```
// Inmos Limited; Confidential

// All the declarations for s42rev.vbc
WITH

FIELD Microword[13:10,8]                    // X bus source select
   XfromA    =  #B00001
   XfromC    =  #B0001Ø
   XfromW    =  #B001ØØ
   XfromPW   =  #B01ØØØ
   XfromI    =  #B1ØØØØ ;

FIELD Microword[16,14,9]                    // Y bus source select
   YfromB          =  #BØØ1
   YfromOPD        =  #BØ1Ø
   YfromMinus15    =  #B1ØØ ;

FIELD Microword[48]                         // X and Y bus connection
   XmergeY  =  #B1 ;
SET
   XfromY   =  XmergeY
   YfromX   =  XmergeY  ;

FIELD Microword[28:26]                      // Carryin multiplexor
   NotCarryfrom1                =  #BØØØ
   NotCarryfromNotAslave15      =  #BØØ1
   NotCarryfromCslaveØ          =  #BØ1Ø
   NotCarryfromNotCslaveØ       =  #BØ11
   NotCarryfromØ                =  #B1ØØ ;
SET
   NoCarry                 = NotCarryfrom1
   NoBorrow                = NotCarryfromØ
   Carry                   = NotCarryfromØ
   Borrow                  = NotCarryfrom1
   CarryfromAslave15       = NotCarryfromNotAslave15
   CarryfromCslaveØ        = NotCarryfromNotCslaveØ
   CarryfromNotCslaveØ     = NotCarryfromCslaveØ ;

FIELD Microword[60:55]                      // Alu operations
   ZfromXplusY       =  #B1ØØ1Ø1     // X + Y + Carry
   ZfromXminusY      =  #B11Ø1Ø1     // X - Y - Borrow
   ZfromYminusX      =  #B1Ø1Ø1Ø     // Y - X - Borrow
   ZfromOnes         =  #BØ11111     // -1
   ZfromNotCarry     =  #B1ØØØØØ     // Ø - Carry
   ZfromZero         =  #BØØØØØØ     // Ø
   ZfromX            =  #BØØØØ11     // X
   ZfromY            =  #BØØ1ØØ1     // Y
   ZfromXxorY        =  #BØØ1Ø1Ø     // X >< Y
   ZfromXxnorY       =  #BØØØ1Ø1     // ~(X >< Y)
   ZfromXorY         =  #BØØ1Ø11     // X \/ Y
   ZfromXandY        =  #BØØØØØ1     // X /\ Y
   ZfromNotX         =  #BØØ11ØØ     // ~X
   ZfromNotY         =  #BØØØ11Ø     // ~Y
   ZfromMinusY       =  #B11Ø11Ø     // (-Y) - Borrow
   ZfromYminus       =  #B1Ø1ØØ1     // Y - Borrow
   ZfromYplus        =  #B1ØØ11Ø     // Y + Carry      ;

FIELD Microword[6:Ø]                        // Z bus destination select
   AfromZ    =  #BØØØØØØ1
   BfromZ    =  #BØØØØØ1Ø
   CfromZ    =  #BØØØØ1ØØ
   WfromZ    =  #BØØØ1ØØØ
```

```
    PWfromZ   =  #B00100000
    IfromZ    =  #B01000000
    OPDfromZ  =  #B10000000 ;

FIELD Microword[49,44]                  // OPD register control
    OPDNot0   =  #B01
    IncOPD    =  #B10 ;

FIELD Microword[53:51,43:41]            // Shifting control
    AShiftin0from1                =  #B000001
    CShiftin15fromCarryout        =  #B00001Ø
    CShiftin15fromArithLogic      =  #B000100
    ShiftLeftCA                   =  #B001000
    ShiftRightC                   =  #B010000
    ShiftRightA                   =  #B100000 ;
SET
    AShiftin0from0                =  #B000000
    CShiftin15from0               =  #B000000 ;

FIELD Microword[18:17]                  // A/B stack control
    AfromB    =  #B01
    BfromA    =  #B10 ;

FIELD Microword[25:21]                  // A/B setting multiplexor
    AsetfromZeq0                =  #B00001
    AsetfromGreaterThan         =  #B01101
    BsetfromZNoteq0             =  #B00110
    BsetfromOverflow            =  #B01010
    Bsetfrom1                   =  #B10010
    BsetfromQuotSign            =  #B10110
    Bsetfrom0                   =  #B11010 ;

FIELD Microword[47:46]                  // C setting from Carryout
    CsetfromNotCarryout         =  #B10
    CsetfromCarryout            =  #B11 ;

FIELD Microword[31:29]                  // destination phase disable
    Disablefrom0                =  #B000
    DisablefromZlt256           =  #B001
    DisablefromNotDivdSign      =  #B010
    DisablefromNotQuotSign      =  #B011
    DisablefromNotBslave15      =  #B100
    DisablefromNotCslave15      =  #B101 ;

FIELD Microword[40:38]                  // sign control for DIV
    SignDivdfromC15             =  #B001
    SignQuotfromZ15             =  #B010
    DoingDiv                    =  #B100 ;

FIELD Microword[34:32]                  // condition select for MIR[1]
    Cond1fromROMFeedback1       =  #B000
    Cond1fromAslave0            =  #B001
    Cond1fromCarryout           =  #B010
    Cond1fromMulStep1           =  #B011
    Cond1from1                  =  #B100
    Cond1fromDivStep0           =  #B101
    Cond1fromDivStep1           =  #B110
    Cond1fromAslave1            =  #B111 ;

FIELD Microword[37:35]                  // condition select for MIR[0]
    Cond0fromROMFeedback0       =  #B000
    Cond0fromDoingDiv           =  #B001
    Cond0fromZeq0               =  #B010
```

```
    Cond0fromMulStep0       =  #B011
    Cond0fromOPDeq0         =  #B100 ;

FIELD Microword[67:61]                  // loads into MIR
    ROMFeedback[0]          =  #B0000001
    ROMFeedback[1]          =  #B0000010
    ROMFeedback[2]          =  #B0000100
    ROMFeedback[3]          =  #B0001000
    ROMFeedback[4]          =  #B0010000
    ROMFeedback[5]          =  #B0100000
    ROMFeedback[6]          =  #B1000000 ;

FIELD Microword[54,50,45,20:19,15,7]   // Interface Control Requests
    MADDRfromZ              =  #B0000001
    DATAINfromZ             =  #B0000010
    Write                   =  #B0000100
    Byte                    =  #B0001000
    ForceFetch              =  #B0010000
    Next                    =  #B0100000
    RunningfromZNoteq0      =  #B1000000 ;
SET
    Read = 0 ;
```

```
// Inmos Limited; Confidential

// a typical piece of violence from the mind of VBCman :-
// howl, gneek, fring, splunshuleecreak
// REVISION TO VBC LEVEL 7, PATCH CODE 42.69

USE "[miles.simple.docls42with.vbc"

DO

      // B:=A; A:=(W!OPD)
LDW:
         XfromW          YfromOPD        NoCarry
         ZfromXplusY
         AtoB
         MADDRfromZ      Read
         LDWread ;

LDWread:
         XfromDATAIN                     NoCarry
         ZfromX
         AfromZ
         Next ;


      // W!OPD:=A; A:=B
STW:
         XfromW          YfromOPD        NoCarry
         ZfromXplusY
         MADDRfromZ      Write
         STWwrite ;

STWwrite:
         XfromA
         AfromB
         Next ;


      // B:=A; A:=W+OPD
LDPW:
         XfromW          YfromOPD        NoCarry
         ZfromXplusY
         AfromZ
         BfromA
         Next ;


      // B:=A; A:=W!OPD; W!OPD:=A+1
      // Actually
      // B := A
      // MADDR := W+OPD
      // Then the rest is as LDTI
LDWI:
         XfromW          YfromOPD        NoCarry
         ZfromXplusY
         MADDRfromZ      Read
         BfromA
         LDTIread ;


      // A:=A!OPD
LDT:
```

```
            XfromA          YfromOPD        NoCarry
            ZfromXplusY
            MADDRfromZ      Read
            LDWread ;


        // A!OPD:=B
STT:
            XfromA          YfromOPD        NoCarry
            ZfromXplusY
            MADDRfromZ      Write
            STTwrite ;

STTwrite:
            XfromY          YfromB
            Next ;


        // A:=A+OPD
LDPT:
            XfromA          YfromOPD        NoCarry
            ZfromXplusY
            AfromZ
            Next ;


        // MADDR:=A+OPD; A:=A!OPD; !MADDR:=!MADDR+1
LDTI:
            XfromA          YfromOPD        NoCarry
            ZfromXplusY
            MADDRfromZ      Read
            LDTIread ;

        // incremented A!OPD stored and saved in A
LDTIread:
            XfromDATAIN     YfromX          Carry
            ZfromPlusY
            AfromZ          Write
            LDTIwrite ;

        // now decrement A
LDTIwrite:
            XfromA          YfromX          Borrow
            ZfromMinusY
            AfromZ
            Next ;


        // I:=I+OPD
J:
            XfromI          YfromOPD        NoCarry
            ZfromXplusY
            IfromZ          Read            ForceFetch
            UMULend ;


        // IF A\=Ø THEN I:=I+OPD ; A:=B
JNZ:
            XfromA                          NoCarry
            ZfromX
            AfromB
            CondØfromZeqØ (JNZend, JNZdojump) ;
```

```
JNZdojump:
        XfromI          YfromOPD        NoCarry
        ZfromXplusY
        IfromZ          Read            ForceFetch
        JNZend ;

JNZend:
        Next ;


    // B:=A;A:=OPD
LDL:
                        YfromOPD        NoCarry
        ZfromY
        AfromZ
        BfromA
        Next ;


    // B:=A; A:=I+OPD
LDPI:
        XfromI          YfromOPD        NoCarry
        ZfromXplusY
        AfromZ
        BfromA
        Next ;


    // OPD := OPD<<4
    // NOTE OPDNotØ also loads OPD[15:4] from Z[11:Ø]
PFIX:
                        YfromOPD        NoCarry
        ZfromY
        OPDNotØ
        Next ;


    // OPD := (NOT OPD)<<4
    // NOTE OPDNotØ also loads OPD[15:4] from Z[11:Ø]
    // 'NOT' allows one more negative number per nibble
    // since -Ø = Ø but NOT Ø = -1.
NFIX:
                        YfromOPD        NoCarry
        ZfromNotY
        OPDNotØ
        Next ;



//_____________________ opr1 operations _____________________\\


    // LET t = A!Ø
    //
    // TEST t = Ø
    // THEN
    //    A ! Ø := W
    //    GOTO WAIT
    // ELSE
```

```
      //    A ! Ø := Ø
      //    A := t
      //    GOTO RUN
SYNC:
         XfromA                              NoCarry
         ZfromX
         MADDRfromZ        Read
         SYNCread ;

SYNCread:
         XfromDATAIN                         NoCarry
         ZfromX
         Write
         CondØfromZeqØ (SYNCwritew, SYNCwriteopd) ;

SYNCwritew:
         XfromW
         WAIT ;

      // assumes that OPD holds Ø (operand nibble of instruction)
SYNCwriteopd:
         XfromY            YfromOPD          NoCarry
         SYNCsavet ;

SYNCsavet:
         XfromDATAIN                         NoCarry
         ZfromX
         AfromZ
         RUN ;


      // IF A = Ø
      // THEN A:= #XFFFF
      // ELSE A:= #XØØØØ
EQZ:
         XfromA                              NoCarry
         ZfromX
         AsetfromZeqØ
         Next ;


      // A,C := B>A(signed), B>A(unsigned)
      // IF B > A
      // THEN A:= #XFFFF
      // ELSE A:= #XØØØØ
      // A - B < Ø if B > A; (all unsigned) therefore a borrow is generated
      // for signed A>B the alu greater.than output is used
GT:
         XfromA            YfromB            NoBorrow
         ZfromXminusY
         AsetfromGreaterThan
         CsetfromNotCarryout
         Next ;


      // A := B /\ A
AND:
         XfromA            YfromB            NoCarry
         ZfromXandY
         AfromZ
         Next ;
```

```
      // A := B \/ A
OR:
         XfromA          YfromB          NoCarry
         ZfromXorY
         AfromZ
         Next ;

      // A:= A >< B
XOR:
         XfromA          YfromB          NoCarry
         ZfromXxorY
         AfromZ
         Next ;

      // C := Ø
CLC:
                                         NoCarry
         ZfromZero
         AfromZ
         Next ;

      // B:=A; A:=C
LDAC:
         XfromC                          NoCarry
         ZfromX
         AfromZ
         BfromA
         Next ;

      // C:=A; A:=B
STAC:
         XfromA                          NoCarry
         ZfromX
         CfromZ
         AfromB
         Next ;

      // A:=(A+B)[15:Ø]
ADD:
         XfromA          YfromB          NoCarry
         ZfromXplusY
         AfromZ
         Next ;

      // A := B + A + C[Ø]
      // B := overflow
      // C := carry
ADDC:
         XfromA          YfromB          CarryfromCslaveØ
         ZfromXplusY
         AfromZ
         BsetfromOverflow
         CsetfromCarryout
         Next ;

      // A := B-A
```

```
SUB:
        XfromA          YfromB          NoBorrow
        ZfromYminusX
        AfromZ
        Next ;

      // A := B - A - C[Ø]
      // B := overflow
      // C := borrow
SUBC:
        XfromA          YfromB          CarryfromNotCslaveØ
        ZfromYminusX
        AfromZ
        BsetfromOverflow
        CsetfromNotCarryout
        Next ;

      // C.A := B * A (signed)
      // B   := overflow
      //
      //      LET extended.bit = Ø
      //      LET multiplier.bit = A[Ø]
      //      multiplier = A;     multiplicand = B;     accumulator = C
      //
      //      FOR i = 1 TO 16 DO
      //      $(
      //          SWITCHON (multiplier.bit).(extended.bit) INTO
      //          $(
      //              CASE ØØ : no change to accumulator
      //              ENDCASE
      //              CASE Ø1 : accumulator := accumulator + multiplicand
      //              ENDCASE
      //              CASE 1Ø : accumulator := accumulator - multiplicand
      //              ENDCASE
      //              CASE 11 : no change to accumulator
      //              ENDCASE
      //          $)
      //      (add) / NOT (subtract) = NOT (multiplier.bit) / multiplier.bit
      //      write.result           = multiplier.bit (+) extended.bit
      //      only write result of C +/- B to C if xor true
      //
      //          multiplier.bit:= A[1]
      //          extended.bit := A[Ø]
      //
      //          alu[15:Ø] := C[15:Ø]
      //          A[14:Ø]   := A[15:1]
      //          A[15]     := alu[Ø] (i.e. C[Ø])
      //          C[14:Ø]   := alu[15:1]
      //          C[15]     := overflow -> Ø,alu[15]
      //              Overflow occurs on Ø - 2*(most neg),
      //              and NOT alu[16]=Ø for this case; only possible
      //              in the first iteration.
      //              If overflow occurs, then do a logical
      //              SR (C[15] := Ø). If no overflow, then do an
      //              arithmetical SR (C[15] := alu[15]).
      //              So we set the bit which is shifted into C[15]
      //              (actually in the alu so alu[15]) /\ NOT overflow
      //      $)
      //
      //      Use TLNG to check whether the contents of C are only
      //      a sign extension of A; if not, B := -1, meaning that
```

```
//      the product is a double length number (overflows
//      single length).
//
//       B := NOT (( C + A[15] ) = Ø )   // testlong
//
//      Control logic:
//        The following signals are input to the Multiply
//        step control logic which produces MIR[1] and
//        MIR[Ø] from them.
//        A[1]        // These are BEFORE the shift right i.e. from
//        A[Ø]        // the master half of A in the source phase
//        OPD=Ø
//
//      The codes generated by the multiply step control are:
//
//                  A[Ø]
//      OPD~=Ø |   Ø    1          OPD = Ø   i.e. Force ØØ in all cases.
//             -----------
//      A[1] Ø | Ø1   1Ø          ØØ   ØØ
//           1 | 11   Ø1          ØØ   ØØ
//
//      That is
//                  A[Ø]
//                | Ø   1
//               --------
//      A[1] Ø | .   +
//           1 | -   .
//
//      The first cycle of all is special, as there is no
//      shift right possible and the only two cases are
//        1) do nothing
//        2) subtract
//      so we MUST ensure that the codes for do nothing and
//      subtract differ in only one bit.

    // clear C, as Carryout is set
MUL:
                        YfromMinus15    NoCarry
        ZfromY
        OPDfromZ
        CsetfromNotCarryout
        Cond1fromAslaveØ (MULshiftsubtract, MULshift) ;

MULshiftadd:
        IncOPD
        ShiftRightA
        XfromC          YfromB          NoCarry
        ZfromXplusY
        ShiftRightC
        Cshiftin15fromArithLogic
        Cond1fromMulStep1               CondØfromMulstepØ
           (MULshiftsubtract, MULshiftadd, MULshift, MULend) ;

MULshiftsubtract:
        IncOPD
        ShiftRightA
        XfromC          YfromB          NoBorrow
        ZfromXminusY
        ShiftRightC
        Cshiftin15fromArithLogic
        Cond1fromMulStep1               CondØfromMulstepØ
           (MULshiftsubtract, MULshiftadd, MULshift, MULend) ;
```

```
MULshift:
        IncOPD
        ShiftRightA
        XfromC                          NoCarry
        ZfromX
        ShiftRightC
        Cshiftin15fromArithLogic
        Cond1fromMulStep1               CondØfromMulstepØ
           (MULshiftsubtract, MULshiftadd, MULshift, MULend) ;

MULend:
      // this is a copy of tlng, checking to find if C is
      // a sign extension of A
        XfromC          YfromX          CarryfromAslave15
        ZfromPlusY
        BsetfromZNoteqØ
        Next ;


      // C.A := B * A + C (unsigned)
      // B   := B                              //must preserve it
      //      LET multiplier.bit = A[Ø]
      //
      //      FOR i = 1 TO 16 DO
      //      $(
      //
      //         SWITCHON (multiplier.bit)     INTO
      //         $(
      //            CASE Ø : no change to accumulator
      //            ENDCASE
      //            CASE 1 : accumulator := accumulator + multiplicand
      //            ENDCASE
      //         $)
      //
      //         A[14:Ø] := A[15:1]
      //         A[15]   := alu[Ø]
      //         C[14:Ø] := alu[15:1]
      //         C[15]   := carry.out
      //             carry.out is the 17th bit of the
      //             unsigned addition, which is shifted
      //             right into C.A
      //
      //         multiplier.bit:= A[Ø]   // AFTER shift right
      //      $)
      //
UMUL:
                        YfromMinus15    NoCarry
        ZfromY
        OPDfromZ
        Cond1fromAslaveØ (UMULshiftadd, UMULshift) ;

UMULshiftadd:
        IncOPD
        ShiftRightA
        XfromC          YfromB          NoCarry
        ZfromXplusY
        ShiftRightC
        Cshiftin15fromCarryout
        Cond1fromAslave1                CondØfromOPDeqØ
           (UMULend1, UMULshiftadd, UMULend, UMULshift) ;

UMULshift:
        IncOPD
```

```
        ShiftRightA
        XfromC                          NoCarry
        ZfromX
        ShiftRightC
        Cshiftin15fromØ
        Cond1fromAslave1                CondØfromOPDeqØ
           (UMULend1, UMULshiftadd, UMULend, UMULshift) ;

UMULend:
        Next ;

UMULend1:
        Next ;


      // A := (C.A)/B
      // B := overflow((C.A)/B)
      // C := (C.A) rem B
      // done by non-restoring divide on unsigned
      // numbers to share microcode
      //
      // definition:   dividend = quotient * divisor  +  remainder
      // remainder has sign of divisor, unless it can be reduced to Ø
      //
      //   1. remove the signs of dividend and divisor and store them
      //
      //   2. enter UDIV code and check for overflow or divide by Ø
      //
      //   3. perform iteration to obtain partial remainder 16 times
      //
      //   4. end correct result in case of -ve partial remainder
      //
      //   5. exit if UDIV, or restore signs to quotient and remainder :
      //
      //      dividend     divisor    quotient     remainder
      //      +            +          +            +
      //      +            -          -            +
      //      -            +          -            -
      //      -            -          +            -
      //
      //   1.
      //      sign.dividend   := C[15]
      //      IF sign.dividend = 1
      //      THEN
      //      $(
      //          A           :=   Ø - A
      //          carry.save := carry.out
      //
      //          carry.in    := NOT (carry.save)
      //          C           := Ø - C
      //      $)
      //      sign.divisor := B[15]
      //      IF sign.divisor = 1
      //      THEN
      //      $(
      //         B := Ø - B
      //      $)
      //      signed.division := 1
      //
      //   2.
      //      UNLESS  C < B
      //      DO
      //      $(
```

```
//          B := -1       /overflow true, break
//      $)
//      x := Ø
//
//  3.
//      FOR i = 1 TO 16 DO
//      $(
//          bit    := NOT (x)
//          x.C.A := C.A << 1
//          A[Ø]    := bit
//
//          TEST bit = 1
//          THEN
//             x.C := x.C - B    /this x is C[15] before shift left!
//          ELSE
//             x.C := x.C + B    /this x is C[15] before shift left!
//      $)
//
//  4.
//      TEST x.C < Ø
//      THEN
//      $(
//         C := C + B
//         A := A << 1   + Ø
//      $)
//      ELSE
//      $(
//         A := A << 1   + 1
//      $)
//
//  5.
//      IF signed.divide = FALSE
//      THEN
//      $(
//         B := Ø                / no overflow
//         RETURN
//      $)
//
//      IF sign.dividend = 1
//      THEN
//      $(
//         C := Ø - C
//      $)
//      IF sign.dividend \= sign.divisor
//      THEN
//      $(
//         A := Ø - A
//         IF A = Ø
//         THEN
//         $(
//            B := Ø             / no overflow
//            RETURN
//         $)
//      $)
//
//      Check A has right sign, unless quotient = Ø
//
//      IF sign.dividend \= sign.divisor \= A[15]  /\
//         A \= Ø
//      THEN
//      $(
//         B := -1               / overflow true
//      $)
```

```
      //  $)
      //
DIV:
        XfromC          YfromB          NoCarry
        ZfromXxorY
        SignDivdfromC15
        SignQuotfromZ15
        DoingDiv
        DIVunsignA ;

DIVunsignA:
        XfromA          YfromX          NoBorrow
        ZfromMinusY
        AfromZ                          DisablefromNotCslave15
        Cond1fromCarryout (DIVunsignC1, DIVunsignC2)  ;

DIVunsignC1:
        XfromC          YfromX          Carry
        ZfromMinusY
        CfromZ                          DisablefromNotCslave15
        DIVunsignB ;

DIVunsignC2:
        XfromC          YfromX          NoCarry
        ZfromMinusY
        CfromZ                          DisablefromNotCslave15
        DIVunsignB ;

DIVunsignB:
                        YfromB          NoBorrow
        BfromZ                          DisablefromNotBslave15
        UDIV ;



//_____________________________ opr2 operations _____________________________\\


      // A := (C.A)/B              (unsigned)
      // B := overflow ((C.A)/B)
      // C := (C.A) rem B          (unsigned)
      //
      // test divide overflow; overflow if C - B >= Ø
UDIV:
        XfromC          YfromB          NoBorrow
        ZfromXminusY
        Cond1fromCarryout (UDIVoverflow, UDIVinitloop)  ;

UDIVoverflow:
        Bsetfrom1
        Next  ;

UDIVinitloop:
        YfromMinus15                    NoCarry
        ZfromY
        OPDfromZ
        UDIVloopsub ;

UDIVloopsub:
        IncOPD
        AshiftinØfrom1
```

```
        ShiftLeftCA         YfromB      NoBorrow
        ZfromXminusY
        CfromZ
        Cond1fromDivStep1               CondØfromOPDeqØ
           (UDIVlastbit1, UDIVloopsub, UDIVlastbitØ, UDIVloopadd)  ;

UDIVloopadd:
        IncOPD
        AshiftinØfromØ
        ShiftLeftCA         YfromB      NoCarry
        ZfromXplusY
        CfromZ
        Cond1fromDivStepØ               CondØfromOPDeqØ
           (UDIVlastbit1, UDIVloopsub, UDIVlastbitØ, UDIVloopadd)  ;

UDIVlastbitØ:
        AshiftinØfromØ
        ShiftLeftCA
        UDIVlastbitØcorrection ;

UDIVlastbit1:
        AshiftinØfrom1
        ShiftLeftCA
        BsetfromQuotSign
        CondØfromDoingDiv (UDIVresignC, UDIVend) ;

UDIVlastbitØcorrection:
        XfromC          YfromB          NoCarry
        ZfromXplusY
        CfromZ
        BsetfromQuotSign
        CondØfromDoingDiv (UDIVresignC, UDIVend) ;

UDIVend:
        BsetfromØ
        Next ;

DIVresignC:
        XfromC          YfromX          NoBorrow
        ZfromMinusY
        CfromZ                          DisablefromNotDivdSign
        DIVresignA ;

      // Set the divide overflow register B unless the quotient
      // is zero, to trap this exceptional case.
DIVresignA:
        XfromA          YfromX          NoBorrow
        ZfromMinusY
        AfromZ
        SetBfromNotZeqØ                 DisablefromNotQuotSign
        DIVend ;

      // Check that B is a sign extension of the corrected quotient;
      // If B is a sign extension, there is no overflow, so B := Ø.
      // If B is not a sign extension, there is an overflow, so B := -1.
      // Therefore use B := (B + A[15]) ~= Ø
DIVend:
                        YfromB          CarryfromAslave15
        ZfromPlusY
        SetBfromNotZeqØ
        Next ;
```

```
      // C := (A<Ø)
SEX:
                                        CarryfromAslave15
         ZfromNotCarry
         CfromZ
         SEXchangec ;


SEXchangec:
         XfromC                         NoCarry
         ZfromNotX
         CfromZ
         Next ;


      // B := NOT ((C=Ø /\ A>=Ø) \/ (C=-1 /\ A<Ø))
      // In fact
      // B := (C + A[15]) \= Ø
      // In fact
      // B := (~C - ~A[15]) \= Ø
TLNG:
         XfromC          YfromX         NoCarry
         ZfromPlusY
         BsetfromZNoteqØ
         Next ;


      // A := B%A
      // shiftright A loads MADDR[-1] (the upperlower latch)
      // with A[Ø]
LB:
         ShiftRightA
         ZfromZero
         LBaddress ;

LBaddress:
         XfromA          YfromB         NoCarry
         ZfromXplusY
         MADDRfromZ      Read           Byte
         LDWread ;


      // B%A := C
      // shiftright A loads MADDR[-1] (the upperlower latch)
      // with A[Ø]
SB:
         ShiftRightA
         ZfromZero
         SBaddress ;

SBaddress:
         XfromA          YfromB         NoCarry
         ZfromXplusY
         MADDRfromZ      Write          Byte
         SBwrite ;

SBwrite:
         XfromC
         Next ;


      // C.A := C.A << B
SL:
```

```
                        YfromB          NoBorrow
        ZfromMinusY
        OPDfromZ
        SLloopinit ;

SLloopinit:
        IncOPD
        CondØfromOPDeqØ (SLend, SLloop) ;

SLloop:
        AshiftinØfromØ
        ShiftLeftCA
                                        NoCarry
        ZfromX
        IncOPD
        CondØfromOPDeqØ (SLend, SLloop) ;

SLend:
        Next ;


    // C.A := C.A >> B
SR:
                        YfromB          NoBorrow
        ZfromMinusY
        OPDfromZ
        SRloopinit ;

SRloopinit:
        IncOPD
        CondØfromOPDeqØ (SRend, SRloop) ;

SRloop:
        ShiftRightA
        XfromC                          NoCarry
        ZfromX
        ShiftRightC
        Cshiftin15fromØ
        IncOPD
        CondØfromOPDeqØ (SRend, SRloop) ;

SRend:
        Next ;


    // I := I + A
JX:
        XfromA                          NoCarry
        ZfromX
        OPDfromZ
        JXforcefetch ;

JXforcefetch:
        XfromI          YfromOPD        NoCarry
        ZfromXplusY
        IfromZ          Read            ForceFetch
        UMULend ;


    // W ! iptr := I
    // W        := W ! link
    // PW! link := W
    // I        := W ! iptr
```

```
      // iptr = 1 ; link = Ø
WAIT:
         XfromW          YfromX          Carry
         ZfromPlusY
         MADDRfromZ      Write
         WAITiptrwrite ;

WAITiptrwrite:
         XfromI
         WAITwlink ;

WAITwlink:
         XfromW                          NoCarry
         ZfromX
         MADDRfromZ      Read
         WAITwlinkread ;

WAITwlinkread:
         XfromDATAIN                     NoCarry
         ZfromX
         WfromZ
         WAITpwlink ;

WAITpwlink:
         XfromPW                         NoCarry
         ZfromX
         MADDRfromZ      Write
         WAITpwlinkwrite ;

WAITpwlinkwrite:
         XfromW
         WAITwiptr ;

WAITwiptr:
         XfromW          YfromX          Carry
         ZfromPlusY
         MADDRfromZ      Read
         WAITforcefetch ;

WAITforcefetch:
         XfromDATAIN                     NoCarry
         ZfromX
         IfromZ          Read            ForceFetch
         UMULend ;


      // PW ! link := A
      // PW        := A
      // A  ! link := W
      // iptr = 1 ; link = Ø
RUN:
         XfromPW                         NoCarry
         ZfromX
         MADDRfromZ      Write
         RUNpwlinkwrite ;

RUNpwlinkwrite:
         XfromA                          NoCarry
         ZfromX
         PWfromZ
         RUNwlink ;

RUNwlink:
```

```
        XfromA                          NoCarry
        ZfromX
        MADDRfromZ      Write
        RUNwlinkwrite ;

RUNwlinkwrite:
        XfromW
        Next ;

      // W ! iptr := I
      // PW       := W
      // W        := W ! link
      // I        := W ! iptr
      // iptr = 1 ; link = Ø
PSE:
        XfromW                          NoCarry
        ZfromX
        PWfromZ
        PSEwiptr ;

PSEwiptr:
        XfromW          YfromX          Carry
        ZfromPlusY
        MADDRfromZ      Write
        PSEwiptrwrite ;

PSEwiptrwrite:
        XfromI
        PSEwlink ;

PSEwlink:
        XfromW                          NoCarry
        ZfromX
        MADDRfromZ      Read
        PSEwlinkread ;

PSEwlinkread:
        XfromDATAIN                     NoCarry
        ZfromX
        WfromZ
        WAITwiptr ;


      // W ! iptr := I
      // PW! link := A
      // A ! link := W!link
      // W,A      := A,W
      // I        := B
      // iptr = 1 ; link = Ø
CALL:
        XfromW                          NoCarry
        ZfromX
        OPDfromZ
        MADDRfromZ      Read
        CALLwlinkread ;

CALLwlinkread:
        XfromA                          NoCarry
        ZfromX
        WfromZ
        CALLwiptr ;
```

```
      // A and W will now be swapped;  and OPD set to W+iptr
CALLwiptr:
                        YfromOPD        NoCarry
          ZfromY
          AfromZ
          IncOPD
          CALLpwlink ;

CALLpwlink:
          XfromPW                       NoCarry
          ZfromX
          MADDRfromZ    Write
          CALLpwlinkwrite ;

CALLpwlinkwrite:
          XfromW        YfromOPD        NoCarry
          ZfromY
          MADDRfromZ    Write
          CALLwiptrwrite ;

CALLwiptrwrite:
          XfromI        YfromB          NoCarry
          ZfromY
          IfromZ
          OPDfromZ
          CALLwlink ;

CALLwlink:
          XfromW                        NoCarry
          ZfromX
          MADDRfromZ    Write
          CALLforcefetch ;

CALLforcefetch:
          XfromDATAIN   YfromOPD        NoCarry
          ZfromY
          Read          ForceFetch
          UMULend ;



      // B := A ! iptr
      // goto CALL
RET:
          XfromA        YfromX          Carry
          ZfromPlusY
          MADDRfromZ    Read
          RETreadiptr ;

RETreadiptr:
          XfromDATAIN                   NoCarry
          ZfromX
          BfromZ
          CALL ;

      // A,B := B,A
REV:
          BfromA
          AfromB
          Next ;


      // running := (A \= Ø)
```

```
      // W := A
      // PW, A := A, PW
      // I := B
SWITCH:
         XfromPW                        NoCarry
         ZfromX
         OPDfromZ
         SWITCHrun ;

SWITCHrun:
         XfromA                         NoCarry
         ZfromX
         WfromZ
         PWfromZ
         RunningfromZNoteqØ
         SWITCHafrompw ;

SWITCHafrompw:
                        YfromOPD        NoCarry
         ZfromY
         AfromZ
         SWITCHforcefetch ;

SWITCHforcefetch:
                        YfromX          NoCarry
         ZfromY
         IfromZ
         Read           ForceFetch
         UMULend ;


      // DO C TIMES
      //  $(
      //     !A := !B
      //     UNLESS iochan(A) DO A := A+1
      //     UNLESS iochan(B) DO B := B+1
      //  $)
      // Actually we have the following
      //
      // B := B-1
      // OPD := A
      // $(
      //    maddr := B+1
      //    READ
      //    UNLESS iochan(B)
      //       B := B+1
      //
      //    C := C-1
      //    IF C=Ø THEN BREAK
      //
      //    maddr := OPD
      //    WRITE
      //    mdata := datain
      //
      //    UNLESS iochan(OPD)
      //       OPD := OPD+1
      //
      //  $) REPEAT
      //
      //  B := B+1
      //
      // If C = Ø, still do one read !
      //
```

```
MOVE:
        XfromA                          NoCarry
        ZfromX
        OPDfromZ
        MOVEbinit;

MOVEbinit:
                        YfromB          Borrow
        ZfromMinusY
        BfromZ
        MOVEloop ;

      // Disabling the destination stops the
      // incremented value from being saved.
      // MADDR is loaded with incremented value,
      // and READ requested by Interface Controller
      // even if the destination is disabled.
MOVEloop:
        XfromDATAIN     YfromB          Carry
        ZfromXplusY
        BfromZ                          DisablefromZlt256
        MADDRfromZ
        Read
        MOVEdeccount ;

MOVEdeccount:
        XfromC          YfromX          Borrow
        ZfromMinusY
        CfromZ
        CondlfromCarryout (MOVEincopd, MOVEend) ;

      // Disabling the destination stops the
      // incremented value of OPD from being saved.
      // MADDR is loaded with incremented value,
      // and WRITE requested by Interface Controller
      // even if the destination is disabled.
MOVEincopd:
                        YfromOPD        NoCarry
        ZfromY
        IncOPD                          DisablefromZlt256
        MADDRfromZ
        Write
        MOVEloop ;

MOVEend:
                        YfromB          Carry
        ZfromPlusY
        BfromZ
        Next ;


      // power-up and reset state
      // ForceFetch clears MADDR by loading it from
      // the Z bus, which is set to Ø.
START:
                                        NoCarry
        ZfromZero
        Read            ForceFetch
        UMULend ;
```