Packing and unpacking routines

```
--  This chapter contains routines for packing and unpacking bits
--  and bytes in bytes and blocks.

--
--  access to bytes in blocks
--

VAL INT bytes.in.a.block IS 512 :

PROTOCOL BLOCK IS [bytes.in.a.block]BYTE :

--
--  accesses to bits in bytes
--

VAL INT bits.in.a.byte IS 8 :

PROTOCOL BIT IS BOOL :

PROC set.bit(BYTE byte, VAL INT bit.number, VAL BOOL bit)
  IF
    NOT bit
      byte := BYTE ((INT byte) /\ (~(1 << bit.number)))
    bit
      byte := BYTE ((INT byte) \/ (1 << bit.number))
:

BOOL FUNCTION get.bit(VAL BYTE byte, VAL INT bit.number) IS
                             (((INT byte) >> bit.number) /\ 1) <> 0 :
```

```
--
--  pack bytes into blocks
--

PROC pack.bytes.into.blocks(CHAN OF BYTE byte.source,
                            CHAN OF SIGNAL end.of.source,
                            CHAN OF BLOCK block.sink     )
  BOOL more.bytes.expected :
  SEQ
    more.bytes.expected := TRUE
    WHILE more.bytes.expected
      [bytes.in.a.block]BYTE buffer :
      ALT
        byte.source ? buffer[0]
          SEQ
            SEQ byte.number = 1 FOR bytes.in.a.block - 1
              BYTE byte IS buffer[byte.number] :
              ALT
                more.bytes.expected & byte.source ? byte
                  SKIP
                more.bytes.expected & end.of.source ? CASE signal
                  more.bytes.expected := FALSE
                NOT more.bytes.expected & SKIP
                  SKIP
            block.sink ! buffer
        end.of.source ? CASE signal
          more.bytes.expected := FALSE
:

--
--  unpack bytes from blocks
--

PROC unpack.bytes.from.blocks(CHAN OF BLOCK block.source,
                              CHAN OF SIGNAL end.of.source,
                              CHAN OF BYTE byte.sink       )
  BOOL more.blocks.expected :
  SEQ
    more.blocks.expected := TRUE
    WHILE more.blocks.expected
      ALT
        [bytes.in.a.block]BYTE buffer :
        block.source ? buffer
          SEQ byte.number = 0 FOR bytes.in.a.block
            byte.sink ! buffer[byte.number]
        end.of.source ? CASE signal
          more.blocks.expected := FALSE
:
```

```
--
--  bit packing routines
--

PROC pack.bits.into.bytes(CHAN OF BIT bit.source,
                          CHAN OF SIGNAL end.of.source,
                          CHAN OF BYTE byte.sink       )
  BOOL more.bits.expected :
  SEQ
    more.bits.expected := TRUE
    WHILE more.bits.expected
      BYTE byte :
      ALT
        BOOL bit :
        bit.source ? bit
          SEQ
            set.bit(byte, 0, bit)
            SEQ bit.number = 1 FOR bits.in.a.byte - 1
              ALT
                more.bits.expected & bit.source ? bit
                  set.bit(byte, bit.number, bit)
                more.bits.expected & end.of.source ? CASE signal
                  more.bits.expected := FALSE
                NOT more.bits.expected & SKIP
                  SKIP
            byte.sink ! byte
        end.of.source ? CASE signal
          more.bits.expected := FALSE
:

PROC pack.bits.into.blocks(CHAN OF BIT bit.source,
                           CHAN OF SIGNAL end.of.source,
                           CHAN OF BLOCK block.sink     )
  CHAN OF BYTE bytes :
  CHAN OF SIGNAL end.of.bytes :
  PAR
    SEQ
      pack.bits.into.bytes(bit.source, end.of.source, bytes)
      end.of.bytes ! signal
    pack.bytes.into.blocks(bytes, end.of.bytes, block.sink)
:
```

```
--
--  bit unpacking routines
--

PROC unpack.bits.from.bytes(CHAN OF BYTE byte.source,
                            CHAN OF SIGNAL end.of.source,
                            CHAN OF BIT bit.sink         )
  BOOL more.bytes.expected :
  SEQ
    more.bytes.expected := TRUE
    WHILE more.bytes.expected
      ALT
        BYTE byte :
        byte.source ? byte
          SEQ bit.number = 0 FOR bits.in.a.byte
            bit.sink ! get.bit(byte, bit.number)
        end.of.source ? CASE signal
          more.bytes.expected := FALSE
:

PROC unpack.bits.from.blocks(CHAN OF BLOCK block.source,
                             CHAN OF SIGNAL end.of.source,
                             CHAN OF BIT bit.sink         )
  CHAN OF BYTE bytes :
  CHAN OF SIGNAL end.of.bytes :
  PAR
    SEQ
      unpack.bytes.from.blocks(block.source, end.of.source, bytes)
      end.of.bytes ! signal
    unpack.bits.from.bytes(bytes, end.of.bytes, bit.sink)
:
```