RIPEMD-160 - Verification of a SPARK/ADA Implementation

Fabian Immler

May 28, 2015

Abstract
This work presents a verification of an implementation in SPARK/ADA [1] of the cryptographic hash-function RIPEMD-160. A functional specification of RIPEMD-160 [2] is given in Isabelle/HOL [3]. Proofs for the verification conditions generated by the static-analysis toolset of SPARK certify the functional correctness of the implementation. The verification conditions are translated to Isabelle/HOL with a modified version of Victor-0.8.0 [4].

Contents
1 Introduction 2
2 Specification of RIPEMD-160 3
3 Global Specifications 6
   3.1 Specification of Bit-Operations . . . . . . . . . . . . . . . . . . . . . . 6
   3.2 Conversions for proof functions . . . . . . . . . . . . . . . . . . . . . 7
4 Verification of $f$ 7
5 Verification of $k_l$ 9
6 Verification of $k_r$ 10
7 Arrays in SPARK vs Lists in Isabelle 12
   7.1 Functions vs Lists . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
   7.2 Maximum Element of Lists . . . . . . . . . . . . . . . . . . . . . . . 13
8 Verification of $r_l$ 13
9 Verification of $r_r$ 15
10 Verification of $s_l$ 17
11 Verification of $s_r$

12 Verification of $round$

13 Verification of $hash$

1 Introduction

The directory ada contains the sourcecode which has been verified against its specification in Isabelle/HOL (close to its pseudocode definition from [2]) in the following. The SPARK-code contains annotations with so called proof functions. The following proof functions (declared in ada/rmd.ads) are specified in Isabelle/HOL:

- bit_and
- bit_or
- bit_xor
- wordops_rotate_left
- f
- $k_l$
- $k_r$
- $r_l$
- $r_r$
- $s_l$
- $s_r$
- steps
- $round$
- rounds
- $rmd_hash$

From the annotations in the SPARK-code, verification conditions were generated using SPARK-GPL-2010 (http://libre.adacore.com/libre/download/):

```
$spark -v cg -rules=lazy ada/shadow/interfaces.ads ada/wordops.ads ada/rmd.ads ada/rmd.adb
```

A slightly modified Version of VICTOR [4] translated these verification conditions to Isabelle (the results can be found in the theories ending with
Obligation and Declaration\textsuperscript{1}. Definitions for the roof-functions are given in the theories with the suffix Specification and the proofs are given in the theories ending in User.

2 Specification of RIPEMD-160

theory RMD
imports \\
begin

typedef word32 = 32 word
typedef byte = 8 word
typedef perm = nat => nat
typedef chain = word32 * word32 * word32 * word32 * word32
typedef block = nat => word32
typedef message = nat => block

definition f::[nat, word32, word32, word32] => word32
where
  f j x y z =
    (if ( 0 <= j & j <= 15) then x XOR y XOR z
     else if (16 <= j & j <= 31) then (x AND y) OR (NOT x AND z)
     else if (32 <= j & j <= 47) then (x OR NOT y) XOR z
     else if (48 <= j & j <= 63) then (x AND z) OR (y AND NOT z)
     else if (64 <= j & j <= 79) then x XOR (y OR NOT z)
     else 0)
definition K::nat => word32
where
  K j =
    (if ( 0 <= j & j <= 15) then 0x00000000
     else if (16 <= j & j <= 31) then 0x5A827999
     else if (32 <= j & j <= 47) then 0x6ED9EBA1
     else if (48 <= j & j <= 63) then 0x8F1BBCDC
     else if (64 <= j & j <= 79) then 0xA953FD4E
     else 0)
definition K'::nat => word32
where
  K' j =
    (if ( 0 <= j & j <= 15) then 0x50A28BE6
     else if (16 <= j & j <= 31) then 0x5A827999
     else if (32 <= j & j <= 47) then 0x6ED9EBA1
     else if (48 <= j & j <= 63) then 0x8F1BBCDC
     else if (64 <= j & j <= 79) then 0xA953FD4E
     else 0)

\textsuperscript{1}There are some slight superficial differences between the original translated files and the ones included here, in order to conform to current Isabelle practice
else 0)

definition r-list :: nat list
  where r-list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 7, 4, 13, 1, 10, 6, 15, 3, 12, 0, 9, 5, 2, 14, 11, 8, 3, 10, 14, 4, 9, 15, 8, 1, 2, 7, 0, 6, 13, 11, 5, 12, 1, 9, 11, 10, 0, 8, 12, 4, 13, 3, 7, 15, 14, 5, 6, 2, 4, 0, 5, 9, 7, 12, 2, 10, 14, 1, 3, 8, 11, 6, 15, 13]

definition r'-list :: nat list
  where r'-list = [5, 14, 7, 0, 9, 2, 11, 4, 13, 6, 15, 8, 1, 10, 3, 12, 6, 11, 3, 7, 0, 13, 5, 10, 14, 15, 8, 12, 4, 9, 1, 2, 15, 5, 1, 3, 7, 14, 6, 9, 11, 8, 12, 2, 10, 0, 4, 13, 8, 6, 4, 1, 3, 11, 15, 0, 5, 12, 2, 13, 9, 7, 10, 14, 12, 15, 10, 4, 1, 5, 8, 7, 6, 2, 13, 14, 0, 3, 9, 11]

definition r :: perm
  where r j = r-list ! j

definition r' :: perm
  where r' j = r'-list ! j

definition s-list :: nat list
  where s-list = [11, 14, 15, 12, 5, 8, 7, 9, 11, 13, 14, 15, 6, 7, 9, 8, 7, 6, 8, 13, 11, 9, 7, 15, 7, 12, 15, 9, 11, 7, 13, 12, 11, 13, 6, 7, 14, 9, 13, 15, 14, 8, 13, 6, 5, 12, 7, 5, 11, 12, 14, 15, 14, 15, 9, 8, 9, 14, 5, 6, 8, 6, 5, 12, 9, 15, 5, 11, 6, 8, 13, 12, 5, 12, 13, 14, 11, 8, 5, 6]

definition s'-list :: nat list
  where s'-list = [8, 9, 9, 11, 13, 15, 15, 5, 7, 7, 8, 11, 14, 14, 12, 6, 9, 13, 15, 7, 12, 8, 9, 11, 7, 7, 12, 7, 6, 15, 13, 11, 9, 7, 15, 11, 8, 6, 6, 14, 12, 13, 5, 14, 13, 13, 7, 5, 15, 5, 8, 11, 14, 14, 6, 14, 6, 9, 12, 9, 12, 5, 15, 8, 8, 5, 12, 9, 12, 5, 14, 6, 8, 13, 6, 5, 15, 13, 11, 11]

definition s :: perm
  where s j = s-list ! j

definition s' :: perm
  where s' j = s'-list ! j
definition h0-0::word32 where h0-0 = 0x67452301
definition h1-0::word32 where h1-0 = 0xEFCDAB89
definition h2-0::word32 where h2-0 = 0x98BADCFE
definition h3-0::word32 where h3-0 = 0x10325476
definition h4-0::word32 where h4-0 = 0xC3D2E1F0
definition h-0 :: chain where
h-0 = (h0-0, h1-0, h2-0, h3-0, h4-0)
definition step-l ::
[ block, chain, nat ] => chain
where
step-l X c j =
(let (A, B, C, D, E) = c in
((* A *) E),
(* B *) word-rotl (s j) (A + f j B C D + X (r j) + K j) + E,
(* C *) B,
(* D *) word-rotl 10 C,
(* E *) D))
definition step-r ::
[ block, chain, nat ] => chain
where
step-r X c′ j =
(let (A′, B′, C′, D′, E′) = c′ in
((* A′ *) E′),
(* B′ *) word-rotl (s′ j) (A′ + f (79 - j) B′ C′ D′ + X (r′ j) + K′ j) + E′,
(* C′ *) B′,
(* D′ *) word-rotl 10 C′,
(* E′ *) D′))
definition step-both ::
[ block, chain * chain, nat ] => chain * chain
where
step-both X cc j = (case cc of (c, c′) =>
(step-l X c j, step-r X c′ j))
definition steps::[ block, chain * chain, nat] => chain * chain
where steps X cc i = foldl (step-both X) cc [0..<i]
definition round::[ block, chain ] => chain
where round X h =
(let (h0, h1, h2, h3, h4) = h in
let ((A, B, C, D, E), (A', B', C', D', E')) = steps X (h, h) 80 in
    ((\* h0 \*) h1 + C + D',
    \* h1 \* h2 + D + E',
    \* h2 \* h3 + E + A',
    \* h3 \* h4 + A + B',
    \* h4 \* h0 + B + C'))

definition rmd-body::[ message, chain, nat ] => chain
where
    rmd-body X h i = round (X i) h

definition rounds::message => chain => nat => chain
where
    rounds X h i = foldl (rmd-body X) h-0 [0..<i]

definition rmd :: message => nat => chain
where
    rmd X len = rounds X h-0 len

end

3 Global Specifications

theory Global-Specification
imports RMD
begin

SPARK has only one integer-type, therefore type-conversions are needed in order to specify the proof-functions in Isabelle.

3.1 Specification of Bit-Operations

The proof-functions for SPARK's bit-opertations are specified with HOL-Word

abbreviation bit--and' :: int => int => int where
    bit--and' m n == uint ((word-of-int m::word32) AND word-of-int n)

abbreviation bit--or' :: int => int => int where
    bit--or' m n == uint ((word-of-int m::word32) OR word-of-int n)

abbreviation bit--xor' :: int => int => int where
    bit--xor' m n == uint ((word-of-int m::word32) XOR word-of-int n)

abbreviation rotate-left' :: int => int => int where
    rotate-left' i w == uint (word-rotl (nat i) (word-of-int w::word32))

    This is how SPARK treats the bitwise not
lemma bit-not-spark-def[simp]:
\[(\text{word-of-int } (4294967295 - x)::\text{word32}) = \text{NOT (word-of-int } x)\]

\langle proof \rangle

3.2 Conversions for proof functions

Here, the proof-functions declared in the SPARK-Annotations are mapped to the corresponding parts of the Isabelle-Specification.

abbreviation k-l' :: int => int where
  k-l' j == uint (K (nat j))
abbreviation k-r' :: int => int where
  k-r' j == uint (K' (nat j))
abbreviation r-l' :: int => int where
  r-l' j == int (r (nat j))
abbreviation r-r' :: int => int where
  r-r' j == int (r' (nat j))
abbreviation s-l' :: int => int where
  s-l' j == int (s (nat j))
abbreviation s-r' :: int => int where
  s-r' j == int (s' (nat j))
abbreviation f' :: int => int => int => int => int where
  f' j x y z ==
      uint (f (nat j) (\text{word-of-int } x::\text{word32}) (\text{word-of-int } y) (\text{word-of-int } z))

end

4 Verification of $f$

theory F-Spark-Specification
imports F-Spark-Declaration Global-Specification
begin

abbreviation bit--and' :: [int, int] => int where
  bit--and' == Global-Specification.bit--and'
abbreviation bit--or' :: [int, int] => int where
  bit--or' == Global-Specification.bit--or'
abbreviation bit--xor' :: [int, int] => int where
  bit--xor' == Global-Specification.bit--xor'
abbreviation f' :: int => int => int => int => int where
  f' == Global-Specification.f'

end
theory F-Spark-User
imports F-Spark-Specification F-Spark-Declaration
begin

lemma goal2':
  shows $0 < \text{bit--or'}(\text{bit--and'} x' y'') (\text{bit--and'} (4294967295 - x') z'')$
  ⟨proof⟩

lemma goal2'2:
  shows $\text{bit--or'}(\text{bit--and'} x' y'') (\text{bit--and'} (4294967295 - x') z'') \leq 4294967295$
  ⟨proof⟩

lemma goal3'1:
  shows $0 < \text{bit--xor'} (\text{bit--or'} x' (4294967295 - y'')) z''$
  ⟨proof⟩

lemma goal3'2:
  shows $\text{bit--xor'} (\text{bit--or'} x' (4294967295 - y'')) z'' \leq 4294967295$
  ⟨proof⟩

lemma goal4'1:
  shows $0 < \text{bit--or'} (\text{bit--and'} x' z'') (\text{bit--and'} y' (4294967295 - z''))$
  ⟨proof⟩

lemma goal4'2:
  shows $\text{bit--or'} (\text{bit--and'} x' z'') (\text{bit--and'} y' (4294967295 - z'')) \leq 4294967295$
  ⟨proof⟩

lemma goal5'1:
  shows $0 < \text{bit--xor'} x'' (\text{bit--or'} y'' (4294967295 - z''))$
  ⟨proof⟩

lemma goal5'2:
  shows $\text{bit--xor'} x'' (\text{bit--or'} y'' (4294967295 - z'')) \leq 4294967295$
  ⟨proof⟩

lemma goal6'1:
  assumes $H8: j'' \leq (15 :: \text{int})$
  shows $\text{bit--xor'} x'' (\text{bit--xor'} y'' z'') = f' j'' x'' y'' z''$
  ⟨proof⟩

lemma goal7'1:
  assumes $H7: (16 :: \text{int}) \leq j''$
  assumes $H8: j'' \leq (31 :: \text{int})$
  shows $\text{bit--or'} (\text{bit--and'} x'' y'') (\text{bit--and'} (4294967295 - x'') z'') = f' j'' x'' y'' z''$
  ⟨proof⟩

8
lemma goal8':1:
  assumes H7: 32 <= j''
  assumes H8: j'' <= 47
  shows bit--xor' (bit--or' x'' (4294967295 - y'')) z'' = f' j'' x'' y'' z''
⟨proof⟩

lemma goal9':1:
  assumes H7: 48 <= j''
  assumes H8: j'' <= 63
  shows bit--or' (bit--and' x'' z'') (bit--and' y'' (4294967295 - z'')) = f' j'' x'' y'' z''
⟨proof⟩

lemma goal10':1:
  assumes H2: j'' <= 79
  assumes H12: 63 < j''
  shows bit--xor' x'' (bit--or' y'' (4294967295 - z'')) = f' j'' x'' y'' z''
⟨proof⟩

lemmas userlemmas =
  goal2':1
  goal2':2
  goal3':1
  goal3':2
  goal4':1
  goal4':2
  goal5':1
  goal5':2
  goal6':1
  goal7':1
  goal8':1
  goal9':1
  goal10':1

end

5 Verification of k_l

theory K-L-Spark-Specification
imports K-L-Spark-Declaration Global-Specification

begin

abbreviation k-l' :: int => int where
  k-l' == Global-Specification.k-l'

end

theory K-L-Spark-User
imports K-L-Spark-Specification K-L-Spark-Declaration
lemma goal6’1:
  fixes j::int
  assumes H1: 0 <= j
  assumes H2: j <= 15
  shows 0 = k-l’ j
⟨proof⟩

lemma goal7’1:
  fixes j::int
  assumes H1: 16 <= j
  assumes H2: j <= 31
  shows 1518500249 = k-l’ j
⟨proof⟩

lemma goal8’1:
  assumes H1: (32 :: int) <= j''
  assumes H2: j'' <= (47 :: int)
  shows (1859775933 :: int) = k-l' j''
⟨proof⟩

lemma goal9’1:
  assumes H1: (48 :: int) <= j''
  assumes H2: j'' <= (63 :: int)
  shows (2400959708 :: int) = k-l' j'' (is ?C1)
⟨proof⟩

lemma goal10’1:
  assumes H2: j'' <= (79 :: int)
  assumes H6: (63 :: int) < j''
  shows (2840853838 :: int) = k-l' j'' (is ?C1)
⟨proof⟩

lemmas userlemmas =
  goal6’1
  goal7’1
  goal8’1
  goal9’1
  goal10’1
end

6 Verification of $k_r$

theory K-R-Spark-Specification
imports K-R-Spark-Declaration Global-Specification
begin

abbreviation k-r' :: int => int where
  k-r' == Global-Specification.k-r'

end
theory K-R-Spark-User
imports K-R-Spark-Specification K-R-Spark-Declaration

begin

lemma goal6'1: assumes H1: (0 :: int) <= j''
  assumes H2: j'' <= (15 :: int)
  shows (1352829926 :: int) = k-r' j'' (is ?C1)
⟨proof⟩

lemma goal7'1: assumes H1: (16 :: int) <= j''
  assumes H2: j'' <= (31 :: int)
  shows (1548603684 :: int) = k-r' j'' (is ?C1)
⟨proof⟩

lemma goal8'1: assumes H1: (32 :: int) <= j''
  assumes H2: j'' <= (47 :: int)
  shows (1836072691 :: int) = k-r' j'' (is ?C1)
⟨proof⟩

lemma goal9'1: assumes H1: (48 :: int) <= j''
  assumes H2: j'' <= (63 :: int)
  shows (2053994217 :: int) = k-r' j'' (is ?C1)
⟨proof⟩

lemma goal10'1: assumes H2: j'' <= (79 :: int)
  assumes H6: (63 :: int) < j''
  shows (0 :: int) = k-r' j'' (is ?C1)
⟨proof⟩

lemmas userlemmas =
goal6'1

end
Arrays in SPARK vs Lists in Isabelle

theory Global-User
imports Main
begin

7.1 Functions vs Lists

Arrays defined in SPARK are represented as functions in Isabelle. In the specification, it is more convenient to use lists. Therefore it is a common task to prove equivalences like $\forall i \leq \text{length } l. \ l!i = f\ i$, where $l$ is the list specified in Isabelle and $f$ the function corresponding to the array defined in SPARK.

Constructing a function from a list makes things easier for the simplifier, otherwise the definition of the list would need to be unfolded ($\text{length } l$) times what yields to efficiency-problems.

primrec list-to-fun where
  list-to-fun [] :: (f::int => int) = f
  | list-to-fun (a # xs) i f = (list-to-fun xs (i + 1) f) (i := (int a))

lemma nth-list-to-fun-eq-aux:
  assumes i-0 <= i and i < length l + i-0
  shows int (l! (i - i-0)) = (list-to-fun l (int i-0) f) (int i)
⟨proof⟩

lemma nth-list-to-fun-eq:
  assumes 0 <= i and i < length l
  shows int (l! i) = (list-to-fun l 0 f) (int i)
⟨proof⟩

A tail-recursive definition makes it even more efficient.

primrec list-to-fun-eff where
  list-to-fun-eff [] :: (f::int => int) = f
  | list-to-fun-eff (a # xs) i f = list-to-fun-eff xs (i + 1) (f(i := (int a)))

lemma list-to-fun-id:
  assumes i-0 > i
  shows list-to-fun-eff l (int i-0) f (int i) = f (int i)
⟨proof⟩

lemma nth-list-to-fun-eff-eq-aux:
assumes \( i-0 \leq i \) and \( i < \text{length } l + i-0 \)
shows \( \text{int} \ l \ (i-0) = (\text{list-to-fun-eff } l \ (\text{int} i-0) \ f) \ (\text{int} i) \)
\langle proof \rangle

lemma nth-list-to-fun-eff-eq:
assumes \( 0 \leq i \) and \( i < \text{length } l \)
shows \( \text{int} \ l \ (i) = (\text{list-to-fun-eff } l \ (\text{int} 0) \ f) \ (\text{int} i) \)
\langle proof \rangle

7.2 Maximum Element of Lists

The following lemmas help the simplifier to prove properties about maximal elements of a list. It is easier to calculate the maximum element of a list in an efficient way (using fold) and prove the correctness of this calculation.

lemma fold-max-leq:
fixes \( i \) \( j \) :: nat
assumes \( i < j \)
shows \( \text{foldl max } i \ l \leq \text{foldl max } j \ l \)
\langle proof \rangle

lemma fold-max-lower:
fixes \( i \) :: nat
shows \( i < \text{foldl max } i \ l \)
\langle proof \rangle

lemma list-max:
fixes \( l \) :: nat list
fixes \( i \) :: nat
assumes \( 0 \leq l \ (i) \)
assumes \( 0 < i \)
assumes \( i < \text{length } l \)
shows \( l \ (i) < \text{foldl max } 0 \ l \)
\langle proof \rangle

lemma list-max-int:
assumes \( l \ (\text{nat } j) \leq \text{foldl max } 0 \ l \)
assumes \( \text{foldl max } 0 \ l = \text{nat } U \)
assumes \( 0 < j \)
assumes \( 0 < U \)
shows \( \text{int} \ (l \ (\text{nat } j)) \leq U \)
\langle proof \rangle

end

8 Verification of \( r_l \)

theory R-L-Spark-Specification
imports Global-Specification R-L-Spark-Declaration

begin

abbreviation r-l' :: int => int where
r-l' := Global-Specification.r-l'

end

theory R-L-Spark-User

imports
R-L-Spark-Specification
R-L-Spark-Declaration
Global-User

begin

lemma goal2'1:
  assumes 0 <= j''
  assumes j'' <= 79
  shows
  (block-permutation---default-arr'')
  (0 := 0, 1 := 1, 2 := 2, 3 := 3, 4 := 4, 5 := 5, 6 := 6, 7 := 7,
  8 := 8, 9 := 9, 10 := 10, 11 := 11, 12 := 12, 13 := 13, 14 := 14,
  15 := 15, 16 := 7, 17 := 4, 18 := 13, 19 := 1, 20 := 10, 21 := 6,
  22 := 15, 23 := 3, 24 := 12, 25 := 0, 26 := 9, 27 := 5, 28 := 2,
  29 := 14, 30 := 11, 31 := 8, 32 := 3, 33 := 10, 34 := 14, 35 := 4,
  36 := 9, 37 := 15, 38 := 8, 39 := 1, 40 := 2, 41 := 7, 42 := 0,
  43 := 6, 44 := 13, 45 := 11, 46 := 5, 47 := 12, 48 := 1, 49 := 9,
  50 := 11, 51 := 10, 52 := 0, 53 := 8, 54 := 12, 55 := 4, 56 := 13,
  57 := 3, 58 := 7, 59 := 15, 60 := 14, 61 := 5, 62 := 6, 63 := 2,
  64 := 4, 65 := 0, 66 := 5, 67 := 9, 68 := 7, 69 := 12, 70 := 2,
  71 := 10, 72 := 14, 73 := 1, 74 := 3, 75 := 8, 76 := 11, 77 := 6,
  78 := 15, 79 := 13))

  j'' =
  R-L-Spark-Specification.r-l' j''

  (proof)

lemma goal2'2:
  assumes 0 <= j''
  assumes j'' <= 79
  shows 0 <= (block-permutation---default-arr'')
  (0 := 0, 1 := 1, 2 := 2, 3 := 3, 4 := 4, 5 := 5, 6 := 6, 7 := 7,
  8 := 8, 9 := 9, 10 := 10, 11 := 11, 12 := 12, 13 := 13, 14 := 14,
  15 := 15, 16 := 7, 17 := 4, 18 := 13, 19 := 1, 20 := 10, 21 := 6,
  22 := 15, 23 := 3, 24 := 12, 25 := 0, 26 := 9, 27 := 5, 28 := 2,
  29 := 14, 30 := 11, 31 := 8, 32 := 3, 33 := 10, 34 := 14, 35 := 4,
  36 := 9, 37 := 15, 38 := 8, 39 := 1, 40 := 2, 41 := 7, 42 := 0,
  43 := 6, 44 := 13, 45 := 11, 46 := 5, 47 := 12, 48 := 1, 49 := 9,
  50 := 11, 51 := 10, 52 := 0, 53 := 8, 54 := 12, 55 := 4, 56 := 13,
  57 := 3, 58 := 7, 59 := 15, 60 := 14, 61 := 5, 62 := 6, 63 := 2,
lemma goal2'3:
assumes 0 <= j''
assumes j'' <= 79
shows (block-permutation---default-arr''
(0 := 0, 1 := 1, 2 := 2, 3 := 3, 4 := 4, 5 := 5, 6 := 6, 7 := 7,
8 := 8, 9 := 9, 10 := 10, 11 := 11, 12 := 12, 13 := 13, 14 := 14,
15 := 15, 16 := 7, 17 := 4, 18 := 13, 19 := 1, 20 := 10, 21 := 6,
22 := 15, 23 := 3, 24 := 12, 25 := 0, 26 := 9, 27 := 5, 28 := 2,
29 := 14, 30 := 11, 31 := 8, 32 := 3, 33 := 10, 34 := 14, 35 := 4,
36 := 9, 37 := 15, 38 := 8, 39 := 1, 40 := 2, 41 := 7, 42 := 0,
43 := 6, 44 := 13, 45 := 11, 46 := 5, 47 := 12, 48 := 1, 49 := 9,
50 := 11, 51 := 10, 52 := 0, 53 := 8, 54 := 12, 55 := 4, 56 := 13,
57 := 3, 58 := 7, 59 := 15, 60 := 14, 61 := 5, 62 := 6, 63 := 2,
64 := 4, 65 := 0, 66 := 5, 67 := 9, 68 := 7, 69 := 12, 70 := 2,
71 := 10, 72 := 14, 73 := 1, 74 := 3, 75 := 8, 76 := 11, 77 := 6,
78 := 15, 79 := 13))
\end
lemma userlemmas = goal2'1 goal2'2 goal2'3
end

9 Verification of \( r_r \)

theory R-R-Spark-Specification
imports Global-Specification R-R-Spark-Declaration
begin

abbreviation r-r' where
r-r' == Global-Specification.r-r'

end
theory R-R-Spark-User
imports R-R-Spark-Specification R-R-Spark-Declaration
Global-User
begin
lemma goal2'.1:
assumes 0 <= j''
assumes j'' <= 79
shows (block-permutation---default-arr'')

(0 := 5, 1 := 14, 2 := 7, 3 := 0, 4 := 9, 5 := 2, 6 := 11, 7 := 4,
8 := 13, 9 := 6, 10 := 15, 11 := 8, 12 := 1, 13 := 10, 14 := 3,
15 := 12, 16 := 6, 17 := 11, 18 := 3, 19 := 7, 20 := 0, 21 := 13,
22 := 5, 23 := 10, 24 := 14, 25 := 15, 26 := 8, 27 := 12, 28 := 4,
29 := 9, 30 := 1, 31 := 2, 32 := 15, 33 := 5, 34 := 1, 35 := 3,
36 := 7, 37 := 14, 38 := 6, 39 := 9, 40 := 11, 41 := 8, 42 := 12,
43 := 2, 44 := 10, 45 := 0, 46 := 4, 47 := 13, 48 := 8, 49 := 6,
50 := 4, 51 := 1, 52 := 3, 53 := 11, 54 := 15, 55 := 0, 56 := 5,
57 := 12, 58 := 2, 59 := 13, 60 := 9, 61 := 7, 62 := 10, 63 := 14,
64 := 12, 65 := 15, 66 := 10, 67 := 4, 68 := 1, 69 := 5, 70 := 8,
71 := 7, 72 := 6, 73 := 2, 74 := 13, 75 := 14, 76 := 0, 77 := 3,
78 := 9, 79 := 11))

j'' = R-R-Spark-Specification.r-r' j''
(proof)

lemma goal2'.2:
assumes 0 <= j''
assumes j'' <= 79
shows 0 <= (block-permutation---default-arr'')

(0 := 5, 1 := 14, 2 := 7, 3 := 0, 4 := 9, 5 := 2, 6 := 11, 7 := 4,
8 := 13, 9 := 6, 10 := 15, 11 := 8, 12 := 1, 13 := 10, 14 := 3,
15 := 12, 16 := 6, 17 := 11, 18 := 3, 19 := 7, 20 := 0, 21 := 13,
22 := 5, 23 := 10, 24 := 14, 25 := 15, 26 := 8, 27 := 12, 28 := 4,
29 := 9, 30 := 1, 31 := 2, 32 := 15, 33 := 5, 34 := 1, 35 := 3,
36 := 7, 37 := 14, 38 := 6, 39 := 9, 40 := 11, 41 := 8, 42 := 12,
43 := 2, 44 := 10, 45 := 0, 46 := 4, 47 := 13, 48 := 8, 49 := 6,
50 := 4, 51 := 1, 52 := 3, 53 := 11, 54 := 15, 55 := 0, 56 := 5,
57 := 12, 58 := 2, 59 := 13, 60 := 9, 61 := 7, 62 := 10, 63 := 14,
64 := 12, 65 := 15, 66 := 10, 67 := 4, 68 := 1, 69 := 5, 70 := 8,
71 := 7, 72 := 6, 73 := 2, 74 := 13, 75 := 14, 76 := 0, 77 := 3,
78 := 9, 79 := 11))

j''
(proof)

lemma goal2'.3:
assumes 0 <= j''
assumes j'' <= 79
shows (block-permutation---default-arr'')

(0 := 5, 1 := 14, 2 := 7, 3 := 0, 4 := 9, 5 := 2, 6 := 11, 7 := 4,
8 := 13, 9 := 6, 10 := 15, 11 := 8, 12 := 1, 13 := 10, 14 := 3,
15 := 12, 16 := 6, 17 := 11, 18 := 3, 19 := 7, 20 := 0, 21 := 13,
22 := 5, 23 := 10, 24 := 14, 25 := 15, 26 := 8, 27 := 12, 28 := 4,

\[ j'' \leq 15 \]

\langle \text{proof} \rangle

lemmas userlemmas = goal2'2 goal2'3 goal2'1

end

10 Verification of \( s_l \)

theory S-L-Spark-Specification
imports Global-Specification S-L-Spark-Declaration

begin

abbreviation \( s-l' : \) int => int where
\( s-l' := \) Global-Specification.s-l'

end

theory S-L-Spark-User
imports
S-L-Spark-Specification
S-L-Spark-Declaration
Global-User

begin

lemma goal2'1:
assumes 0 <= j''
assumes j'' <= 79
shows \( \text{(rotate-definition---default-arr''} \)
lemma goal2'2:
  assumes 0 <= j''
  assumes j'' <= 79
  shows 0 <= (rotate-definition---default-arr'')

proof

lemma goal2'3:
  assumes 0 <= j''
  assumes j'' <= 79
  shows (rotate-definition---default-arr'')

proof

lemmas userlemmas = goal2'2 goal2'3 goal2'1
end

11 Verification of \(s_r\)

theory S-R-Spark-Specification
imports Global-Specification S-R-Spark-Declaration
begin

abbreviation \(s-r' \:: \text{int \Rightarrow int\ where} \)
\( s-r' @== \text{Global-Specification.s-r'} \)
end

theory S-R-Spark-User
imports
S-R-Spark-Specification
S-R-Spark-Declaration
Global-User
begin

lemma goal2'1:
assumes 0 <= j''
assumes j'' <= 79
shows (rotate-definition---default-arr'')
\((0 @:= 8, 1 @:= 9, 2 @:= 9, 3 @:= 11, 4 @:= 13, 5 @:= 15, 6 @:= 15, 7 @:= 5, \)
\(8 @:= 7, 9 @:= 7, 10 @:= 8, 11 @:= 11, 12 @:= 14, 13 @:= 14, 14 @:= 12, \)
\(15 @:= 6, 16 @:= 9, 17 @:= 13, 18 @:= 15, 19 @:= 7, 20 @:= 12, 21 @:= 8, \)
\(22 @:= 9, 23 @:= 11, 24 @:= 7, 25 @:= 7, 26 @:= 12, 27 @:= 7, 28 @:= 6, \)
\(29 @:= 15, 30 @:= 13, 31 @:= 11, 32 @:= 9, 33 @:= 7, 34 @:= 15, 35 @:= 11, \)
\(36 @:= 8, 37 @:= 6, 38 @:= 6, 39 @:= 14, 40 @:= 12, 41 @:= 13, 42 @:= 5, \)
\(43 @:= 14, 44 @:= 13, 45 @:= 13, 46 @:= 7, 47 @:= 5, 48 @:= 15, 49 @:= 5, \)
\(50 @:= 8, 51 @:= 11, 52 @:= 14, 53 @:= 14, 54 @:= 6, 55 @:= 14, 56 @:= 6, \)
\(57 @:= 9, 58 @:= 12, 59 @:= 9, 60 @:= 12, 61 @:= 5, 62 @:= 15, 63 @:= 8, \)
\(64 @:= 8, 65 @:= 5, 66 @:= 12, 67 @:= 9, 68 @:= 12, 69 @:= 5, 70 @:= 14, \)
\(71 @:= 6, 72 @:= 8, 73 @:= 13, 74 @:= 6, 75 @:= 5, 76 @:= 15, 77 @:= 13, \)
\(78 @:= 11, 79 @:= 11)) \)
\(j'' @== \)
\(S-R-Spark-Specification.s-r' j'' \)
(proof)

lemma goal2'2:
assumes 0 <= j''
assumes j'' <= 79
shows 0 <= (rotate-definition---default-arr''
\((0 @:= 8, 1 @:= 9, 2 @:= 9, 3 @:= 11, 4 @:= 13, 5 @:= 15, 6 @:= 15, 7 @:= 5, \)
\(8 @:= 7, 9 @:= 7, 10 @:= 8, 11 @:= 11, 12 @:= 14, 13 @:= 14, 14 @:= 12, \)
\(15 @:= 6, 16 @:= 9, 17 @:= 13, 18 @:= 15, 19 @:= 7, 20 @:= 12, 21 @:= 8, \)
begin

abbreviation bit--and' :: [int , int ] => int where
  bit--and' == Global-Specification.bit--and'
abbreviation bit--or' :: [int , int ] => int where
  bit--or' == Global-Specification.bit--or'
abbreviation bit--xor' :: [int , int ] => int where

end

12 Verification of round

theory Round-Specification
imports Global-Specification Round-Declaration

begin
\[ \text{bit-xor}' = \text{Global-Specification.bit-xor}' \]

**abbreviation** \( f' :: [\text{int}, \text{int}, \text{int}, \text{int}] \rightarrow \text{int} \) where
\[ f' = \text{Global-Specification.f} \]

**abbreviation** \( k-l' :: \text{int} \rightarrow \text{int} \) where
\[ k-l' = \text{Global-Specification.k-l}' \]

**abbreviation** \( k-r' :: \text{int} \rightarrow \text{int} \) where
\[ k-r' = \text{Global-Specification.k-r}' \]

**abbreviation** \( r-l' :: \text{int} \rightarrow \text{int} \) where
\[ r-l' = \text{Global-Specification.r-l}' \]

**abbreviation** \( r-r' :: \text{int} \rightarrow \text{int} \) where
\[ r-r' = \text{Global-Specification.r-r}' \]

**abbreviation** \( \text{wordops--rotate-left}' :: [\text{int}, \text{int}] \rightarrow \text{int} \) where
\[ \text{wordops--rotate-left}' = \text{Global-Specification.rotate-left}' \]

**abbreviation** \( s-l' :: \text{int} \rightarrow \text{int} \) where
\[ s-l' = \text{Global-Specification.s-l}' \]

**abbreviation** \( s-r' :: \text{int} \rightarrow \text{int} \) where
\[ s-r' = \text{Global-Specification.s-r}' \]

**abbreviation** from-chain :: \( \text{chain}' \rightarrow \text{chain} \) where
\[
\text{from-chain } c \Rightarrow (\text{word-of-int}(h0'\text{chain } c), \text{word-of-int}(h1'\text{chain } c), \text{word-of-int}(h2'\text{chain } c), \text{word-of-int}(h3'\text{chain } c), \text{word-of-int}(h4'\text{chain } c))
\]

**abbreviation** from-chain-pair :: \( \text{chain-pair}' \rightarrow \text{chain} * \text{chain} \) where
\[
\text{from-chain-pair } cc \Rightarrow (\text{from-chain}(\text{left}'\text{chain-pair } cc), \text{from-chain}(\text{right}'\text{chain-pair } cc))
\]

**abbreviation** to-chain :: \( \text{chain} \Rightarrow \text{chain}' \) where
\[
\text{to-chain } c \Rightarrow (\text{let } (h0, h1, h2, h3, h4) = c \text{ in } \text{chain---default-rcd}' (| h0'\text{chain } := \text{uint } h0, h1'\text{chain } := \text{uint } h1, h2'\text{chain } := \text{uint } h2, h3'\text{chain } := \text{uint } h3, h4'\text{chain } := \text{uint } h4))
\]

**abbreviation** to-chain-pair :: \( \text{chain} * \text{chain} \Rightarrow \text{chain-pair}' \) where
\[
\text{to-chain-pair } c \Rightarrow (\text{let } (c1, c2) = c \text{ in } (| \text{left}'\text{chain-pair } = \text{to-chain } c1, \text{right}'\text{chain-pair } = \text{to-chain } c2 |))
\]

**abbreviation** steps' :: [\(\text{chain-pair}', \text{int}, \text{block}] \Rightarrow \text{chain-pair}' \) where
\[
\text{steps'} cc i b \Rightarrow \text{to-chain-pair } (\text{steps...})
\]
abbreviation \texttt{round'} :: \{ \text{chain}', \text{block}' \} \Rightarrow \text{chain} \quad \text{where} \\
\texttt{round'} c b \equiv \text{to-chain} (\text{round} (%n. \text{word-of-int} (b \ (\text{int} \ n))) \ (\text{from-chain} \ c))

end

theory \texttt{Round-User}

imports \texttt{Round-Specification Round-Declaration}

begin

lemma \texttt{uint-word-of-int-id}:
  assumes \(0 <\leq (x ::\text{int})\)
  assumes \(x <\leq 4294967295\)
  shows \(\text{uint} (\text{word-of-int} x ::\text{word32}) = x\)
⟨proof⟩

lemma \texttt{steps-step}: \(\text{steps} \ X \ cc \ (\text{Suc} \ i) = \text{step-both} \ X \ (\text{steps} \ X \ cc \ i) \ i\)
⟨proof⟩

lemma \texttt{from-to-id}:
\(\text{from-chain-pair} \ (\text{to-chain-pair} \ CC) = CC\)
⟨proof⟩

lemma \texttt{steps-to-steps'}:
\(F \ A \ (\text{steps} \ X \ cc \ i) \ B = F \ A \ (\text{from-chain-pair} \ (\text{to-chain-pair} \ (\text{steps} \ X \ cc \ i))) \ B\)
⟨proof⟩

lemma \texttt{steps'}-step:
  assumes \(0 <\leq i\)
  shows \(\text{steps'} \ cc \ (i + 1) \ X = \text{to-chain-pair} \ (\text{step-both} \ )
  \(\ (\lambda n. \text{word-of-int} \ (X \ (\text{int} \ n)))\)
  \(\ (\text{from-chain-pair} \ (\text{steps'} \ cc \ i \ X))\)
  \(\ (\text{nat} \ i)\)
⟨proof⟩

lemma \texttt{step-from-hyp}:
  fixes \(a \ b \ c \ d \ e\)
  fixes \(a' \ b' \ c' \ d' \ e'\)
  fixes \(a-0 \ b-0 \ c-0 \ d-0 \ e-0\)
  fixes \(x\)
  fixes \(j\)
  assumes \texttt{step-hyp}:
\[
\text{chain-pair---default-rcd}''
\]
\[(\text{left'}\text{chain-pair} := \text{chain}---\text{default-rcd}''\]
\[(\text{h0'}\text{chain} := a, \text{h1'}\text{chain} := b, \text{h2'}\text{chain} := c, \text{h3'}\text{chain} := d, \]
\[\text{h4'}\text{chain} := e)\),
\]{
\[\text{right'}\text{chain-pair} := \text{chain}---\text{default-rcd}''\]
\[(\text{h0'}\text{chain} := a', \text{h1'}\text{chain} := b', \text{h2'}\text{chain} := c', \text{h3'}\text{chain} := d', \]
\[\text{h4'}\text{chain} := e')\} = \]
\[\text{steps'}\]
\[(\text{chain-pair---default-rcd}''\]
\[(\text{left'}\text{chain-pair} := \text{chain}---\text{default-rcd}''\]
\[(\text{h0'}\text{chain} := a-0, \text{h1'}\text{chain} := b-0, \text{h2'}\text{chain} := c-0, \]
\[\text{h3'}\text{chain} := d-0, \text{h4'}\text{chain} := e-0)\),
\]{
\[\text{right'}\text{chain-pair} := \text{chain}---\text{default-rcd}''\]
\[(\text{h0'}\text{chain} := a-0, \text{h1'}\text{chain} := b-0, \text{h2'}\text{chain} := c-0, \]
\[\text{h3'}\text{chain} := d-0, \text{h4'}\text{chain} := e-0)\}]
\[
\text{j} x
\]
\text{assumes a-borders: } 0 <= a <= 4294967295 (\text{is} <= ?M)
\text{assumes b-borders: } 0 <= b b <= ?M
\text{assumes c-borders: } 0 <= c c <= ?M
\text{assumes d-borders: } 0 <= d d <= ?M
\text{assumes e-borders: } 0 <= e e <= ?M
\text{assumes a'-borders: } 0 <= a' a' <= ?M
\text{assumes b'-borders: } 0 <= b' b' <= ?M
\text{assumes c'-borders: } 0 <= c' c' <= ?M
\text{assumes d'-borders: } 0 <= d' d' <= ?M
\text{assumes e'-borders: } 0 <= e' e' <= ?M
\text{assumes x-borders: } 0 <= x (r-l') x (r-l') <= ?M
\text{ 0 <= x (r-r') x (r-r') <= ?M}
\text{assumes j-borders: } 0 <= j j <= 79
\text{shows}
\text{chain-pair---default-rcd}''
\[(\text{left'}\text{chain-pair} := \text{chain}---\text{default-rcd}''\]
\[(\text{h0'}\text{chain} := e, \]
\[\text{h1'}\text{chain} := \]
\[(\text{wordops--rotate-left}' (s-l' j) \]
\[(((a + f' j b c d) \text{mod} 4294967296 + \]
\[x (r-l' j)) \text{mod} \]
\[4294967296 + \]
\[k-l' j) \text{mod} \]
\[4294967296) + \]
\[e) \text{mod} \]
\[4294967296, \]
\[\text{h2'}\text{chain} := b, \text{h3'}\text{chain} := \text{wordops--rotate-left}' 10 c, \]
\[\text{h4'}\text{chain} := d)\),
\]{
\[\text{right'}\text{chain-pair} := \text{chain}---\text{default-rcd}''\]
\[(\text{h0'}\text{chain} := e', \]
\[\text{h1'}\text{chain} := \]
\[(\text{wordops--rotate-left}' (s-r' j) \]
\[(((a' + f' (79 - j) b' c' d') \text{mod} \]
\[4294967296) + \]
\[e' j (r-r' j)) \text{mod} \]
\[4294967296) + \]
\[e j) \text{mod} \]
\[4294967296, \]
\[\text{h2'}\text{chain} := b, \text{h3'}\text{chain} := \text{wordops--rotate-left}' 10 c, \]
\[\text{h4'}\text{chain} := d)\),
\]{
\text{23}
4294967296 +
x (r-r' j)) mod 4294967296 +
k-r' j) mod 4294967296 +
e') mod 4294967296,

h2'chain := b', h3'chain := wordops--rotate-left' 10 c',
h4'chain := d'][j] =

steps'
(chain-pair---default-red''
(left'chain-pair := chain---default-red''
(h0'chain := a-0, h1'chain := b-0, h2'chain := c-0,
h3'chain := d-0, h4'chain := e-0),
right'chain-pair := chain---default-red''
(h0'chain := a-0, h1'chain := b-0, h2'chain := c-0,
h3'chain := d-0, h4'chain := e-0)])

(j + 1) x

(proof)

abbreviation
f-0-result == (((ca'' + f-spark' 0 cb'' cc'' cd'') mod 4294967296 +
x'' (r-l-spark' 0)) mod 4294967296 + k-l-spark' 0) mod 4294967296

abbreviation
f-79-result == (((ca'' + f-spark' 79 cb'' cc'' cd'') mod 4294967296 +
x'' (r-r-spark' 0)) mod 4294967296 + k-r-spark' 0) mod 4294967296

lemma goal61'1:
assumes ca-borders: 0 <= ca'' <= ca' <= 4294967295 (is - <= ?M)
assumes cb-borders: 0 <= cb'' <= cb' <= ?M
assumes cc-borders: 0 <= cc'' <= cc' <= ?M
assumes cd-borders: 0 <= cd'' <= cd' <= ?M
assumes ce-borders: 0 <= ce'' <= ce' <= ?M
assumes r-l-0-borders: 0 <= r-l-spark' 0 r-l-spark' 0 <= 15
assumes r-r-0-borders: 0 <= r-r-spark' 0 r-r-spark' 0 <= 15
assumes returns:
wordops--rotate' (s-l-spark' 0) f-0-result =
wordops--rotate-left' (s-l-spark' 0) f-0-result
wordops--rotate' (s-r-spark' 0) f-79-result =
wordops--rotate-left' (s-r-spark' 0) f-79-result
wordops--rotate' 10 cc'' = wordops--rotate-left' 10 cc''
f-spark' 0 cb'' cc'' cd'' = f' 0 cb'' cc'' cd''
f-spark' 79 cb'' cc'' cd'' = f' 79 cb'' cc'' cd''
k-l-spark' 0 = k-l' 0
k-r-spark' 0 = k-r' 0
r-l-spark' 0 = r-l' 0
r-r-spark' 0 = r-r' 0
s-l-spark' 0 = s-l' 0
s-r-spark' 0 = s-r' 0

24
assumes $x$-borders: $\forall i. 0 \leq i \land i \leq 15 \rightarrow 0 \leq x'' i \leq x'' i \leq ?M$

shows chain-pair---default-rcd''

\(\left\{ \text{left'chain-pair := chain---default-rcd''} \right\} \)
\(\{h0\text{'chain := ce''}, \quad h1\text{'chain :=} \)
\((\text{wordops--rotate'} (s-l-spark' 0) \)
\(((((ca'' + f-spark' 0 cb'' cc'' cd'') \mod 4294967296 + \)
\(x'' (r-l-spark' 0)) \mod 4294967296 + \)
\(k-l-spark' 0) \mod 4294967296) + \)
\(ce'') \mod 4294967296, \)
\(h2\text{'chain := cb''}, h3\text{'chain := wordops--rotate'} 10 cc'', \)
\(h4\text{'chain := cd''}) \}

\(\right\{ \text{right'chain-pair := chain---default-rcd''} \right\} \)
\(\{h0\text{'chain := ce''}, \quad h1\text{'chain :=} \)
\((\text{wordops--rotate'} (s-r-spark' 0) \)
\(((\text{ca'' + f-spark'} 79 cb'' cc'' cd'') \mod 4294967296 + \)
\((x'' (r-r-spark' 0)) \mod 4294967296 + \)
\(k-r-spark' 0) \mod 4294967296) + \)
\(ce'') \mod 4294967296, \)
\(h2\text{'chain := cb''}, h3\text{'chain := wordops--rotate'} 10 cc'', \)
\(h4\text{'chain := cd''}) \}

steps'

\(\left\{ \text{chain-pair---default-rcd''} \right\} \)
\(\left\{ \text{left'chain-pair := chain---default-rcd''} \right\} \)
\(\{h0\text{'chain := ca''}, h1\text{'chain := cb''}, h2\text{'chain := cc'',} \)
\(h3\text{'chain := cd''}, h4\text{'chain := ce''}) \}

\(\right\{ \text{right'chain-pair := chain---default-rcd''} \right\} \)
\(\{h0\text{'chain := ca''}, h1\text{'chain := cb''}, h2\text{'chain := cc'',} \)
\(h3\text{'chain := cd''}, h4\text{'chain := ce''}) \}

1 $x''$

\(\langle \text{proof} \rangle\)

abbreviation rotate-arg-l ==
\(((\text{cla'' + f-spark'} (\text{loop--1-j'' + 1}) \text{clb'' cle'' cld''}) \mod 4294967296 + \)
\(x'' (r-l-spark' (\text{loop--1-j'' + 1})) \mod 4294967296 + \)
\(k-l-spark' (\text{loop--1-j'' + 1}) \mod 4294967296) \)

abbreviation rotate-arg-r ==\(((\text{cre'' + f-spark'} (79 - (\text{loop--1-j'' + 1})) \text{crb'' crc'' crd''}) \mod 4294967296 + \)
\(x'' (r-r-spark' (\text{loop--1-j'' + 1})) \mod 4294967296 + \)
\(k-r-spark' (\text{loop--1-j'' + 1}) \mod 4294967296) \)
lemma goal62':1:
assumes cla-borders: 0 <= cla" <= 4294967295 (is - <= ?M)
assumes clb-borders: 0 <= clb" clb" <= ?M
assumes cle-borders: 0 <= cle" cle" <= ?M
assumes cld-borders: 0 <= cld" cld" <= ?M
assumes cle-borders: 0 <= cle" cle" <= ?M
assumes cra-borders: 0 <= cra" cra" <= ?M
assumes crb-borders: 0 <= crb" crb" <= ?M
assumes crc-borders: 0 <= crc" crc" <= ?M
assumes crd-borders: 0 <= crd" crd" <= ?M
assumes step-hyp:
  chain-pair---default-rcd''
   (k-l-spark\' wordops--rotate' wordops--rotate-left) rotate-arg-l =
   wordops---rotate-left' (s-l-spark' (loop--1-j'' + 1)) rotate-arg-l
   wordops---rotate' (s-r-spark' (loop--1-j'' + 1)) rotate-arg-r =
   wordops---rotate-left' (s-r-spark' (loop--1-j'' + 1)) rotate-arg-r
   f-spark' (loop--1-j'' + 1) clb" clc" cld" =
   f' (loop--1-j'' + 1) clb" clc" cld" =
   f' (78 - loop--1-j") crb" crc" crd" =
   wordops---rotate' 10 clc" = wordops---rotate-left' 10 clc"
   wordops---rotate' 10 crc" = wordops---rotate-left' 10 crc"
   k-l-spark' (loop--1-j'' + 1) = k-l' (loop--1-j'' + 1)
   k-r-spark' (loop--1-j'' + 1) = k-r' (loop--1-j'' + 1)
   r-l-spark' (loop--1-j'' + 1) = r-l' (loop--1-j'' + 1)
   r-r-spark' (loop--1-j'' + 1) = r-r' (loop--1-j'' + 1)
   s-l-spark' (loop--1-j'' + 1) = s-l' (loop--1-j'' + 1)
   s-r-spark' (loop--1-j'' + 1) = s-r' (loop--1-j'' + 1)
assumes x-borders: \forall i. 0 <= i & i <= 15 -> 0 <= x" i & x" i <= ?M
assumes r-l-borders:
\[0 \leq r-l-spark' \text{(loop--}1-j'' + 1) \leq 15\]
assumes \(r-r\)-borders:
\[0 \leq r-r-spark' \text{(loop--}1-j'' + 1) \leq 15\]
assumes \(j\)-loop-1-borders: \(0 \leq \text{loop--}1-j'' \text{ loop--}1-j'' \leq 78\)
shows chain-pair---default-rcd''
\[\langle \text{left'chain-pair} := \text{chain---default-rcd''} \rangle\]
\[\langle h0'chain := \text{cle''}, h1'chain := (\text{wordops--rotate'} (s-l-spark' \text{(loop--}1-j'' + 1))) \rangle\]
\[(((\text{cla'' + f-spark'} \text{(loop--}1-j'' + 1) \text{ clb clc cld''}) \mod 4294967296 + \text{x'' \text{(r-l-spark'} \text{(loop--}1-j'' + 1) \text{)}} \mod 4294967296 + k-l-spark' \text{(loop--}1-j'' + 1) \text{ mod 4294967296}) + \text{cle''}) \mod 4294967296, h2'chain := \text{clb''}, h3'chain := \text{wordops--rotate'} 10 \text{ cle''}, h4'chain := \text{cld''} \rangle\]
\[\langle \text{right'chain-pair} := \text{chain---default-rcd''} \rangle\]
\[\langle h0'chain := \text{cre''}, h1'chain := (\text{wordops--rotate'} (s-r-spark' \text{(loop--}1-j'' + 1))) \rangle\]
\[(((\text{cra'' + f-spark'} (79 - (\text{loop--}1-j'' + 1)) \text{ crb'' crc'' \text{ crd''}}) \mod 4294967296 + \text{x'' \text{(r-r-spark'} \text{(loop--}1-j'' + 1) \text{)}} \mod 4294967296 + k-r-spark' \text{(loop--}1-j'' + 1) \text{ mod 4294967296}) + \text{cre''}) \mod 4294967296, h2'chain := \text{crb''}, h3'chain := \text{wordops--rotate'} 10 \text{ crc''}, h4'chain := \text{crd''} \rangle\]
\[\langle \text{steps'} \rangle\]
\[\langle \text{chain-pair---default-rcd''} \rangle\]
\[\langle \text{left'chain-pair} := \text{chain---default-rcd''} \rangle\]
\[\langle h0'chain := \text{ca---init''}, h1'chain := \text{cb---init''}, h2'chain := \text{cc---init''}, h3'chain := \text{cd---init''}, h4'chain := \text{ce---init''} \rangle\]
\[\langle \text{right'chain-pair} := \text{chain---default-rcd''} \rangle\]
\[\langle h0'chain := \text{ca---init''}, h1'chain := \text{cb---init''}, h2'chain := \text{cc---init''}, h3'chain := \text{cd---init''}, h4'chain := \text{ce---init''} \rangle\]
\[\langle \text{(loop--}1-j'' + 2) \text{ x''} \rangle\]
\[\langle \text{proof} \rangle\]
abbreviation INIT-CHAIN == chain---default-rcd'"
lemma goal76'1:
  assumes cla-borders: 0 <= cla'' cla'' <= 4294967295 (is - <= ?M)
  assumes clb-borders: 0 <= clb'' clb'' <= ?M
  assumes clc-borders: 0 <= clc'' clc'' <= ?M
  assumes cld-borders: 0 <= cld'' cld'' <= ?M
  assumes cle-borders: 0 <= cle'' cle'' <= ?M
  assumes cra-borders: 0 <= cra'' cra'' <= ?M
  assumes crb-borders: 0 <= crb'' crb'' <= ?M
  assumes crc-borders: 0 <= crc'' crc'' <= ?M
  assumes crd-borders: 0 <= crd'' crd'' <= ?M
  assumes cre-borders: 0 <= cre'' cre'' <= ?M
  assumes ca-init-borders: 0 <= ca-~~ init'' ca-~~ init'' <= ?M
  assumes cb-init-borders: 0 <= cb-~~ init'' cb-~~ init'' <= ?M
  assumes cc-init-borders: 0 <= cc-~~ init'' cc-~~ init'' <= ?M
  assumes cd-init-borders: 0 <= cd-~~ init'' cd-~~ init'' <= ?M
  assumes ce-init-borders: 0 <= ce-~~ init'' ce-~~ init'' <= ?M
  assumes step-hyp:
    chain-pair-~~ default-rcd''
    (left'chain-pair := chain-~~ default-rcd''
      (h0'chain := cla'', h1'chain := clb'', h2'chain := cle'', h3'chain := cld'')
      h4'chain := cle''))
    (right'chain-pair := chain-~~ default-rcd''
      (h0'chain := cra'', h1'chain := crb'', h2'chain := crc'', h3'chain := crd'')
      h4'chain := cre''))
    steps'
    (chain-pair-~~ default-rcd''
      (left'chain-pair := chain-~~ default-rcd''
        (h0'chain := ca-~~ init'', h1'chain := cb-~~ init'', h2'chain := cc-~~ init'')
        h3'chain := cd-~~ init'')
        h4'chain := ce-~~ init''))
    (right'chain-pair := chain-~~ default-rcd''
      (h0'chain := ca-~~ init'', h1'chain := cb-~~ init'', h2'chain := cc-~~ init'')
      h3'chain := cd-~~ init'',
      h4'chain := ce-~~ init''))
  80 x''
  shows chain-~~ default-rcd''
    (h0'chain := ((cb-~~ init'' + cle'') mod 4294967296 + crd'') mod 4294967296,
     h1'chain := ((cc-~~ init'' + clb'') mod 4294967296 + cre'') mod 4294967296,
     h2'chain := ((cd-~~ init'' + cle'') mod 4294967296 + cra'') mod 4294967296,
     h3'chain := ((ce-~~ init'' + cla'') mod 4294967296 + crb'') mod 4294967296,
     h4'chain := ((ca-~~ init'' + clb'') mod 4294967296 + crc'') mod 4294967296)
    =
    round'
    (chain-~~ default-rcd'')
\[(h0'\text{chain} := ca---init'', h1'\text{chain} := cb---init'', h2'\text{chain} := cc---init'', h3'\text{chain} := cd---init'', h4'\text{chain} := cc---init'\emptyset)\]

\(x''\)

\langle proof \rangle

\begin{lemmas}
userlemmas = goal61'1 goal62'1 goal76'1
\end{lemmas}

end

\section{Verification of hash}

\begin{theory}
Hash-Specification
\end{theory}

\begin{imports}
Hash-Declaration Global-Specification
\end{imports}

begin

\begin{abbreviation}
\begin{from-chain} \begin{mapping}\begin{from-chain} \begin{where}
\end{where}\end{from-chain}\end{mapping}\end{from-chain} where
\end{abbreviation}

\begin{abbreviation}
\begin{to-chain} \begin{where}
\end{where}\end{to-chain} where
\end{abbreviation}

\begin{abbreviation}
\begin{round'} \begin{where}
\end{where}\end{round'} where
\end{abbreviation}

\begin{abbreviation}
\begin{rounds'} \begin{where}
\end{where}\end{rounds'} where
\end{abbreviation}

\begin{abbreviation}
\begin{rmd-hash'} \begin{where}
\end{where}\end{rmd-hash'} where
\end{abbreviation}

29
theory Hash-User
imports Hash-Specification Hash-Declaration
begin

lemma goal12':
  assumes H1: x--index--subtype--1--first'' = (0 :: int)
  assumes H6:
    chain---default-rcd''
    (| h0'chain
      := ca--1'' |
    )
    (| h1'chain
      := cb--1'' |
    )
    (| h2'chain
      := cc--1'' |
    )
    (| h3'chain
      := cd--1'' |
    )
    (| h4'chain
      := ce--1'' |
    )
    = round'
      ( chain---default-rcd''
        (| h0'chain
          := (1732584193 :: int) |
        )
        (| h1'chain
          := (4023233417 :: int) |
        )
        (| h2'chain
          := (2562383102 :: int) |
        )
        (| h3'chain
          := (271733878 :: int) |
        )
        (| h4'chain
          := (3285377520 :: int) |
        )
      ) ( x'' x--index--subtype--1--first'' )
shows \( \text{chain---default-rcd''} \)
\[
\begin{align*}
( & h_0' \text{chain} \\
& : c_a-1'' ) \\
( & h_1' \text{chain} \\
& : c_b-1'' ) \\
( & h_2' \text{chain} \\
& : c_c-1'' ) \\
( & h_3' \text{chain} \\
& : c_d-1'' ) \\
( & h_4' \text{chain} \\
& : c_e-1'' )
\end{align*}
\]
\(= \text{rounds'}( \text{chain---default-rcd''} \)
\[
\begin{align*}
( & h_0' \text{chain} \\
& : (1732584193 :: \text{int}) ) \\
( & h_1' \text{chain} \\
& : (4023233417 :: \text{int}) ) \\
( & h_2' \text{chain} \\
& : (2562383102 :: \text{int}) ) \\
( & h_3' \text{chain} \\
& : (271733878 :: \text{int}) ) \\
( & h_4' \text{chain} \\
& : (3285377520 :: \text{int}) )
\end{align*}
\]
\(x--\text{index--subtype--1--first''} + (1 :: \text{int}) \)
\(x'' \)
\(\langle \text{proof} \rangle \)
\(\text{lemma} \ \text{rounds-step}: \)
\(\text{assumes} \ \theta \in\leq i \)
\(\text{shows} \ \text{rounds} \ X \ b \ (\text{Suc} \ i) = \text{round} \ (X \ i) \ (\text{rounds} \ X \ b \ i) \)
\(\langle \text{proof} \rangle \)
\(\text{lemma} \ \text{from-to-id}: \ \text{from-chain} \ (\text{to-chain} \ C) = C \)
\(\langle \text{proof} \rangle \)
\(\text{lemma} \ \text{steps-to-steps}: \)
round \( X \) \((\text{fold}_l \ a \ b \ c)\) = round \( X \) \((\text{from-chain} \ (\text{to-chain} \ (\text{fold}_l \ a \ b \ c)))\)

\text{lemma} \ rounds'-step:
\begin{itemize}
\item \text{assumes} \ 0 \leq i
\item \text{shows} \ rounds' \ c \ (i + 1) \ x = \text{round}' (\text{rounds'} \ c \ i \ x) \ (x \ i)
\end{itemize}

\text{lemma} \ goal13'1:
\begin{itemize}
\item \text{assumes} \ 0 \leq \text{loop--1--i}''
\item \text{assumes} \ H1:
\begin{itemize}
\item \text{chain---default-rcd}''
\item \( h0' \text{chain} := ca'' \)
\item \( h1' \text{chain} := cb'' \)
\item \( h2' \text{chain} := cc'' \)
\item \( h3' \text{chain} := cd'' \)
\item \( h4' \text{chain} := ce'' \)
\end{itemize}
\item \( \text{rounds}' \)
\begin{itemize}
\item \text{chain---default-rcd}''
\item \( h0' \text{chain} := (1732584193 :: \text{int}) \)
\item \( h1' \text{chain} := (4023233417 :: \text{int}) \)
\item \( h2' \text{chain} := (2562383102 :: \text{int}) \)
\item \( h3' \text{chain} := (271733878 :: \text{int}) \)
\item \( h4' \text{chain} := (3285377520 :: \text{int}) \)
\item \( \text{loop--1--i}'' + (1 :: \text{int}) \)
\end{itemize}
\item \text{x}''
\end{itemize}

\text{assumes} \ H18:
shows %chain---default-rcd''
( | h0'chain
   := ca-1''
 |)
( | h1'chain
   := cb-1''
 |)
( | h2'chain
   := cc-1''
 |)
( | h3'chain
   := cd-1''
 |)
( | h4'chain
   := ce-1''
 |)
= round' (%chain---default-rcd''
 ( | h0'chain
   := ca''
 |)
( | h1'chain
   := cb''
 |)
( | h2'chain
   := cc''
 |)
( | h3'chain
   := cd''
 |)
( | h4'chain
   := ce''
 |)
)
( x'' ( loop--1--i'' + (1 :: int) )
)

\[
\]
(\ h4'chain \\
\ = \ rounds' \\
\ h0'chain := 1732584193 :: int \\
\ h1'chain := 4023233417 :: int \\
\ h2'chain := 2562383102 :: int \\
\ h3'chain := 271733878 :: int \\
\ h4'chain := 3285377520 :: int \\
\ loop--1--i'' + (2 :: int) \)
\[
\langle \text{proof} \rangle
\]

\textbf{lemma} \ goal17'1: \hspace{1cm} \\
\textbf{assumes} \ H1: \hspace{1cm}
chain---default-rcd''
\ h0'chain := ca'' \\
\ h1'chain := cb'' \\
\ h2'chain := cc'' \\
\ h3'chain := cd'' \\
\ h4'chain := ce'' \\
\ = \ rounds' \\
( \ chain---default-rcd'' \\
\ h0'chain
\)
\]

34
\[ h1' \text{chain} := (1732584193 :: \text{int}) \]
\[ h2' \text{chain} := (4023233417 :: \text{int}) \]
\[ h3' \text{chain} := (2562383102 :: \text{int}) \]
\[ h4' \text{chain} := (271733878 :: \text{int}) \]
\[ h5' \text{chain} := (3285377520 :: \text{int}) \]
\[ \text{x--index--subtype--1--last}'' + (1 :: \text{int}) \]

\( x'' \)

shows \( \text{chain}---\text{default-rcd}'' \)
\[ h0' \text{chain} := ca'' \]
\[ h1' \text{chain} := cb'' \]
\[ h2' \text{chain} := ce'' \]
\[ h3' \text{chain} := cd'' \]
\[ h4' \text{chain} := ce'' \]
\[ = \text{rmd-hash}' \]
\[ x'' (\text{x--index--subtype--1--last}'' + (1 :: \text{int}) \) \]

⟨proof⟩

lemmas userlemmas = goal12'1 goal13'1 goal17'1
end

References

