Network Working Group
Request For Comments: 1936
Category: Informational

J. Touch
B. Parham
ISI

April 1996

Implementing the Internet Checksum in Hardware

Аппаратная реализация расчета контрольных сумм Internet

Статус документа

Этот документ является информационным и не содержит каких-либо стандартов Internet. Документ можно распространять без ограничений.

Аннотация

В документе представлен эффективный метод аппаратного расчета контрольных сумм Internet и представлен код PLD для программирования одного недорогого элемента расчета с производительностью 1,26 Гбит/с.

Введение

Контрольные суммы (Internet Checksum) применяются в различных протоколах Internet для контроля целостности данных в заголовках (например, IP) [4] и теле пакетов (например, UDP, TCP) [5][6]. Эффективная программная реализация контрольных сумм предложена в нескольких RFC [1][2][3][7].

Эффективная программная реализация алгоритма Internet Checksum часто встраивается в операции копирования (раздел 2 в [1]). Эта операция будет все чаще выполняться выделенным оборудованием прямого доступа к памяти (DMA[1]), поэтому в устройства DMA начали начали включать специальные блоки для расчета Internet Checksum в процессе переноса данных.

В этом документе представлена архитектура эффективного, конвейерного механизма расчета контрольных сумм Internet Checksum, подходящего для включения в оборудование DMA [8]. Это можно реализовать в сравнительно недорогих программируемых логических устройствах PLD[2] ($40 в 1995 г.), поддерживающих скорость передачи 1,26 Гбит/с (26 нсек на 32-битовое слово). В Приложении A представлен псевдокод для такого устройства. Предложенная модель была реализована на оборудовании хост-интерфейса PC-ATOMIC [8]. Предполагается, что это решение подойдет для общего применения сообществом Internet.

Остальная часть документа включает перечисленные ниже разделы.

Обзор контрольных сумм Internet

Сложение дополнений до 1 и до 2

Интерфейсы

Заключение

Приложение A. Исходный код PLD

Обзор контрольных сумм Internet

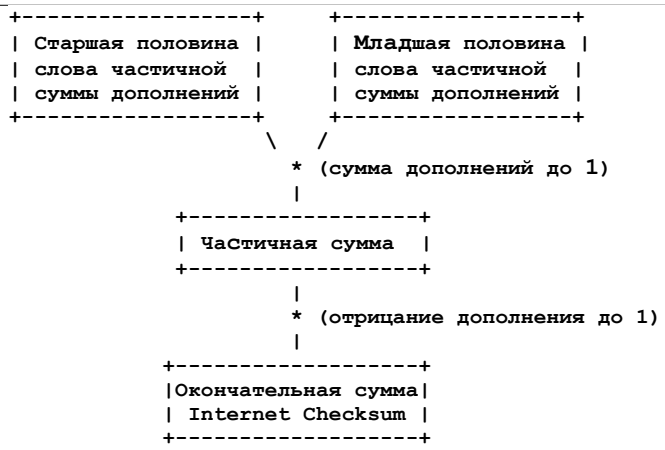
Контрольные суммы Internet служат для обнаружения повреждений блока данных [1]. Сумма инициализируется значением 0 и рассчитывается как дополнение суммы дополнений до 1 для 16-битовых блоков. При отсутствии ошибок сложение полученной контрольной суммы с рассчитанной заново должно давать в результате 0.

Контрольная сумма обладает рядом свойст [1]:

- смена порядка байтов на выходе эквивалентна смене порядка байтов на входе;
- смена порядка 16-битовых слов не меняет результат;
- возможен инкрементный расчет;
- возможен отложенный перенос;
- параллельное сложение отложенных переносов, инкрементный расчет и независимость от порядка 16-битовых слов.

В документе описана реализация с параллельным расчетом двух частичных контрольных сумм, в одной учитываются четные, а в другой нечетные половины 32-битовых блоков данных. Результатом является пара частичных контрольных сумм (четная и нечетная), которые можно объединить, а результат инвертировать для создания корректного значения Internet Checksum. Метод связан с параллельным суммированием длинных слов (long-word), используемым в эффективной программной реализации [1].

[1]Direct memory access.
[2]Programmable logic device.



Сложение дополнений до 1 и до 2

Для расчета Internet Checksum используется сумматор дополнений до 1 и побитовый инвертор. Сумматор дополнений можно создать естественным путем или на основе компонент с дополнением до 2.

Реализация сумматора дополнений до 1 на основе дополнения до 2 требует двух сумматоров с дополнением до 2 или двух циклов на операцию сложения. Выполняется суммирование, затем перенос старшего бита передается на вход переноса и выполняется второе суммирование (дополнение до 1 обозначено {+1s}, а дополнение до 2 - {+2s}).

```
a {+1s} b == (a {+2s} b) + carry(a {+2s} b)
```

Например,

```
halfword16 a,b;
word32 c;
a {+1s} b == r
```

так что

```
c = a {+2s} b;                          # значение суммы
r = (c & 0xFFFF) {+2s} (c >> 16);       # перенос суммы
```

Сумматор с дополнением до 2 сложнее с учетом переноса (операция для каждой строки, где есть AND или XOR {^})

4-битовый сумматор дополнений до 2 с переносом имеет вид[1]

```
a,b : входные данные
p   : распространение переноса, где pi = ai*bi = (ai)(bi)
g   : генерация переноса, где gi = ai + bi

Out0 := a0 ^ b0 ^ ci
Out1 := a1 ^ b1 ^ (cip0     + g0)
Out2 := a2 ^ b2 ^ (cip0p1   + g0p1   + g1)
Out3 := a3 ^ b3 ^ (cip0p1p2 + g0p1p2 + g1p2 + g2)
Cout := cip0p1p2p3 + g0p1p2p3 + g1p2p3 + g2p3 + g3
```

Корректный сумматор с переносом вперед и дополнением до 1 принимает, что перенос сложения дополнений до 2 эквивалентен «тороидальному» (кольцевому) переносу. Биты сумматора с с переносом вперед и дополнением до 1 имеют ту же комплексность, что и старший бит сумматора с дополнением до 2. Таким образом, сумма дополнений до 1 (и контрольная сумма Internet) не зависят от позиции битов. Была произведена замена ci на co и сокращение (элементы OR в каждой паре строк).

4-битовый кольцевой сумматор дополнений до 1 с переносом имеет вид

```
Out0 = a0 ^ b0 ^ (g3       + g2p3     + g1p2p3   + g0p1p2p3)
Out1 = a1 ^ b1 ^ (g3p0     + g2p3p0   + g1p2p3p0 + g0)
Out2 = a2 ^ b2 ^ (g3p0p1   + g2p3p0p1 + g1       + g0p1)
Out3 = a3 ^ b3 ^ (g3p0p1p2 + g2       + g1p2     + g0p1p2)
```

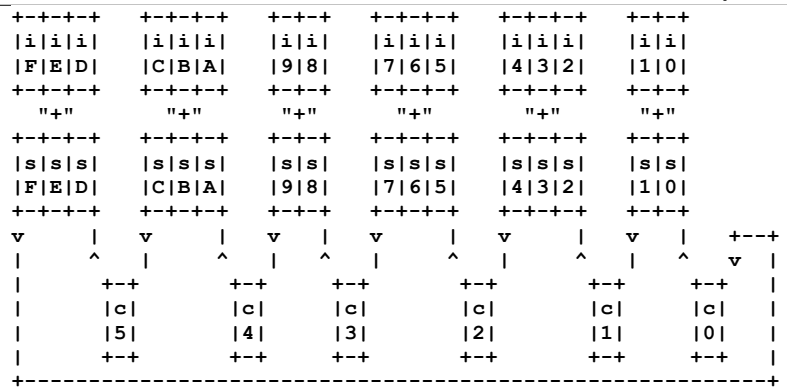
Аппаратная реализация может применять эту «тороидальную» конструкцию напрямую вместе с традиционными быстрыми сумматорами с дополнением до 2 для реализации конвейерного сумматора с дополнением до 1 [8].

Реализация VLSI может использовать любой сумматор с полным просмотром вперед, адаптированный для кольцевого режима и эквивалентности битов, как отмечено выше. В предложенной реализации PLD используются сумматоры на основе 2- и 3-битовых компонентов, которые соединяются в кольцо через регистры переноса. Это основано на отложенном распространении переноса для реализации конвейера переносов между быстрыми сумматорами.

«Тороидальный» конвейер сумматоров показан на рисунке ниже.

Реализация быстрых сумматоров на устройствах PLD в настоящее время ограничена 3 битами, поскольку для сумматора с i битами требуется $4+2^i$ элементов результата, а нынешние PLD поддерживают лишь 16. Получаемому в результате устройству нужно не более 5 тактов «бездействия» (idle) для передачи переносов через конвейер накопления.

[1] В оригинале ошибочно сказано
p : carry propagate, where pi = ai + bi
g : carry generate, where gi = ai*bi = (ai)(bi)
См. https://www.rfc-editor.org/errata/eid530. *Прим. перев.*

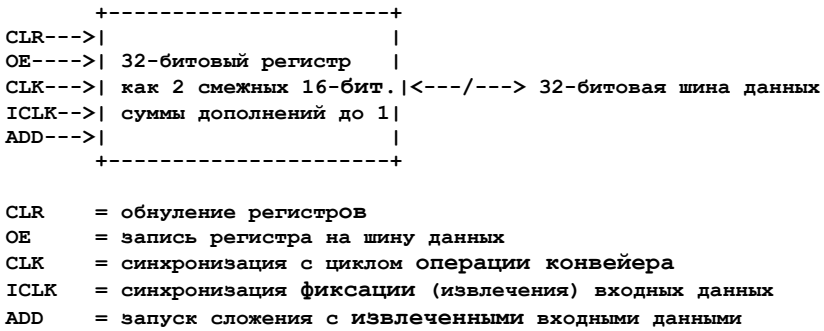


Интерфейсы

Описанное выше устройство было установлено в плату хост-интерфейса VL-Bus PC [8]. Оно имеет аппаратный и программный интерфейс, описанные ниже.

Аппаратный интерфейс

Аппаратная часть расчета Internet Checksum представляется как однопортовый 32-битовый регистр с сигналами синхронизации и управления [8], показанный ниже.



CLR обнуляет регистр контрольной суммы и вход фиксации (latch) данных. Здесь нет явных данных (load), это эквивалентно CLR с последующей записью значения в фиктивное место.

OE записывает содержимое регистра на шину данных (третье состояние).

CLK запускает работу конвейера. Если нет вновь извлеченных входных данных для сложения (через ICLK, ADD), прибавляется виртуальнуй «0» для опустошения конвейера.

ICLK («прозрачно») извлекает значение из шины данных для внутренней фиксации и суммирования в «аккумуляторе» по следующему сигналу ADD. Сигнал ADD вызывает накопление извлеченных входных данных (ICLK) в конвейере контрольной суммы. ADD и ICLK обычно связаны воедино. Одно 32-битовое значение данных может быть извлечено и аккумулировано в сумматоре конвейера за каждый такт продолжительностью 26 нсек в предположении стабильности данных в момент сигнала ADD/ICLK.

Внутренний 32-битовый регистр организован в форме двух 16-битовых сумм с дополнением до 1 для четного и нечетного 16-битовых слов потока данных. Для расчета Internet Checksum из этих значений эти половинки складываются с дополнением до 1 и результат инвертируется.

Программный интерфейс

Устройство используется как отображаемый на память регистр и доступ к нему для чтения выглядит как обычное считывание памяти.

Устройство управляется через внешний регистр, отображаемый на память. Биты этого регистра очищают (сбрасывают) устройство (по линии CLR), а также включают и отключают его (линия ADD). Линия CLR может дополнительно отображаться на запись в память, например для того, чтобы считывание по адресу было неразрушающим считыванием регистра контрольной суммы, а запись любого значения сбрасывала бы этот регистр. Управление включением и выключением должно сохраняться в отдельном регистре.

Устройство предназначено для работы в фоновом режиме при операциях переноса в памяти (DMA или программируемый ввод-вывод). После включения все операции переноса по этой шине складываются в регистре контрольной суммы. Контрольная сумма доступна в течение 5 тактов за последним разрешенным накоплением данных. Эта задержка часто скрывается механизмами доступа к памяти и арбитража шины. При необходимости для требуемой задержки могут выполняться команды «остановки» (stall).

В приведенном ниже примере предполагается, что устройство размещено по адресу CKSUMLOC. Считывание по этому адресу означает чтение регистра контрольной суммы, а запись очищает этот регистр. Регистр управления имеет адрес CTLLOC, а бит включения и выключения контрольной суммы - CKSUMBIT (1 включает, 0 выключает). Для подсчета контрольной суммы нужно очистить регистр (возможно с инициализацией контрольной суммы), выполнить серию передач и использовать результат, как показано ниже.

```
/******* Инициализация *******/
*(CTLLOC) &= ~((ctlsize)(CKSUMBIT));     /* выключение sum */
(word32)(*(CKSUMLOC)) = 0;               /* очистка регистра */
*(CTLLOC) |= CKSUMBIT;                   /* включение sum */
{ (необязательно) запись начального значения в фиктивное место }

/***** Перенос данных *****/
{ Выполнение одного или нескольких переносов (чтение и запись) DMA или PIO }

/***** Выборка результата *****/
*(CTLLOC) &= ~((ctlsize)(CKSUMBIT));     /* выключение sum */
sum = (word32)(*(CKSUMLOC));             /* чтение sum */
sum = (sum & 0xFFFF) + (sum >> 16);      /* заполнение половинок */
sum = (sum & 0xFFFF) + (sum >> 16);      /* сложение с переносом */
ipcksum = (halfword16)(~(sum & 0xFFFF)); /* обращение */
```

Заключение

В этом документе описан блок аппаратного расчета Internet Checksum, который можно реализовать на недорогих PLD, обеспечивающий производительность 1,26 Гбит/с. Блок был реализован на хост-интерфейсе PC-ATOMIC [8]. Предполагается, что это найдет широкое применение в сообществе Internet.

Вопросы безопасности

Вопросы безопасности не рассматриваются здесь, поскольку суммы Internet Checksum не служит мерой защиты.

Благодарности

Авторы благодарят членов подразделений ISI High-Performance Computing and Communications (особенно Mike Carlton) и Advanced Systems за помощь при создании устройства и подготовке этого документа.

Литература

[1] Braden, R., Borman, D., and Partridge, C., "Computing the Internet Checksum," Network Working Group RFC-1071, ISI, Cray Research, and BBN Labs, Sept. 1988.

[2] Mallory, T., and Kullberg, A., "Incremental Updating of the Internet Checksum," Network Working Group RFC-1141, BBN Comm., Jan. 1990.

[3] Plummer, W., "TCP Checksum Function Design," IEN-45, BBN, 1978, включенный как приложение в RFC-1071.

[4] Postel, Jon, "Internet Protocol," Network Working Group RFC- 791/STD-5, ISI, Sept. 1981.

[5] Postel, Jon, "User Datagram Protocol," Network Working Group RFC-768/STD-6, ISI, Aug. 1980.

[6] Postel, Jon, "Transmission Control Protocol," Network Working Group RFC-793/STD-7, ISI, Sept. 1981.

[7] Rijsinghani, A., "Computation of the Internet Checksum via Incremental Update," Network Working Group RFC-1624, Digital Equipment Corp., May 1994.

[8] Touch, J., "PC-ATOMIC", ISI Tech. Report. SR-95-407, June 1995.

Адреса авторов

Joe Touch

University of Southern California/Information Sciences Institute

4676 Admiralty Way

Marina del Rey, CA 90292-6695

USA

Phone: +1 310-822-1511 x151

Fax: +1 310-823-6714

URL: http://www.isi.edu/~touch

EMail: touch@isi.edu

Bruce Parham

University of Southern California/Information Sciences Institute

4676 Admiralty Way

Marina del Rey, CA 90292-6695

USA

Phone: +1 310-822-1511 x101

Fax: +1 310-823-6714

EMail: bparham@isi.edu

Приложение A. Исходный код PLD

Ниже приведен исходный код для AMD MACH-435 PLD, состоящего из 8 эквивалентных 22V10 блоков PLD, соединенных настраиваемой внутренней матрицей.

```
---- (Далее приведен код PLD) ----

TITLE    Ускоритель PC-ATOMIC IP Sum — 1-тактовая версия 2- и 3-бита, 26 нсек
PATTERN  ip_sum
REVISION 1.01
AUTHOR   J. Touch & B. Parham
COMPANY  USC/ISI
DATE     06/21/94

CHIP    ip_sum          MACH435

; Накопление за 1 такт (1 уровень логики).
;
; Выделены ресурсы для сокращения времени сборки.
;
; Используется 1 входной регистр dl для захвата значения шины данных
; по фронту. Сумма дополнение до 1 собирается в регистре q.
; Входные и выходные данные доступны через двухсторонние выводы dq.
;
; Используются 2 группы 6-битвых ресурсов переноса cy
;
; Используется 3-битовый полный сумматор с переносом (установка за 6 тактов)
; 16 битов сгруппированы как [000102 030405 0607 080910 111213 1415]
;                             [161718 192021 2223 242526 272829 3031]
;
; Блокировка выводов ускоряет установку и предназначена для форсированного
; включения 4-битовых компонентов в отдельные «сегменты» PLD.
; Это можно указать в форме
;       GROUP MACH_SEG_A        dq[6..0]
;       GROUP MACH_SEG_B        dq[14..8]
;       GROUP MACH_SEG_C        dq[22..16]
;       GROUP MACH_SEG_D        dq[30..24]

;
; Управляющие выводы
;
PIN     20      clk             ; Синхронизация сумматора
PIN     62      ip_add          ; Прибавление текущих данных к сумме
PIN     83      ip_sum_ena      ; Вывод текущей суммы
PIN     41      ip_clr          ; Очистка текущей суммы
PIN 23  ip_dclk                 ; Ввод захваченных данных (с привязкой
                                    ; или без привязки к синхронизации)

;
; dq — контакты шины данных
; dl — входной регистр
;
PIN     [9..3]          dq[6..0] IPAIR dl[6..0]         ; порт ввода-вывода
PIN     [18..12]        dq[14..8] IPAIR dl[14..8]       ; порт ввода-вывода
PIN     [30..24]        dq[22..16] IPAIR dl[22..16]     ; порт ввода-вывода
PIN     [39..33]        dq[30..24] IPAIR dl[30..24]     ; порт ввода-вывода
PIN     ?       dq[31,23,15,7] IPAIR dl[31,23,15,7]     ; порт ввода-вывода

;
; q — регистр частичной контрольной суммы
; dl — входной регистр
; dq — выводы шины данных
;
NODE    ?       q[31..0] OPAIR dq[31..0]        ; внутренние данные в регистре
NODE    ?       dl[31..0] REG                   ; входной регистр

;
; cy — биты регистра переноса
;
NODE    ?       cy[31,29,26,23,21,18,15,13,10,7,5,2] REG
                                                ; 1-битовый внутренний перенос

EQUATIONS

;
; .trst — управление с 3 состояниями (0 означает, что это всегда входы)
;
ip_add.trst             = 0
ip_clr.trst             = 0
ip_sum_ena.trst         = 0

;
; Выборка данных во внутренний регистр на каждом такте (только для корректного)
;
dl[31..0].clkf          = ip_dclk       ; Всегда выбирать данные
```

```
        ; не использовать setf, rstf или trst для dl
        ; нужно, чтобы dl отражал входные регистры, а не внутренние ячейки
        ; входными регистрами не обязательно должны быть setf, rstf или trst

;
; Управление регистром контрольной суммы
;
dq[31..0].clkf          = clk                   ; clk синхронизирует действия
dq[31..0].setf          = gnd                   ; регистры не предустанавливаются
dq[31..0].rstf          = ip_clr                ; очистка при сбросе
dq[31..0].trst          = ip_sum_ena            ; выходная сумма включена - чтение

;
; Управление регистром переноса
;
cy[31,29,26,23,21,18,15,13,10,7,5,2].clkf       = clk
cy[31,29,26,23,21,18,15,13,10,7,5,2].setf       = gnd    ; нет предустановки
cy[31,29,26,23,21,18,15,13,10,7,5,2].rstf       = ip_clr ; очистка при сбросе

;
; Выборка входных данных
; Просто берутся все вывод при высоком уровне ip_add
; Данные попадают во входной регистр
;
dl[31..0]               := dq[31..0]

;
; Комбинаторный сумматор
;
; Создается как последовательность 2- и 3-битовых (с переносом вперед) полных
; сумматоров передающих перенос в регистр переноса конвейера
;
; sum[n] — сумма битов
; cy[m] — биты переноса
; ":+:" - XOR

;
; SUM[0] = (A0 :+: B0 :+: CARRY_IN)
;
; CY[0] = ((A0 * B0) + ((A0 :+: B0) * CARRY_IN))
;
; На деле второе можно выполнить в форме
;
; CY[0] = ((A0 * B0) + ((A0 + B0) * CARRY_IN))
;
; поскольку XOR не будет сводиться на нет случаем AND по причине
; что результатом всегда будет 1 из-за первого члена и в любом случае
; это снижает требуемое число XOR, которое ограничено в PLD
;
; Сумма младшего слова
;

;
; первые 5 битов [0..4] младшего слова
;
q[0]    := (q[0] :+: (ip_add * dl[0]) :+: cy[15])

q[1]    := (q[1] :+: (ip_add * dl[1]) :+:
        ((ip_add *
                (q[0] * dl[0] +
                dl[0] * cy[15])) +
        (q[0] * cy[15])))

q[2]    := (q[2] :+: (ip_add * dl[2]) :+:
        ((ip_add *
                (q[1] * dl[1] +
                 q[1] *  q[0] *  dl[0] +
                dl[1] *  q[0] *  dl[0] +
                 q[1] * dl[0] * cy[15] +
                dl[1] * dl[0] * cy[15] +
                dl[1] *  q[0] * cy[15])) +
        (q[1] * q[0] * cy[15])))

cy[2]   := ((ip_add *
                (q[2] * dl[2] +
                 q[2] *  q[1] * dl[1] +
                dl[2] *  q[1] * dl[1] +
                 q[2] *  q[1] *  q[0] *  dl[0] +
                 q[2] * dl[1] *  q[0] *  dl[0] +
                dl[2] *  q[1] *  q[0] *  dl[0] +
                dl[2] * dl[1] *  q[0] *  dl[0] +
                 q[2] *  q[1] * dl[0] * cy[15] +
                 q[2] * dl[1] *  q[0] * cy[15] +
                 q[2] * dl[1] * dl[0] * cy[15] +
                dl[2] *  q[1] *  q[0] * cy[15] +
```

```
                dl[2] *  q[1] * dl[0] * cy[15] +
                dl[2] * dl[1] *  q[0] * cy[15] +
                dl[2] * dl[1] * dl[0] * cy[15])) +
        (q[2] * q[1] * q[0] * cy[15]))

q[3]    := (q[3] :+: (ip_add * dl[3]) :+: cy[2])

q[4]    := (q[4] :+: (ip_add * dl[4]) :+:
        ((ip_add *
                (q[3] * dl[3] +
                dl[3] * cy[2])) +
        (q[3] * cy[2])))

;
; следующие 3 бита [5..7] младшего слова
;
q[5]    := (q[5] :+: (ip_add * dl[5]) :+:
        ((ip_add *
                (q[4] * dl[4] +
                 q[4] *  q[3] * dl[3] +
                dl[4] *  q[3] * dl[3] +
                 q[4] * dl[3] * cy[2] +
                dl[4] * dl[3] * cy[2] +
                dl[4] *  q[3] * cy[2])) +
        (q[4] * q[3] * cy[2])))

cy[5]   := ((ip_add * (
                 q[5] * dl[5] +
                 q[5] *  q[4] * dl[4] +
                dl[5] *  q[4] * dl[4] +
                 q[5] *  q[4] *  q[3] * dl[3] +
                 q[5] * dl[4] *  q[3] * dl[3] +
                dl[5] *  q[4] *  q[3] * dl[3] +
                dl[5] * dl[4] *  q[3] * dl[3] +
                 q[5] *  q[4] * dl[3] * cy[2] +
                 q[5] * dl[4] *  q[3] * cy[2] +
                 q[5] * dl[4] * dl[3] * cy[2] +
                dl[5] *  q[4] *  q[3] * cy[2] +
                dl[5] *  q[4] * dl[3] * cy[2] +
                dl[5] * dl[4] *  q[3] * cy[2] +
                dl[5] * dl[4] * dl[3] * cy[2])) +
        (q[5] * q[4] * q[3] * cy[2]))

q[6]    := (q[6] :+: (ip_add * dl[6]) :+: cy[5])

q[7]    := (q[7] :+: (ip_add * dl[7]) :+:
        ((ip_add *
                (q[6] * dl[6] +
                dl[6] * cy[5])) +
        (q[6] * cy[5])))

cy[7]   := ((ip_add *
                (q[7] * dl[7] +
                 q[7] *  q[6] * dl[6] +
                dl[7] *  q[6] * dl[6] +
                 q[7] * dl[6] * cy[5] +
                dl[7] * dl[6] * cy[5] +
                dl[7] *  q[6] * cy[5])) +
        (q[7] * q[6] * cy[5]))

;
; следующие 5 битов [8..12] младшего слова
;
q[8]    := (q[8] :+: (ip_add * dl[8]) :+: cy[7])

q[9]    := (q[9] :+: (ip_add * dl[9]) :+:
        ((ip_add *
                (q[8] * dl[8] +
                dl[8] * cy[7])) +
        (q[8] * cy[7])))

q[10]   := (q[10] :+: (ip_add * dl[10]) :+:
        ((ip_add *
                (q[9] * dl[9] +
                 q[9] *  q[8] * dl[8] +
                dl[9] *  q[8] * dl[8] +
                 q[9] * dl[8] * cy[7] +
                dl[9] * dl[8] * cy[7] +
                dl[9] *  q[8] * cy[7])) +
        (q[9] * q[8] * cy[7])))

cy[10]  := ((ip_add *
                (q[10] * dl[10] +
                 q[10] *  q[9] * dl[9] +
                dl[10] *  q[9] * dl[9] +
                 q[10] *  q[9] *  q[8] * dl[8] +
```

```
                 q[10] * dl[9] *  q[8] * dl[8] +
                dl[10] *  q[9] *  q[8] * dl[8] +
                dl[10] * dl[9] *  q[8] * dl[8] +
                 q[10] *  q[9] * dl[8] * cy[7] +
                 q[10] * dl[9] *  q[8] * cy[7] +
                 q[10] * dl[9] * dl[8] * cy[7] +
                dl[10] *  q[9] *  q[8] * cy[7] +
                dl[10] *  q[9] * dl[8] * cy[7] +
                dl[10] * dl[9] *  q[8] * cy[7] +
                dl[10] * dl[9] * dl[8] * cy[7])) +
        (q[10] * q[9] * q[8] * cy[7]))

q[11]   := (q[11] :+: (ip_add * dl[11]) :+: cy[10])

q[12]   := (q[12] :+: (ip_add * dl[12]) :+:
        ((ip_add *
                (q[11] * dl[11] +
                dl[11] * cy[10])) +
        (q[11] * cy[10])))

;
; финальные 3 бита [13..15] младшего слова
;
q[13]   := (q[13] :+: (ip_add * dl[13]) :+:
        ((ip_add *
                (q[12] * dl[12] +
                 q[12] *  q[11] * dl[11] +
                dl[12] *  q[11] * dl[11] +
                 q[12] * dl[11] * cy[10] +
                dl[12] * dl[11] * cy[10] +
                dl[12] *  q[11] * cy[10])) +
        (q[12] * q[11] * cy[10])))

cy[13]  := ((ip_add * (
                 q[13] * dl[13] +
                 q[13] *  q[12] * dl[12] +
                dl[13] *  q[12] * dl[12] +
                 q[13] *  q[12] *  q[11] * dl[11] +
                 q[13] * dl[12] *  q[11] * dl[11] +
                dl[13] *  q[12] *  q[11] * dl[11] +
                dl[13] * dl[12] *  q[11] * dl[11] +
                 q[13] *  q[12] * dl[11] * cy[10] +
                 q[13] * dl[12] *  q[11] * cy[10] +
                 q[13] * dl[12] * dl[11] * cy[10] +
                dl[13] *  q[12] *  q[11] * cy[10] +
                dl[13] *  q[12] * dl[11] * cy[10] +
                dl[13] * dl[12] *  q[11] * cy[10] +
                dl[13] * dl[12] * dl[11] * cy[10])) +
        (q[13] * q[12] *  q[11] * cy[10]))

q[14]   := (q[14] :+: (ip_add * dl[14]) :+: cy[13])

q[15]   := (q[15] :+: (ip_add * dl[15]) :+:
        ((ip_add *
                (q[14] * dl[14] +
                dl[14] * cy[13])) +
        (q[14] * cy[13])))

cy[15]  := ((ip_add *
                (q[15] * dl[15] +
                 q[15] *  q[14] * dl[14] +
                dl[15] *  q[14] * dl[14] +
                 q[15] * dl[14] * cy[13] +
                dl[15] * dl[14] * cy[13] +
                dl[15] *  q[14] * cy[13])) +
        (q[15] * q[14] * cy[13]))

; Сумма старшего слова
;

;
; первые 5 битов [16..20] старшего слова
;
q[16]   := (q[16] :+: (ip_add * dl[16]) :+: cy[31])

q[17]   := (q[17] :+: (ip_add * dl[17]) :+:
        ((ip_add *
                (q[16] * dl[16] +
                dl[16] * cy[31])) +
        (q[16] * cy[31])))

q[18]   := (q[18] :+: (ip_add * dl[18]) :+:
        ((ip_add *
                (q[17] * dl[17] +
                 q[17] *  q[16] * dl[16] +
```

```
                dl[17] *  q[16] * dl[16] +
                 q[17] * dl[16] * cy[31] +
                dl[17] * dl[16] * cy[31] +
                dl[17] *  q[16] * cy[31])) +
        (q[17] * q[16] * cy[31])))

cy[18]  := ((ip_add *
                (q[18] * dl[18] +
                 q[18] *  q[17] * dl[17] +
                dl[18] *  q[17] * dl[17] +
                 q[18] *  q[17] *  q[16] * dl[16] +
                 q[18] * dl[17] *  q[16] * dl[16] +
                dl[18] *  q[17] *  q[16] * dl[16] +
                dl[18] * dl[17] *  q[16] * dl[16] +
                 q[18] *  q[17] * dl[16] * cy[31] +
                 q[18] * dl[17] *  q[16] * cy[31] +
                 q[18] * dl[17] * dl[16] * cy[31] +
                dl[18] *  q[17] *  q[16] * cy[31] +
                dl[18] *  q[17] * dl[16] * cy[31] +
                dl[18] * dl[17] *  q[16] * cy[31] +
                dl[18] * dl[17] * dl[16] * cy[31])) +
        (q[18] * q[17] * q[16] * cy[31]))

q[19]   := (q[19] :+: (ip_add * dl[19]) :+: cy[18])

q[20]   := (q[20] :+: (ip_add * dl[20]) :+:
        ((ip_add *
                (q[19] * dl[19] +
                dl[19] * cy[18])) +
        (q[19] * cy[18])))

;
; следующие 3 бита [21..23] старшего слова
;
q[21]   := (q[21] :+: (ip_add * dl[21]) :+:
        ((ip_add *
                (q[20] * dl[20] +
                 q[20] *  q[19] * dl[19] +
                dl[20] *  q[19] * dl[19] +
                 q[20] * dl[19] * cy[18] +
                dl[20] * dl[19] * cy[18] +
                dl[20] *  q[19] * cy[18])) +
        (q[20] * q[19] * cy[18])))

cy[21]  := ((ip_add * (
                 q[21] * dl[21] +
                 q[21] *  q[20] * dl[20] +
                dl[21] *  q[20] * dl[20] +
                 q[21] *  q[20] *  q[19] * dl[19] +
                 q[21] * dl[20] *  q[19] * dl[19] +
                dl[21] *  q[20] *  q[19] * dl[19] +
                dl[21] * dl[20] *  q[19] * dl[19] +
                 q[21] *  q[20] * dl[19] * cy[18] +
                 q[21] * dl[20] *  q[19] * cy[18] +
                 q[21] * dl[20] * dl[19] * cy[18] +
                dl[21] *  q[20] *  q[19] * cy[18] +
                dl[21] *  q[20] * dl[19] * cy[18] +
                dl[21] * dl[20] *  q[19] * cy[18] +
                dl[21] * dl[20] * dl[19] * cy[18])) +
        (q[21] * q[20] * q[19] * cy[18]))

q[22]   := (q[22] :+: (ip_add * dl[22]) :+: cy[21])

q[23]   := (q[23] :+: (ip_add * dl[23]) :+:
        ((ip_add *
                (q[22] * dl[22] +
                dl[22] * cy[21])) +
        (q[22] * cy[21])))

cy[23]  := ((ip_add *
                (q[23] * dl[23] +
                 q[23] *  q[22] * dl[22] +
                dl[23] *  q[22] * dl[22] +
                 q[23] * dl[22] * cy[21] +
                dl[23] * dl[22] * cy[21] +
                dl[23] *  q[22] * cy[21])) +
        (q[23] * q[22] * cy[21]))

;
; следующие 5 битов [24..28] старшего слова
;
q[24]   := (q[24] :+: (ip_add * dl[24]) :+: cy[23])

q[25]   := (q[25] :+: (ip_add * dl[25]) :+:
        ((ip_add *
```

```
                (q[24] * dl[24] +
                dl[24] * cy[23])) +
        (q[24] * cy[23])))

q[26]   := (q[26] :+: (ip_add * dl[26]) :+:
        ((ip_add *
                (q[25] * dl[25] +
                 q[25] *  q[24] * dl[24] +
                dl[25] *  q[24] * dl[24] +
                 q[25] * dl[24] * cy[23] +
                dl[25] * dl[24] * cy[23] +
                dl[25] *  q[24] * cy[23])) +
        (q[25] * q[24] * cy[23])))

cy[26]  := ((ip_add *
                (q[26] * dl[26] +
                 q[26] *  q[25] * dl[25] +
                dl[26] *  q[25] * dl[25] +
                 q[26] *  q[25] *  q[24] * dl[24] +
                 q[26] * dl[25] *  q[24] * dl[24] +
                dl[26] *  q[25] *  q[24] * dl[24] +
                dl[26] * dl[25] *  q[24] * dl[24] +
                 q[26] *  q[25] * dl[24] * cy[23] +
                 q[26] * dl[25] *  q[24] * cy[23] +
                 q[26] * dl[25] * dl[24] * cy[23] +
                dl[26] *  q[25] *  q[24] * cy[23] +
                dl[26] *  q[25] * dl[24] * cy[23] +
                dl[26] * dl[25] *  q[24] * cy[23] +
                dl[26] * dl[25] * dl[24] * cy[23])) +
        (q[26] * q[25] * q[24] * cy[23]))

q[27]   := (q[27] :+: (ip_add * dl[27]) :+: cy[26])

q[28]   := (q[28] :+: (ip_add * dl[28]) :+:
        ((ip_add *
                (q[27] * dl[27] +
                dl[27] * cy[26])) +
        (q[27] * cy[26])))

;
; финальные 3 бита [29..31] старшего слова
;
q[29]   := (q[29] :+: (ip_add * dl[29]) :+:
        ((ip_add *
                (q[28] * dl[28] +
                 q[28] *  q[27] * dl[27] +
                dl[28] *  q[27] * dl[27] +
                 q[28] * dl[27] * cy[26] +
                dl[28] * dl[27] * cy[26] +
                dl[28] *  q[27] * cy[26])) +
        (q[28] * q[27] * cy[26])))

cy[29]  := ((ip_add * (
                 q[29] * dl[29] +
                 q[29] *  q[28] * dl[28] +
                dl[29] *  q[28] * dl[28] +
                 q[29] *  q[28] *  q[27] * dl[27] +
                 q[29] * dl[28] *  q[27] * dl[27] +
                dl[29] *  q[28] *  q[27] * dl[27] +
                dl[29] * dl[28] *  q[27] * dl[27] +
                 q[29] *  q[28] * dl[27] * cy[26] +
                 q[29] * dl[28] *  q[27] * cy[26] +
                 q[29] * dl[28] * dl[27] * cy[26] +
                dl[29] *  q[28] *  q[27] * cy[26] +
                dl[29] *  q[28] * dl[27] * cy[26] +
                dl[29] * dl[28] *  q[27] * cy[26] +
                dl[29] * dl[28] * dl[27] * cy[26])) +
        (q[29] * q[28] * q[27] * cy[26]))

q[30]   := (q[30] :+: (ip_add * dl[30]) :+: cy[29])

q[31]   := (q[31] :+: (ip_add * dl[31]) :+:
        ((ip_add *
                (q[30] * dl[30] +
                dl[30] * cy[29])) +
        (q[30] * cy[29])))

cy[31]  := ((ip_add *
                (q[31] * dl[31] +
                 q[31] *  q[30] * dl[30] +
                dl[31] *  q[30] * dl[30] +
                 q[31] * dl[30] * cy[29] +
                dl[31] * dl[30] * cy[29] +
                dl[31] *  q[30] * cy[29])) +
        (q[31] * q[30] * cy[29]))
```

```
;
; Выходные биты на выходных контактах (ожидание включения)
;
dq[0]   := {q[0]}
dq[1]   := {q[1]}
dq[2]   := {q[2]}
dq[3]   := {q[3]}
dq[4]   := {q[4]}
dq[5]   := {q[5]}
dq[6]   := {q[6]}
dq[7]   := {q[7]}
dq[8]   := {q[8]}
dq[9]   := {q[9]}
dq[10]  := {q[10]}
dq[11]  := {q[11]}
dq[12]  := {q[12]}
dq[13]  := {q[13]}
dq[14]  := {q[14]}
dq[15]  := {q[15]}

dq[16]  := {q[16]}
dq[17]  := {q[17]}
dq[18]  := {q[18]}
dq[19]  := {q[19]}
dq[20]  := {q[20]}
dq[21]  := {q[21]}
dq[22]  := {q[22]}
dq[23]  := {q[23]}
dq[24]  := {q[24]}
dq[25]  := {q[25]}
dq[26]  := {q[26]}
dq[27]  := {q[27]}
dq[28]  := {q[28]}
dq[29]  := {q[29]}
dq[30]  := {q[30]}
dq[31]  := {q[31]}

;
; Завершение.
;
```

Перевод на русский язык
Николай Малых

nmalykh@protokols.ru