, (0^1) mod 509 = 0, (1^1) mod 509 = 1, (45^{1529}) mod 225 = 0. Результаты m для mod_exp сошлись с расчётными, как отображено на рис. 4.

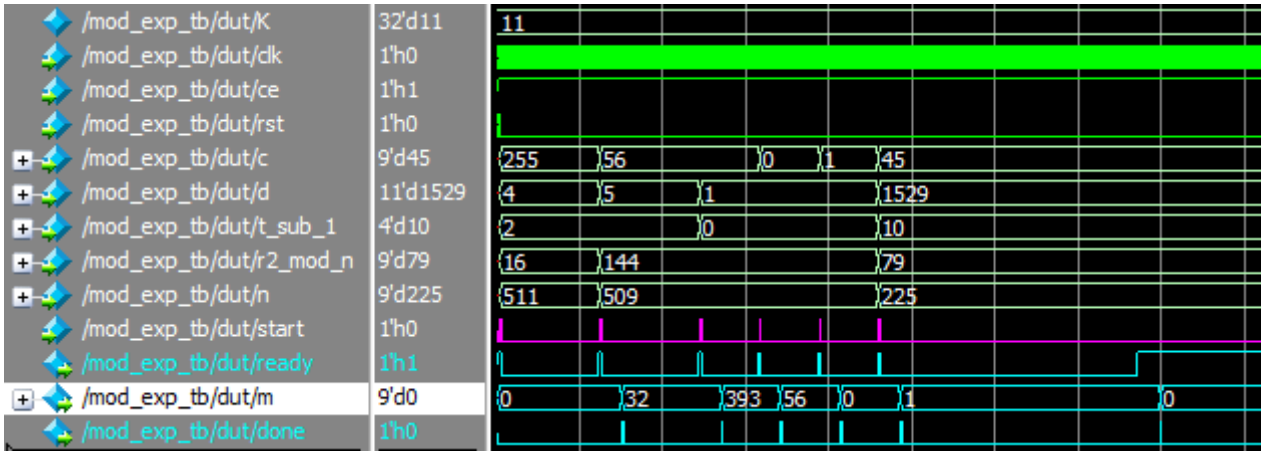


Рис. 4 – Результаты моделирования mod_exp с DATA_WIDTH = 9 в ModelSim/Questa

Дальнейшая проверка mod_exp осуществлена для DATA_WIDTH = (64+1), K = 67, что является конфигурацией для зашифрования 64-разрядных входных значений текста *c* и 65-разрядных выходных значений шифротекста m, или 65-разрядных входных значений шифротекста c и 65-разрядных (или 64-разрядных для частного случая) выходных значений текста m.

Для верификации mod_exp для DATA_WIDTH = (64+1) были проверены следующие вычисления:

1. m = (64'h_FFFF_FFFF_FFFF_FFFF ^ 2'd_3) mod 65'h_1_FFFF_FFFF_FFFF_FFFF = 65'h_1_BFFF_FFFF_FFFF_FFFF;

2. m = (64'h_FBFF_FAFF_FFFC_FF3F ^ 3'd_5) mod 65'h_1_FFBF_FFAF_FFFF_FCFF = 65'h_0_7510_1EEA_011D_4B47;

3. m = (64'h_0FFA_34FF_FF4C_F230 ^ 2'd_3) mod 65'h_1_A5FF_F6FF_BF1F_FFC1 = 65'h_1_1133_288D_E07E_7C07;

4. m = (64'h_BAAD_F00D_BAAD_F00D ^ 2'd_3) mod 65'h_1_1CEB_00DA_1CEB_00DD = 65'h_0_57DF_720D_D7B9_B1F6;

5. m = (65'h_1_FFFF_CAFE_D00D_FFFE ^ 2'd_3) mod 65'h_1_FFFF_CAFE_D00D_FFFF = 65'h_1_FFFF_CAFE_D00D_FFFE;

6. m = (64'h_DEAD_FEED_DEAD_FEED ^ 3'd_4) mod 65'h_1_FACE_FEED_FACE_FEED = 65'h_1_525F_7F69_9561_2A53;

7. m = (64'h_000F_F1CE_000F_F1CE ^ 2'd_3) mod 65'h_1_1BAD_B002_1BAD_B001 = 65'h_0_818E_AFD0_818E_AFD0;

8. m = (64'h_1010_1010_1010_1010 ^ 2'd_3) mod 65'h_1_0101_0101_0101_0101 = 65'h_0_F101_0101_0101_0101;

9. m = (64'h_0_0000_00FF_0FF1_01F1 ^ 3'd_5) mod 65'h_0_0000_01F1_A1F1_A1F1 = 65'h_0_0000_00DD_239F_DBE5;

10. m = (64'h_00BA_B10C_FEE1_FA11 ^ 3'd_7) mod 65'h_1_CAFE_BABE_BADD_BEAF = 65'h_1_72AC_7839_8A86_C805.

Все результаты m работы mod_exp сошлись с расчётными, как показано на рис. 5, 6, 7 (файл mod_exp_tb_2.sv).

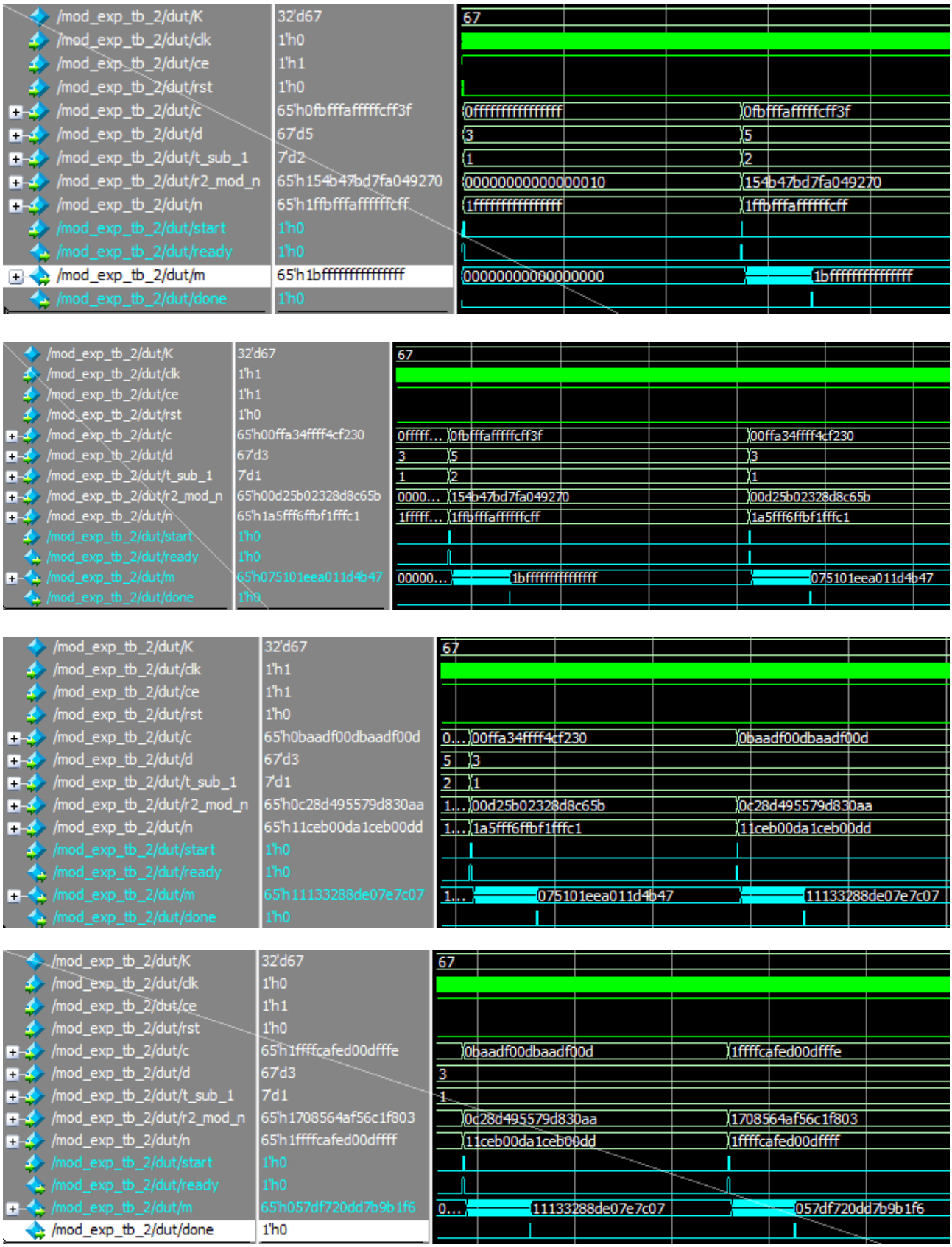


Рис. 5 – Результаты моделирования mod_exp с DATA_WIDTH = (64+1) в ModelSim/Questa (1)

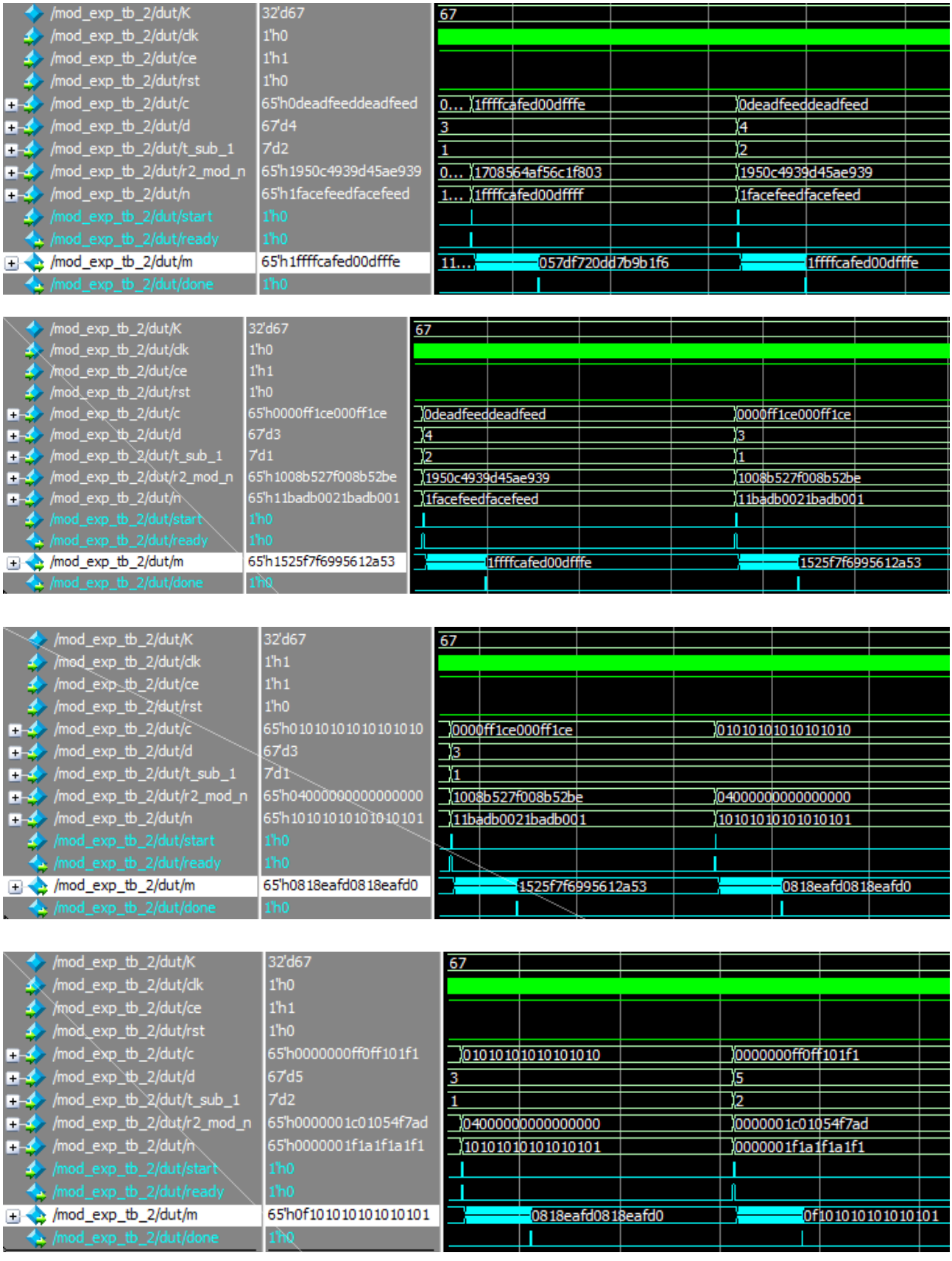


Рис. 6 – Результаты моделирования mod_exp с DATA_WIDTH = (64+1) в ModelSim/Questa (2)

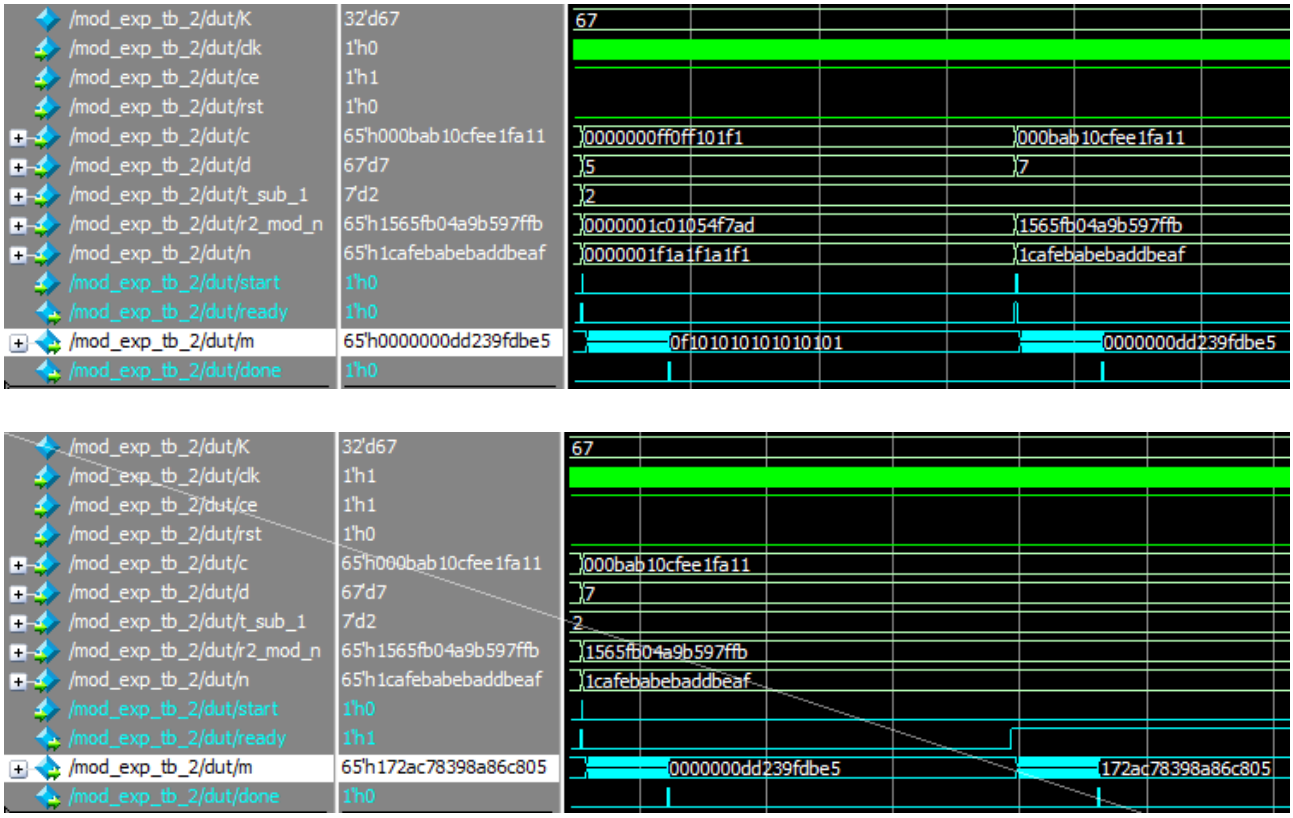


Рис. 7 – Результаты моделирования mod_exp с DATA_WIDTH = (64+1) в ModelSim/Questa (3)

Автоматизированная верификация компонента mod_exp средствами System Verilog в ModelSim/Questa

Тестовые последовательности, указанные в предыдущих разделах, имеют существенное ограничение: данные тесты хоть и необходимы на начальном этапе отладки, но тестовое покрытие по входным данным (coverage) для них незначительно, а значит после успешного прохождения этих тестов компонент ещё нельзя считать доверительно-верифицированным. Дальнейшая верификация требует создания автоматизированного генератора тестовых последовательностей, использующего методологию ограниченно-случайной верификации (Constrained Random Verification). Такой тест-генератор был описан в файле mod_exp_tb_3.sv.

При автоматической генерации тестовых последовательностей теоретически можно добиться стопроцентного покрытия по входным данным, но на практике при верификации компонента в симуляторе достижение стопроцентного покрытия может потребовать огромного ресурса вычислительного времени. В таком случае выбирается некоторое доверительное тестовое покрытие, или, другими словами, некоторое доверительное число прохождений тестов со случайными наборами входных данных, но при этом должно быть обеспечено качество такого набора тестов, т. е. наличие в этом наборе всех ключевых граничных случаев. После прохождения тестового набора, удовлетворяющего вышеуказанным требованиям, компонент по договорённости считается прошедшим процесс верификации на этапе симуляции.

Основным инструментом описываемого автоматизированного теста (mod_exp_tb_3.sv) является класс mod_exp_dataset_c. Данный класс предназначен, в частности, для генерации случайных (rand) входных данных для mod_exp (rand c, rand d, rand n) согласно указанным ограничениям (constraint c1), а также для расчёта входных данных для mod_exp, зависящих от c, d, n (r2_mod_n, t_sub_1). Класс mod_exp_dataset_c также содержит метод для расчёта эталонного model_m = (c ^ d) mod n.

Метод calc_r2_mod_n() предназначен для генерации r2_mod_n для случайно сгенерированного ранее n. Метод calc_t_sub_1() генерирует t_sub_1 по случайно сгенерированному ранее d.

Метод calc_model_m() предназначен для расчёта эталонного model_m: model_m = (c ^ d) mod n, причём для расчёта model_m используется классический (и менее быстродействующий) метод возведения в степень по модулю по схеме справа-налево (Алгоритм 0). Значение model_m используется для проверяющего сопоставления model_m и m, рассчитанного тестируемым mod_exp. Ввиду того, что быстродействующий алгоритм Монтгомери возведения в степень по модулю, используемый в верифицируемом компоненте mod_exp, существенно отличается от Алгоритма 0 (и даже имеет дополнительное относительно Алгоритма 0 ограничение на нечётное n), Алгоритм 0 является рациональным выбором для расчёта эталонного model_m. Это подтверждается тем, что ввиду сильных различий в реализации проверяемого и проверяющего алгоритмов [возведения в степень по модулю] наименее вероятна ситуация, при которой (m == model_m), но ни m, ни model_m не являются верным результатом вычисления (c ^ d) mod n. Таким образом, выбор различных реализаций проверяемого и проверяющего алгоритмов является одним из важнейших методов, позволяющих достичь более высокого качества верификации.

Метод display_contents() предназначен для отображения значений c, d, n, r2_mod_n, t_sub_1 и model_m в случае возникновения ошибки (m != model_m).

Метод copy(ref mod_exp_dataset_c other) предназначен для копирования атрибутов c, d, n, r2_mod_n, t_sub_1 и model_m из другого объекта класса mod_exp_dataset_c. Метод копирования необходим в ситуации, при которой нужно сохранить старые значения атрибутов объекта перед тем, как в объекте ограниченно-случайным образом обновятся атрибуты (что и происходит в описываемом тесте).

Метод post_randomize() вызывается в System-Verilog-симуляторе автоматически (неявно) после выполнения вызова mod_exp_dataset_c_object.*randomize*(). После выполнения

вызова mod_exp_dataset_c_object.*randomize*() все атрибуты объекта mod_exp_dataset_c_object, помеченные как рандомизируемые (rand) получают новые случайные значения (с равномерным распределением), но при этом эти случайные значения обязательно удовлетворяют всем заданным ограничениям (constraints). В классе mod_exp_dataset_c метод post_randomize() функционально предназначен для расчёта r2_mod_n и t_sub_1, зависящих от рандомизируемых c, d, n, а также для расчёта model_m, также определяемого через ограниченно-случайные c, d, n.

Группа *функционального* покрытия cg_all_valid_mod_exp_inputs класса mod_exp_dataset_c предназначена для автоматизированного подсчёта покрытия списка всех допустимых различных входных данных для mod_exp. Число корзин покрытия $N \approx ((2^{DATA\_WIDTH})^3 / 4)$, описанное таблицей пересечений valid_c_d_n_cross в cg_all_valid_mod_exp_inputs, соответствует списку всех допустимых различных входных данных для mod_exp.

Автоматизированный тест для верификации mod_exp записан в двух процессах тестирующего компонента module mod_exp_tb_3.

Процесс initial auto_test подаёт ограниченно-рандомизированные входные данные на mod_exp, после чего процесс ожидает готовности mod_exp получать новые входные данные (1 == ready). После (1 == ready) тест сохраняет текущие значения входного тестового набора (prev_mod_exp_dataset.copy(mod_exp_dataset)), так как готовность принимать новые данные в mod_exp наступает чуть раньше, чем готов результат m (но результат m готов всегда раньше, чем настанет очередная готовность принимать входные данные). Затем цикл for переходит на следующую итерацию, в начале которой снова генерируются новые входные данные (mod_exp_dataset.*randomize*()).

Процесс always check_m ожидает (1 == done), после чего сравнивает m и prev_mod_exp_dataset.model_m: в случае ошибки (m != model_m) симуляция приостанавливается и процесс дополнительно отображает текущие значения входных данных, для которых возникла ошибка.

Число ошибок содержится в счётчике number_of_run_errors: это число выводится процессом initial auto_test по окончании всех тестов.

Автоматизированный тест настраивается двумя параметрами: DATA_WIDTH (задаёт размерность c, d, n; должно выполняться DATA_WIDTH >= 4) и NUMBER_OF_TEST_RUNS (задаёт число генерируемых тестов или, другими словами, число итераций).

Особый интерес с точки зрения практической верификации mod_exp представляют тесты с небольшим DATA_WIDTH и большим NUMBER_OF_TEST_RUNS. Это обусловлено следующими соображениями. Во-первых, для небольших DATA_WIDTH время выполнения одной итерации относительно небольшое, а, значит, можно за короткое время выполнить очень большое число итераций. Во-вторых, большое число итераций означает большее покрытие всех возможных входных тестовых данных. В-третьих, ввиду небольших и одинаковых разрядностей у c, d, n, обусловленных малым DATA_WIDTH, число N возможных отличных друг от друга входных наборов данных относительно невелико (по сравнению с числом различных тестовых наборов входных данных для больших DATA_WIDTH, т. к. при DATA_WIDTH-разрядных c, d, n число $N \sim (2^{DATA\_WIDTH})^3$ без учёта поправок на (c < n) и нечётное n, или $N \approx ((2^{DATA\_WIDTH})^3 / 4)$). Следовательно, для малых значений DATA_WIDTH с большими значениями NUMBER_OF_TEST_RUNS в качестве настроек автоматизированного теста, можно на практике за короткое время выполнения обеспечить более полное покрытие (или вообще стопроцентное покрытие), а, значит, и сгенерировать большее число редких граничных случаев входных данных, для которых вероятны ошибки работы mod_exp. Здесь особенно интересны те граничные наборы входных данных, которые могут выявить ошибки, не обусловленные только разрядностью mod_exp. Таким образом, вместе с верификацией mod_exp для целевого (и, возможно, очень большого)

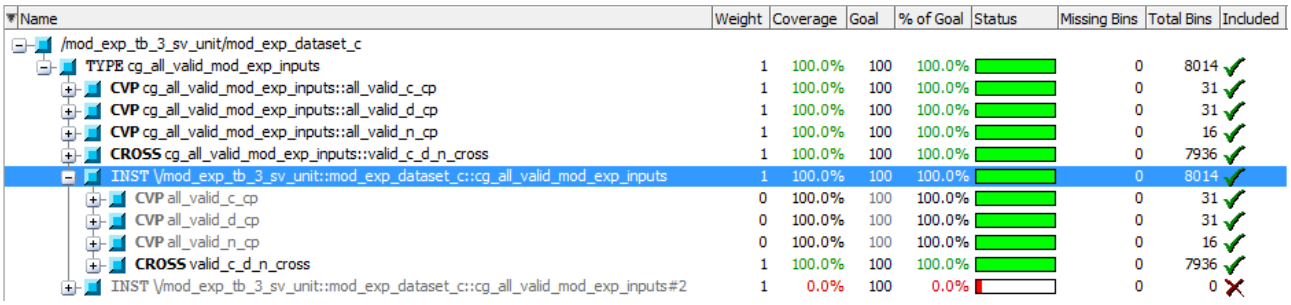
DATA_WIDTH имеет смысл предварительно осуществить автоматизированную верификацию mod_exp с очень большим числом итераций NUMBER_OF_TEST_RUNS для малого DATA_WIDTH.

Верификация компонента mod_exp была осуществлена для различных DATA_WIDTH и NUMBER_OF_TEST_RUNS.

Сначала была пройдена верификация для DATA_WIDTH = (4+1) и NUMBER_OF_TEST_RUNS = 100'000. Тест был пройден без ошибок:

```
[...]
# test #           0
# test #           1
# test #           2
[...]
# test #       99991
# test #       99992
# test #       99993
# test #       99994
# test #       99995
# test #       99996
# test #       99997
# test #       99998
# test #       99999
# TEST_FINISHED
# NUMBER_OF_RUN_ERRORS_OCCURED:           0
# Break in NamedBeginStat auto_test at .../RSA/Project/src/tb/mod_exp_tb_3.sv
line 258
# Simulation Breakpoint: Break in NamedBeginStat auto_test at
.../Project/src/tb/mod_exp_tb_3.sv line 258
# MACRO ...\Project\run_mod_exp_tb_3_no_wave.do PAUSED at line 6
[...]
```

По окончании теста для DATA_WIDTH = (4+1) по входным данным было обеспечено покрытие 100% (рис. 8).



Name	Weight	Coverage	Goal	% of Goal	Status	Missing Bins	Total Bins	Included
/mod_exp_tb_3_sv_unit/mod_exp_dataset_c								
TYPE cg_all_valid_mod_exp_inputs	1	100.0%	100	100.0%		0	8014	✓
CVP cg_all_valid_mod_exp_inputs::all_valid_c_cp	1	100.0%	100	100.0%		0	31	✓
CVP cg_all_valid_mod_exp_inputs::all_valid_d_cp	1	100.0%	100	100.0%		0	31	✓
CVP cg_all_valid_mod_exp_inputs::all_valid_n_cp	1	100.0%	100	100.0%		0	16	✓
CROSS cg_all_valid_mod_exp_inputs::valid_c_d_n_cross	1	100.0%	100	100.0%		0	7936	✓
INST \/mod_exp_tb_3_sv_unit::mod_exp_dataset_c::cg_all_valid_mod_exp_inputs	1	100.0%	100	100.0%		0	8014	✓
CVP all_valid_c_cp	0	100.0%	100	100.0%		0	31	✓
CVP all_valid_d_cp	0	100.0%	100	100.0%		0	31	✓
CVP all_valid_n_cp	0	100.0%	100	100.0%		0	16	✓
CROSS valid_c_d_n_cross	1	100.0%	100	100.0%		0	7936	✓
INST \/mod_exp_tb_3_sv_unit::mod_exp_dataset_c::cg_all_valid_mod_exp_inputs#2	1	0.0%	100	0.0%		0	0	✗

Рис. 8 – Подсчёт покрытия для mod_exp с DATA_WIDTH = (4+1) в ModelSim/Questa

Далее была пройдена верификация для DATA_WIDTH = (5+1) и NUMBER_OF_TEST_RUNS = 1'000'000. Тест был пройден без ошибок:

```
[...]
# test #           0
# test #           1
# test #           2
[...]
# test #      999991
# test #      999992
# test #      999993
# test #      999994
# test #      999995
# test #      999996
# test #      999997
# test #      999998
# test #      999999
# TEST_FINISHED
# NUMBER_OF_RUN_ERRORS_OCCURED:           0
# Break in NamedBeginStat auto_test at .../src/tb/mod_exp_tb_3.sv line 258
# Simulation Breakpoint: Break in NamedBeginStat auto_test at
.../Project/src/tb/mod_exp_tb_3.sv line 258
# MACRO ...\run_mod_exp_tb_3_no_wave.do PAUSED at line 6
[...]
```

По окончании теста для DATA_WIDTH = (5+1) по входным данным было обеспечено покрытие 100% (рис. 9).

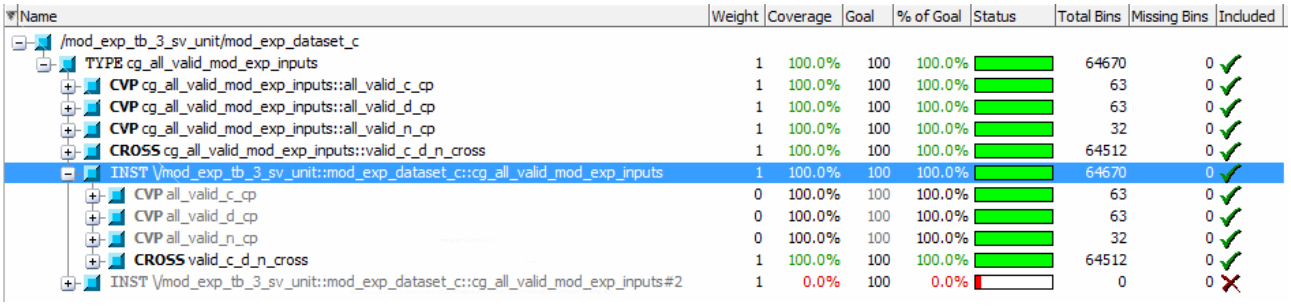


Рис. 9 – Подсчёт покрытия для mod_exp с DATA_WIDTH = (5+1) в ModelSim/Questa

Ввиду ограничений симулятора Questa Sim -64 10.2c дальнейший подсчёт покрытия для DATA_WIDTH > (5+1) оказался технически невозможен, так как число корзин покрытия (bins) для таких DATA_WIDTH оказалось слишком велико.

Далее была пройдена верификация для DATA_WIDTH = (8+1) и NUMBER_OF_TEST_RUNS = 10'000'000. Тест был пройден без ошибок:

```
[...]
# test #           0
# test #           1
# test #           2
[...]
# test #     9999991
# test #     9999992
# test #     9999993
# test #     9999994
# test #     9999995
# test #     9999996
# test #     9999997
# test #     9999998
# test #     9999999
# TEST_FINISHED
# NUMBER_OF_RUN_ERRORS_OCCURED:           0
# Break in NamedBeginStat auto_test at .../src/tb/mod_exp_tb_3.sv line 215
# Simulation Breakpoint: Break in NamedBeginStat auto_test at
.../src/tb/mod_exp_tb_3.sv line 215
# MACRO ...\run\run_mod_exp_tb_3_no_wave.do PAUSED at line 6
[...]
```

Далее была пройдена верификация для DATA_WIDTH = (64+1) и NUMBER_OF_TEST_RUNS = 1'000'000. Тест был пройден без ошибок:

```
[...]
# test #           0
# test #           1
# test #           2
[...]
# test #      999989
# test #      999990
# test #      999991
# test #      999992
# test #      999993
# test #      999994
# test #      999995
# test #      999996
# test #      999997
# test #      999998
# test #      999999
# TEST_FINISHED
# NUMBER_OF_RUN_ERRORS_OCCURED:           0
# Break in NamedBeginStat auto_test at .../src/tb/mod_exp_tb_3.sv line 215
# Simulation Breakpoint: Break in NamedBeginStat auto_test at
.../Project/src/tb/mod_exp_tb_3.sv line 215
# MACRO ...\run\run_mod_exp_tb_3_no_wave.do PAUSED at line 6
[...]
```

Автоматизированные тесты запускались также с параметрами DATA_WIDTH = 257, 513, 1025, 2049, 4097. Успешное прохождение ряда итераций для каждого из указанных DATA_WIDTH говорит о работоспособности параметризуемого System-Verilog-описания компонента mod_exp для любых актуальных размерностей входных данных.

Очевидно, что число итераций, необходимых для полного покрытия по входным данным, возрастает с увеличением DATA_WIDTH, так как для б***о***льших DATA_WIDTH число всех возможных входных данных $N \approx ((2^{DATA\_WIDTH})^3 / 4)$ возрастает. Время выполнения каждой итерации также возрастает с увеличением DATA_WIDTH. Таким образом, для больших DATA_WIDTH обеспечение хотя бы (предполагаемо) нескольких процентов покрытия по входным данным уже занимает огромное время выполнения симуляции, а это значит, что на практике полная верификация mod_exp для больших DATA_WIDTH невозможна только средствами ModelSim/Questa.

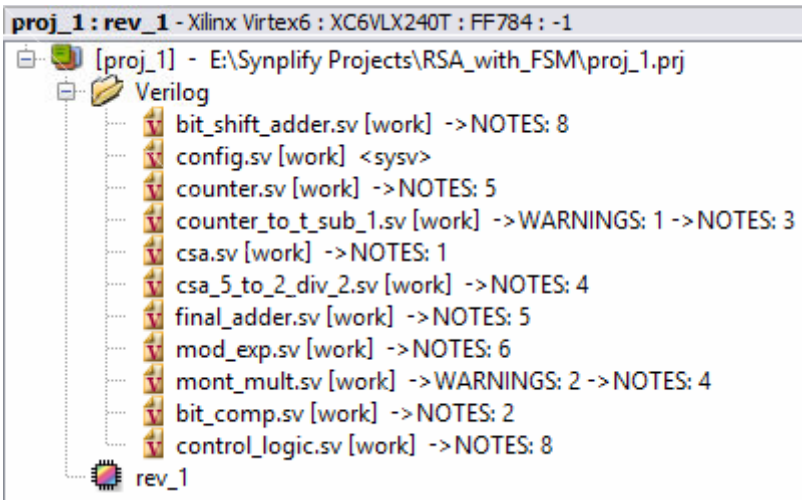
Ввиду того, что вышеуказанные автоматизированные тесты были пройдены без ошибок, можно заключить, что:

– компонент mod_exp можно считать ***полностью*** *верифицированным для ((3+1) ≤ DATA_WIDTH ≤ (5+1));*

– компонент mod_exp допустимо считать ***доверительно-верифицированным*** *для всех допустимых DATA_WIDTH.*

Результаты синтеза для mod_exp в Synopsys Synplify для FPGA

Для синтеза модуля mod_exp была использована САПР Synopsys Synplify Pro G-2012.09. Целевой FPGA для синтеза была выбрана Xilinx Virtex 6 : XC6VLX240T : FF784 : -1. Результаты синтеза для mod_exp #(parameter DATA_WIDTH = (`CONFIG_DATA_WIDTH)) показаны на рис. 10, 11, 12.



Project Settings

Project Name	proj_1	Implementation Name	rev_1
Top Module	[auto]	Pipelining	0
Retiming	0	Resource Sharing	0
Use Xilinx Xflow	0	Fanout Guide	10000
Disable I/O Insertion	0	Disable Sequential Optimizations	0
Clock Conversion	1	Use Xilinx Partition Flow	0

Run Status

Job Name	Status				CPU Time	Real Time	Memory	Date/Time
Compile Input Detailed report	Complete	16	0	0	-	0m:09s	-	15.02.2016 14:14:54
Premap Detailed report	Complete	3	1	0	0m:08s	0m:08s	311MB	15.02.2016 14:15:04
Map & Optimize Detailed report	Complete	143	6	0	03m:07s	03m:08s	497MB	15.02.2016 14:18:12

Area Summary

I/O ports	5142	Non I/O Register bits	26139 (8%)
I/O Register bits	0	Block Rams	0 (416)
DSP48s	0 (768)	LUTs	23859 (15%)
Detailed report		Hierarchical Area report	

Timing Summary

Clock Name	Req Freq	Est Freq	Slack
mod_exp|clk	500.0 MHz	309.7 MHz	-1.229
Detailed report			

Optimizations Summary

Combined Clock Conversion	1 / 0 more

Рис. 10 – Результаты синтеза mod_exp с (`CONFIG_DATA_WIDTH = 1024+1) для Xilinx Virtex 6 : XC6VLX240T : FF784 : -1

Project Settings

Project Name	proj_1	Implementation Name	rev_1
Top Module	[auto]	Pipelining	0
Retiming	0	Resource Sharing	0
Use Xilinx Xflow	0	Fanout Guide	10000
Disable I/O Insertion	0	Disable Sequential Optimizations	0
Clock Conversion	1	Use Xilinx Partition Flow	0

Run Status

Job Name	Status				CPU Time	Real Time	Memory	Date/Time
Compile Input Detailed report	Complete	16	0	0	-	0m:02s	-	15.02.2016 14:22:40
Premap Detailed report	Complete	3	1	0	0m:01s	0m:01s	220MB	15.02.2016 14:22:43
Map & Optimize Detailed report	Complete	156	4	0	01m:09s	01m:09s	428MB	15.02.2016 14:23:53

Area Summary

I/O ports	2581	Non I/O Register bits	13357 (4%)
I/O Register bits	0	Block Rams	0 (416)
DSP48s	0 (768)	LUTs	10748 (7%)
Detailed report		Hierarchical Area report	

Timing Summary

Clock Name	Req Freq	Est Freq	Slack
mod_exp|clk	500.0 MHz	372.4 MHz	-0.685
Detailed report			

Optimizations Summary

Combined Clock Conversion	1 / 0 more

Рис. 11 – Результаты синтеза mod_exp с (`CONFIG_DATA_WIDTH = 512+1) для Xilinx Virtex 6 : XC6VLX240T : FF784 : -1

Project Settings

Project Name	proj_1	Implementation Name	rev_1
Top Module	[auto]	Pipelining	0
Retiming	0	Resource Sharing	0
Use Xilinx Xflow	0	Fanout Guide	10000
Disable I/O Insertion	0	Disable Sequential Optimizations	0
Clock Conversion	1	Use Xilinx Partition Flow	0

Run Status

Job Name	Status				CPU Time	Real Time	Memory	Date/Time
Compile Input Detailed report	Complete	16	0	0	-	0m:01s	-	15.02.2016 14:29:18
Premap Detailed report	Complete	3	1	0	0m:00s	0m:00s	199MB	15.02.2016 14:29:20
Map & Optimize Detailed report	Complete	54	4	0	0m:14s	0m:15s	234MB	15.02.2016 14:29:35

Area Summary

I/O ports	338	Non I/O Register bits	1810 (0%)
I/O Register bits	0	Block Rams	0 (416)
DSP48s	0 (768)	LUTs	1517 (1%)
Detailed report		Hierarchical Area report	

Timing Summary

Clock Name	Req Freq	Est Freq	Slack
mod_exp|clk	500.0 MHz	406.3 MHz	-0.461
Detailed report			

Optimizations Summary

Combined Clock Conversion	1 / 0 more

Рис. 12 – Результаты синтеза mod_exp с (`CONFIG_DATA_WIDTH = 64+1) для Xilinx Virtex 6 : XC6VLX240T : FF784 : -1

Для полученных результатов для FPGA Xilinx Virtex 6 : XC6VLX240T : FF784 : -1 можно рассчитать максимальную пропускную способность компонента mod_exp. Учитывая, что для поточного (непрерывного) режима скорость шифрования очередного блока будет определяться скоростью отработки ступени конвейера mod_exp с блоками MontMult, то из Алгоритма 4 можно выразить необходимое число тактов для одного зашифрования/расшифрования как N = (k+1)*(t+2), где k+1 – число тактов, необходимое для вычисления MontMult с полной разрядностью n равной k, а t – эффективное число разрядов в d.

Важно отметить, что для каждого нового значения t число N будет отличаться, поэтому расчёт проведён для одного из типовых значений d, а именно для *d = 65537, для которого t = 17*.

Для процессора RSA с **`CONFIG_DATA_WIDTH = 1024+1**: k = 2+1024+1, N = (2+1024+1+1)*(17+2) = 19532. Имея полученную максимальную частоту 309,7 МГц, пропускная способность v = 309,7 * 1024 / 19532 = **16,237 Мбит/с**.

Для процессора RSA с **`CONFIG_DATA_WIDTH = 512+1**: k = 2+512+1, N = (2+512+1+1)*(17+2) = 9804. Имея полученную максимальную частоту 372,4 МГц, пропускная способность v = 372,4 * 512 / 9804 = **19,448 Мбит/с**.

Для процессора RSA с **`CONFIG_DATA_WIDTH = 64+1**: k = 2+64+1, N = (2+64+1+1)*(17+2) = 1292. Имея полученную максимальную частоту 406,3 МГц, пропускная способность v = 406,3 * 64 / 1292 = **20,126 Мбит/с**.

Проектирование подсистемы отключения питания для процессора RSA с использованием средств UPF

Ввиду того, что использование компонента процессора шифрования RSA предполагается в системе на кристалле широкого назначения, в которой процессор RSA будет использоваться не во всех режимах работы, данный модуль процессора шифрования имеет смысл реализовать с функцией отключения питания. Проектирование такой функциональности предполагает дополнительное определение компонентов подсистемы питания, таких как схема управления питанием (на RTL) и не-RTL элементы, такие, как порты и цепи питания, блоки изоляции и ключ выключения и включения питания (см. «Проектирование цифровых систем с пониженным энергопотреблением с применением технологии UPF описания подсистемы питания», Журнал Информатика №3(47) июль-сентябрь 2015, Объединенный институт проблем информатики НАН Беларуси, Минск).

Процессор шифрования RSA с возможностью отключения питания описан в двух дополняющих друг друга файлах: rsa_top.sv (System Verilog RTL) и rsa_top_power_intent.upf (UPF). Файл tb\rsa_top_tb.sv содержит тест, проверяющий работоспособность включения и выключения питания модуля rsa_top. Симуляция проекта проводилась средствами Mentor Graphics Questa Sim -64 10.2c в режиме Power-Aware Simulation (данный режим не поддерживается в Model Sim). Использованные сценарии запуска симуляции содержатся в файлах run_rsa_top_tb_opt.do и run_rsa_top_tb.do. Укрупнённая схема модуля rsa_top, полученная средствами Questa Sim, показана на рис. 13 (цветовое кодирование схемы позволяет различать элементы, находящиеся в разных доменах питания).

RTL-файл rsa_top.sv, описывающий модуль rsa_top содержит компонент mod_exp_0 (являющийся ядром процессора шифрования) в качестве основного и одновременно в качестве единственного отключаемого от питания компонента. В свою очередь файл rsa_top_power_intent.upf определяет компонент mod_exp_0 файла rsa_top.sv как единственный элемент, находящийся в отключаемом домене питания PD_sw (на рис. 13 закрашено фиолетовым); все остальные элементы модуля rsa_top определены как находящиеся в домене PD_top (на рис. 13 синий цвет), который является не отключаемым со стороны rsa_top. Отключение домена PD_sw со стороны rsa_top осуществляется по сигналу w_sw_disable (определён в rsa_top.sv). В файле rsa_top_power_intent.upf данный сигнал подключён как управляющий для ключа питания SW. Асинхронный сигнал w_sw_ack, уведомляющий о включении или выключении SW, принимается в rsa_top.sv через синхронизирующую цепочку (сигнал d2_w_sw_ack) для предотвращения метастабильности в основной части схемы.

Ввиду того, что mod_exp_0 является отключаемым от питания, на схеме установлен блок выключения сигнала синхронизации на основе защёлки (выходной сигнал clk_gated, сигнал управления выключением clk_disable), который запрещает подачу синхросигнала в период, когда mod_exp_0 выключен. Важно отметить, что подача сигнала отключения синхросигнала (mod_exp_0.ce = 0) в период выключенного питания не имеет никакого воздействия, так как mod_exp_0 отключается от питания целиком, что и обуславливает необходимость внешнего блока выключения сигнала синхронизации.

С целью предотвращения передачи неопределённых случайных выходных сигналов из отключённого домена питания PD_sw во включённые домены питания, файл rsa_top_power_intent.upf задаёт стратегию автоматической расстановки блоков изоляции (сигнал включения по уровню w_iso_en, выходной сигнал изолированного сигнала 0). Таким образом, описанный в файле rsa_top.sv сигнал w_iso_en подключается в файле rsa_top_power_intent.upf как сигнал включения изоляции. После применения UPF-стратегии изоляции RTL-сигналы w_ready, w_m, w_done автоматически получают блоки изоляции (на рис. 13 обозначены бирюзовым цветом). Так как сигнал включения изоляции не должен изменять состояние сигналов выхода изоляции, компонент mod_exp_0 по сигналу w_outputs_force_zero форсирует значения выходных сигналов mod_exp_0 в 0 ещё до включения изоляции (компонент mod_exp_wrapper является обёрткой mod_exp, где сигнал

w_outputs_force_zero форсирует выходные для mod_exp сигналы в 0).

Компонент rsa_power_control_0 содержит конечный автомат, который обеспечивает правильную последовательность сигналов управления питанием во времени (это сигналы w_outputs_force_zero, w_iso_en, w_sw_disable, w_reset_on_enable, w_clk_disable). Входные сигналы req_disable и req_enable являются внешними командами для смены режима питания, а сигнал w_mode_change_ack сигнализирует об окончании процесса смены режима питания.

Внутренний сигнал power_enabled модуля rsa_top используется для формирования сигнала ready, который при отключённом питании для mod_exp_0 должен быть форсирован в 0.

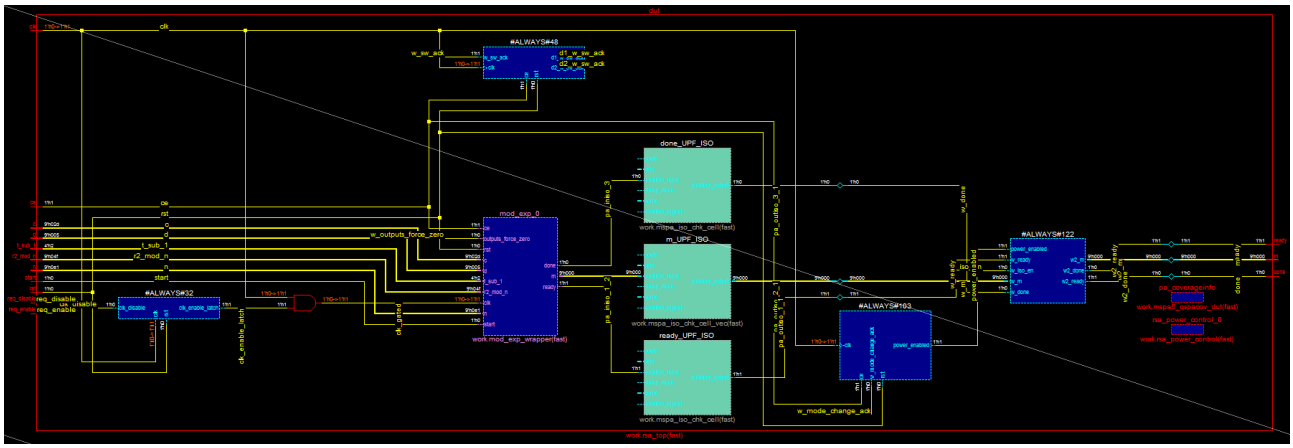
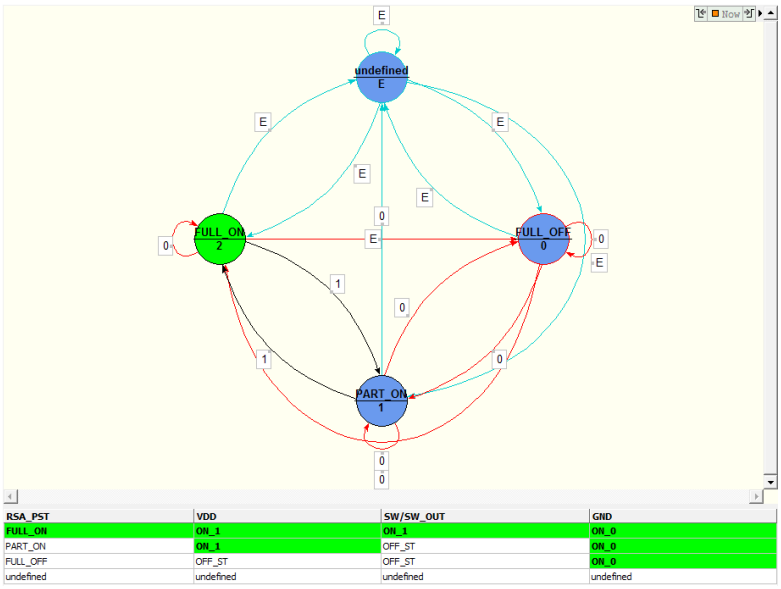


Рис. 13 – Укрупнённая схема модуля rsa_top

Для целей верификации файл rsa_top_power_intent.upf содержит таблицу всех возможных разрешённых состояний питания для rsa_top:

```
# Global power states table (for documentation and verification):
create_pst RSA_PST                -supplies {VDD     SW/SW_OUT  GND }
add_pst_state FULL_ON -pst RSA_PST  -state {ON_1    ON_1       ON_0}
add_pst_state PART_ON -pst RSA_PST  -state {ON_1    OFF_ST     ON_0}
add_pst_state FULL_OFF -pst RSA_PST -state {OFF_ST  OFF_ST     ON_0}
```

На рис 14. показана визуализация этой таблицы средствами Questa Sim.



RSA_PST	VDD	SW/SW_OUT	GND
FULL_ON	ON_1	ON_1	ON_0
PART_ON	ON_1	OFF_ST	ON_0
FULL_OFF	OFF_ST	OFF_ST	ON_0
undefined	undefined	undefined	undefined

Рис. 14 – Граф состояний питания rsa_top

Результат симуляции модуля rsa_top, описанного файлами rsa_top.sv и rsa_top_power_intent.upf с использованием теста tb\rsa_top_tb.sv и run_rsa_top_tb.do показан на рис. 15.

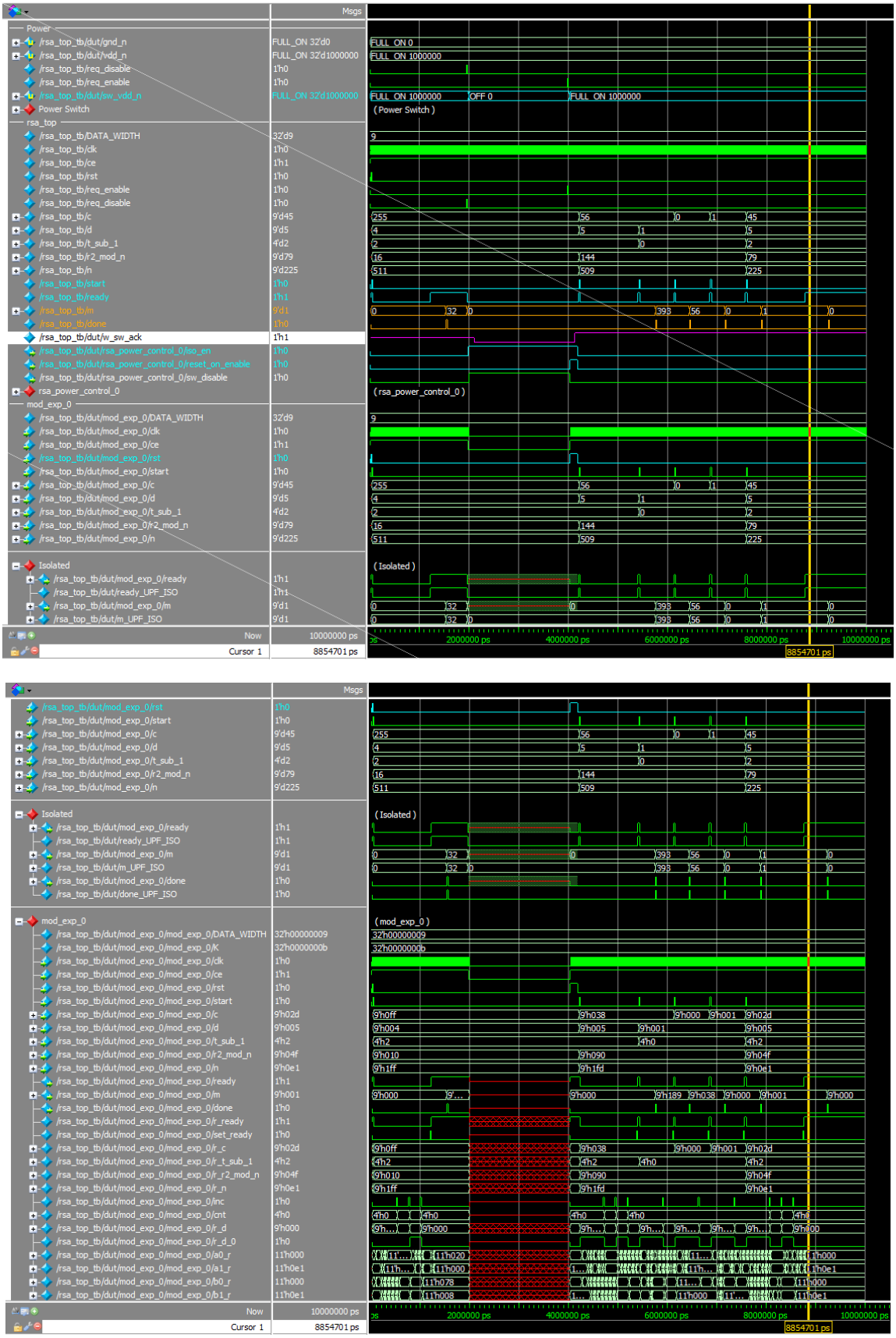


Рис. 15 – Временная диаграмма для теста tb\rsa_top_tb.sv

Целью теста tb\rsa_top_tb.sv является проверка работоспособности именно подсистемы питания, так как ядро mod_exp было верифицировано ранее. В ходе теста домен PD_sw после периода работы был выключен, а через какое-то время снова включён. Точками интереса на рис. 15 являются периоды времени симуляции в окрестностях включения и выключения питания, а также сам период выключенного питания для домена PD_sw.

Как видно на временной диаграмме, в режиме Power-Aware Simulation имеют место дополнительные обозначения, связанные с питанием. Так, сигналы *_UPF_ISO, показанные в группе Isolated, всегда имеют определённые значения, в то время как входы изоляции получают неопределённые значения ввиду отсутствия питания; но ввиду того, что данные сигналы пропускаются через блок изоляции, Questa Sim отображает для них период изоляции зелёной полупрозрачной заливкой. Те драйвера сигналов домена PD_sw, которые получают неопределённые значения ввиду отключения питания, на диаграмме также получают на период отключения питания заливку красной диагональной решёткой. Порты и цепи питания, описанные только в файле rsa_top_power_intent.upf (и не являющиеся частью RTL-описания) отображаются в симуляторе значками с буквой u (от UPF).

Одним из основных инструментов при верификации работы подсистемы питания средствами Questa Sim являются многочисленные автоматические проверки, результаты которых выводятся как в файлы отчётов (папка /report), так и в окно вывода сообщений Transcript. Сначала Questa Sim осуществляет *статические* проверки элементов подсистемы питания на этапе оптимизации (vopt). Например, блоки изоляции считаются *статически* необходимыми, если существуют сигналы между двумя доменами, один из которых (источник) *может* быть отключен, пока включён другой (необходимость определяется по таблице разрешённых состояний питания RSA_PST):

```
# -- Analyzing UPF for Power Aware Static Checks...
# -- Analysis of Power Aware Static Checks Complete.
# -- Power Aware Static Checks status:
# ** Note: (vopt-9857) [ UPF_ISO_STATIC_CHK ] Found Total 11 Valid isolation cells.
```

Динамические уведомления и проверки осуществляются уже по ходу работы симуляции:

```
# ** Note: (vsim-8902) MSPA_PD_STATUS_INFO: Time: 2000000 ps, Power domain 'PD_sw' is
powered down.
# ** Error: (vsim-PA-8908) MSPA_PD_OFF_ACT : Time: 4020000 ps,
rsa_top_tb.dut.mod_exp_0.ce toggled  during power down of power domain: PD_sw
#
# ** Error: (vsim-PA-8908) MSPA_PD_OFF_ACT : Time: 4020000 ps,
rsa_top_tb.dut.mod_exp_0.rst toggled  during power down of power domain: PD_sw
#
# ** Error: (vsim-PA-8908) MSPA_PD_OFF_ACT : Time: 4020000 ps,
rsa_top_tb.dut.mod_exp_0.outputs_force_zero toggled  during power down of power domain:
PD_sw
#
# ** Note: (vsim-8902) MSPA_PD_STATUS_INFO: Time: 4020000 ps, Power domain 'PD_sw' is
powered up.
```

Как видно из отчёта, в ходе работы теста произошли события включения и выключения питания (Notes), а также некоторые проверки завершились ошибкой (Errors). В данном случае эти проверки можно проигнорировать, так как все ошибочные события имеют метку 4020000 ps и происходят в одно и то же время [симуляции] с включением питания, но событие включения питания происходит позже по времени выполнения, из-за чего и возникает ошибка проверки. Если, например, задержать время подачи сброса rst хотя бы на один такт, то возникнут множественные ошибки вида «`# ** Error: (vsim-8912) MSPA_NRET_ASYNCFF: Time: 12 ns, Asynchronous(set/reset) control for the following flop(s) of power domain 'PD1' is not asserted at power up: # /tb/top_inst/out1. # File: test.upf, Line: 4, Power Domain:PD1`».

Таким образом, с использованием UPF-описания компонентов подсистемы питания с использованием средств Questa Sim была осуществлена логическая симуляция и верификация RTL-логики отключения питания для модуля rsa_top. Дальнейшая разработка полученного смешанного RTL/UPF-описания предполагает отображение абстрактных моделей ключа питания и блоков изоляции на конкретные библиотечные блоки, после чего может быть осуществлено проектирование топологии части системы на кристалле, реализующей процессор RSA.