C to VHDL Converter in a Codesign Environment

Matthew F. Parkinson, Paul M. Taylor and Sri Parameswaran

Department of Electrical and Computer Engineering
University of Queensland, St. Lucia 4072
Brisbane, Queensland, Australia
email : parkinso@s1.elec.uq.oz.au

Abstract

Automation of the Hardware/Software Codesign methodology brings with it the need to develop sophisticated high-level synthesis tools. This paper presents a tool which is the result of such development. This tool converts standard C code into an equivalent VHDL behavioural description. This description is used to generate a chip-level hardware interconnect of identical functionality to the original C code.

1 Introduction

Automated design methodologies in digital systems have until recently been limited entirely to the design of hardware. Automated Hardware/Software Codesign (HSC) offers a design methodology for a total system (ie. both hardware and software).

For a totally hardware oriented design (eg. ASICs) the development time is prohibitive in bringing fresh and affordable products to the market. Equally restrictive is a totally software based solution which will perform slowly due to the use of a generalised computing architecture (ie. a RISC based microprocessor). This is where designing for a hybrid between a hardware and software based implementation can be of particular advantage.

A codesign methodology enables the specification of an algorithm totally in software. Through an automated design process the algorithm is optimally partitioned into both hardware and software, thus allowing the designer to be distanced from the hardware specific techniques of improving an algorithm's performance. This in turn allows the designer to concentrate on the algorithm's design.

Algorithm bottlenecks are usually limited to a small portion of the actual code. By converting these critical code segments into hardware, an ideal partitioning of the algorithm's execution into both hardware and software is achieved. An overview of the automated Hardware/Software Codesign methodology is briefly outlined in Figure 1. This automated partitioning process initially identifies the critical code segments within the software. These code segments are then used to provide a near optimal partition between hardware and software implementations. By next applying the process of synthesis to the partitioned code it is possible to achieve significant acceleration of the algorithm.

The automation of this partitioning process also permits the design to be independent of the final hardware required for execution. The actual hardware implementation is determined through cost and resource constraints. This easily allows the designer to take advantage of emerging technologies without the requisite redesign of the system from the ground up. An example of a proposed hybrid architecture is outlined in Figure 2.

Acceleration of the algorithm is achieved by converting the high-level language description of the hardware partition into VHDL [2] code. By then passing the VHDL code through the package SYNT [3], we have both a hardware description of the partition, as well as feedback on the cost in terms of chip area, and execution time. This cost is related to implementing the algorithm on a XILINX FPGA.

As our system is specified in C, a tool was constructed to turn sections of C code into behavioural VHDL [2]. From VHDL we construct hardware using the behavioural level synthesis tool SYNT [3]. The tool described here is the C to VHDL converter.

Section 3 details the C to VHDL synthesis tool, while Section 4 details a large example.

For a complete description of the HSC process as it is applied above, please refer to [1] and [4].

2 C to VHDL converter

The process of converting C code to VHDL isn't restricted to the area of Hardware/Software Codesign. As a general synthesis tool, C2VHDL provides the means of transferring C algorithms to FPGA interconnect. For custom-chip solutions it provides a good front-end tool to the SYNT package.

0-8186-6215-8/94 $03.00

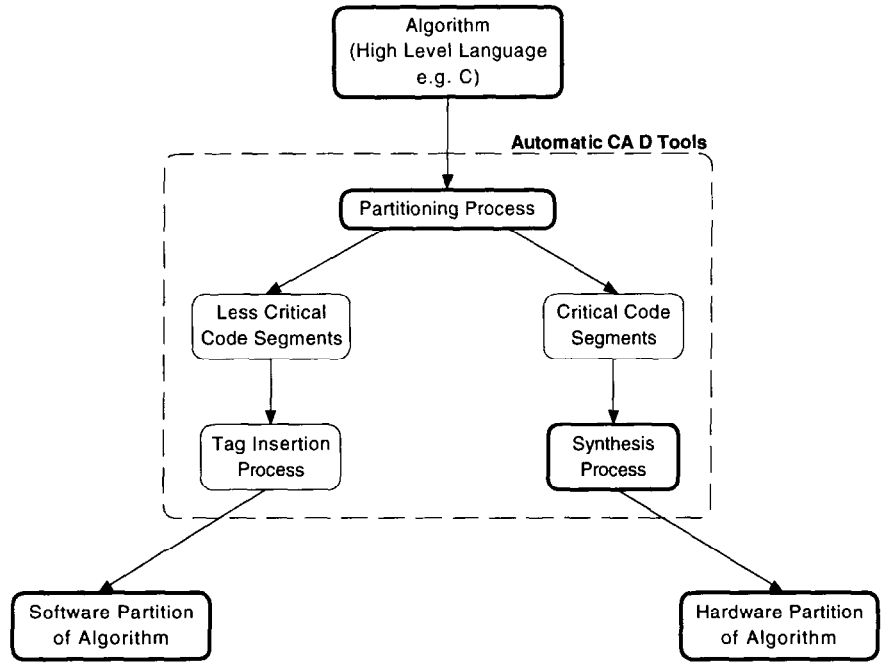

Figure 1 - Automated Hardware/Software Codesign Methodology

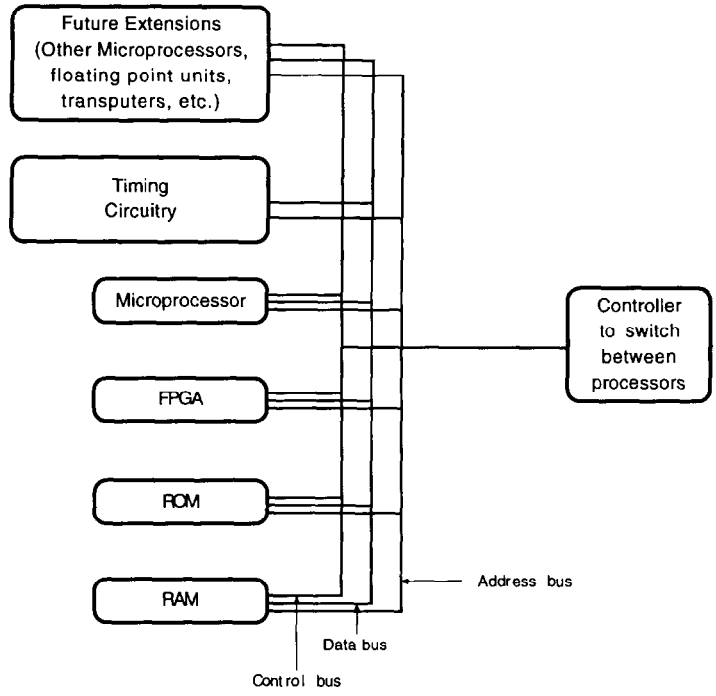

Figure 2 - Hybrid Architecture Configuration

The consistent use of a familiar language such as C means design time for products is greatly reduced. In turn, time to market for profit oriented products is minimised. By employing testability and simulation in the design process, off-the-shelf algorithms may be used in future products. It is important to reduce design effort through the reuse of standard algorithms.

The ability to perform automated design of hardware is essential to the HSC process. The code is initially described using the C language. VHDL is derived from the language ADA. Consequently it is similar to C as far as standard high-level constructs are concerned. The process we use in C2VHDL is based around this fact. The design of the system is based on GNU CC [5], and its GCC compiler (version 2.4.3.1). The GCC code is centred around a YACC [6] description of the C language.

The YACC parser (c-parse.y) is composed of productions. When these productions are triggered, the C code associated with them is executed. By strategically placing C code on the productions which parse variable declarations, the identifiers and their types are recorded. Once the declaration section is finished this information is written to the VHDL output file, with the appropriate change in style to suit the VHDL code. The same procedure is applied to all of the C constructs to produce a complete VHDL description of the original C code.

The VHDL code produced from this parsing process is consequently passed through the synthesis package SYNT. The form of VHDL implemented by SYNT is called SynVHDL. It is a proper subset of VHDL, in that there have been no additions made to the VHDL language. It does however fail to embrace all of the constructs found in standard VHDL. This in turn limits the ability for C2VHDL to convert all constructs found in the C language successfully to VHDL. As SynVHDL improves on its subset of VHDL, so too will C2VHDL improve on its ability to support the full constructs of the C language. The limitations aren't seen to be of great significance however, with most major C constructs being supported.

The structure of the code produced by C2VHDL is influenced by the nature of the HSC process, as well as the limitations imposed by SYNT, in the form of SynVHDL. A section of C code is converted to VHDL with a single ENTITY ... PORT description, and a single ARCHITECTURE body. SYNT can only synthesize for a single VHDL process, and requires a totally behavioural VHDL description. The ARCHITECTURE is therefore composed in a behavioural fashion, consisting of only one process.

The PORT description details the transfer of information across the hardware/software boundary. In the case of HSC, the software partition must transfer data across this PORT interface. The current implementation of SYNT doesn't allow the use of PROCEDURES or FUNCTIONS in VHDL. This will not present a great problem as far as the HSC process is concerned. Its interest is in converting compound code sections to hardware. As chip area is at a premium, it isn't necessary to convert larger sections of code. This is in keeping with the concept that critical bottlenecks will consist of several compound code sections.

The conversion to VHDL of assignment constructs involving various combinations of operators is performed by firstly breaking the statement into the individual calculations. This is achieved by creating temporary variables for each term in the calculation. This not only maintains the precedence of operators, but also prevents calculations being placed in the conditional clause of IF ... THEN statements and the like.

The size of data types and variables is maintained throughout the conversion. By using pointers when passing information across the chip boundary, the data transfer interface is minimised. This is also reflected in chip-area utilisation. Usually, the smaller the data interface, the fewer the on-chip registers that are required.

The conversion to VHDL of conditional statements is reasonably straight-forward. The use of temporary variables can be seen in the example of Figure 3.

```
if (g > 3) {                    as_10 := g;
        g = 4;                  as_11 := 3;
        p = 6;                  IF as_10 > as_11 THEN
} else {                        g := 4;
        g = 5;                          p <= 6;
        p = 2;                  ELSE
}                                       g := 5;
                                        p <= 2;
                                END IF;
```

Figure 3 - IF .. ELSE conversion

The temporary variables are optimized out in the synthesis package SYNT.

Another example is the *switch* on *case* statements which are converted to CASE on WHEN statements. Each *case* must end in a *break* at this stage. This is to prevent flow on to the next *case* statement, as this isn't directly supported in VHDL. The *default* is converted to a WHEN OTHERS construct. It isn't necessary to include the *default* statement, as WHEN OTHERS will be included regardless. It isn't necessary to have any statements after the *default* if so desired. If multiple *case* options are placed at the same statement in the code, this is handled by ORing the options together in the equivalent WHEN statement, as can be seen in the example of Figure 4.

The handling of *for* loops is through conversion to a WHILE loop, with ST1 going before the loop, ST2 is the conditional of the loop itself and ST3 going as the last statement of the loop. This works even for multiple statements in any of ST1 or ST3, as can be seen in the examples in Figure 5.

The other types of loops are handled similarly. The handling of pre and post incrementing is handled with temporary variables. If the post operation occurs within other expressions, it is handled by updating the variable itself. The temporary variable, which holds the value before the update, is used in the actual expression. In general, all of the standard operators may be applied.

3 An example

The Appendix in section 7 at the end of this paper, includes two separate descriptions. The first is a C program listing. The second is the VHDL description which our C2VHDL converter has produced. C2VHDL does not produce code for all C.

As we are limited by the VHDL subset that SYNT takes, only those constructs which SYNT can handle have been implemented. For instance SYNT cannot handle floating point numbers, therefore we have not allowed floating point numbers in C. The VHDL description which results from this program also does not have an "entity" section. This is because the program can be tailored for differing types of input output (such as serial input output or parallel input output) using the same architecture. Finally, there are a large number of temporary values. These are removed by SYNT when it does register allocation. These temporary values also help SYNT achieve a more efficient hardware synthesis.

```
switch (g) {
        case 1 :
                a = a + b;
                break;
        case 2 :
                a = a + c;
                break;
        case 3 :
        case 6 :
        case 9 :
                a = a + d;
                break;
        case 8 :
                a = a + e;
                break;
        default :
}
```

```
CASE g IS
WHEN 1 =>
                as_1 := a;
                as_2 := b;
        a := as_1 + as_2;
        WHEN 2 =>
                as_3 := a;
        as_4 := c;
        a := as_3 + as_4;
WHEN 3 | 6 | 9 =>
                as_5 := a;
                as_6 := d;
        a := as_5 + as_6;
        WHEN 8 =>
                as_7 := a;
                as_8 := e;
                a := as_7 + as_8;
        WHEN OTHERS =>
                NULL;
END CASE;
```

Figure 4 - SWITCH .. CASE conversion

```
for (ST1; ST2; ST3) {
        .
}
```

```
ST1;
WHILE ST2 LOOP
        .
        ST3;
END LOOP;
```

```
b = 2346;
for (a = 1; a < 10; b = b + 4, a++) {
        b = b / 3;
}
```

```
b := 2346;
a := 1;
as_62 := a;
as_63 := 10;
WHILE as_62 < as_63 LOOP
        as_67 := b;
        as_68 := 3;
        b := as_67 / as_68;
        as_64 := b;
        as_65 := 4;
        b := as_64 + as_65;
        as_66 := a;
        a := a + 1;
END LOOP;
```

Figure 5 - FOR conversion

4 Future research

We are presently experimenting with another synthesis tool, AMICAL, which has the capability to synthesize VHDL procedures and functions. At this preliminary stage it would appear that AMICAL [11] removes a number of other limitations imposed on us by SYNT. Hence any limitations imposed are a direct result of the synthesis tool used.

We are in the process of building working systems using the C2VHDL tool at present. This is incorporated in the overall hardware software codesign project. It is here that problems such as referencing variables which exist in main memory will need to be resolved.

5 Conclusions

We have presented a tool which converts C programs to VHDL. This tool was developed as part of the hardware software codesign project at the University of Queensland. Since the tool is used for synthesis, only those constructs which the VHDL synthesis tool allows are included in this system. As the synthesis tools become more sophisticated, a larger subset of C can be converted to VHDL.

6 References

[1] Matthew F. Parkinson, Paul M. Taylor, and Sri Parameswaran, "An Automated Hardware/Software Codesign (HSC) using VHDL," Proceedings of the First Asia Pacific Conference on Hardware Description Languages and their Applications (APCHDLSA '93), December 1993.

[2] P. Ashenden, "VHDL Cookbook," - Internet

[3] Mats Fredriksson, Ahmed Hemani, Kurt Nordqvist, "SYNT 1.0 USER'S GUIDE," - Internet

[4] Matthew F. Parkinson, Paul M. Taylor, and Sri Parameswaran, "A Profiler for Automated Translation of Signal Processing Algorithms into High Speed Hardware/Software Hybrid Architectures," Proceedings of Microelectronics '93, October 1993.

[5] GNU CC, Reference Manual - Internet

[6] YACC, Reference Manual - Internet

[7] B. Bose, M. E. Tuna, and S. D. Johnson, "System Factorisation in Codesign," Proceedings of the 1993 IEEE International Conference on Computer Design (ICCD '93), October 1993.

[8] P. M. Athanas, and H. F. Silverman, "Processor Reconfiguration Through Instruction-Set Metamorphosis," Computer IEEE, pp. 11-18, March 1993.

[9] R. Ernst, J. Henkel, "Hardware-Software Codesign of Embedded Controllers Based on Hardware Extraction,"

Handout from First Int'l Workshop on Hardware-Software Codesign, Estes Park, Colo., 1992.

[10] R. Gupta, CC. Coelho, and G. De Micheli, "Synthesis and Simulation of Digital Systems Containing Interacting Hardware and Software Components," Proc. DAC, IEEE CS Press, Los Alamitos, Calif., Order No. 2822, 1992, pp. 225-230.

[11] K. O'Brien, M. Rahmouni, P. Kission, M. Aichouchi, A. Jemai, H. Ding, A. A. Jerraya, "AMICAL - Interactive Architectural Synthesis Based on VHDL - User's Manual," System-Level Synthesis Group, Laboratoire TIMA/INPG, 46, Avenue Felix Viallet, 38031, Grenoble CEDEX, FRANCE
(email: obrien@rhone.imag.fr).

7 Appendix

```
#include <stdio.h>

int i;

int A(char *h, short j, char r) {
    unsigned char matt;
    char s[23];
    int g;
    long a;
    int b = 46;
    char d;
    unsigned char *e;
    char f[157];
    long t;

    matt = 'a' + 3;
    b = i + 4;
    i = 1;
    b = 1;
    i = i + b;
    j = j * 5;
    j = 5 * j;
    g = 0;
    switch (g) {
        case 1: a += 1;
            break;
        case 2: a += 2;
            break;
        case 3:
        case 9: a += 3;
            break;
        case 4:
        case 8: a += 4;
            break;
    }
    switch (g) {
        case 5: b /= 3;
            break;
    }
    switch (g) {
        case 5:
            for (t = 0; t < 10; t++) {
                a *= 4;
                break;
            }
            break;
        default:
            a++;
    }
    do {
        t--;
        continue;
        g = g + 1;
    } while (t > 0L);
    while (t > 0L) {
        t--;
        t++;
        break;
        g = g + 1;
    }
    if (g > 10)
        g = 4;
    else
        g = 5;
    (g > 10) ? (g = 4) : (g = 5);
    a = 10 + sizeof(int);
    a = 15 - sizeof(long);
    a = 24 * sizeof(char);
    a = 24 / sizeof(float);
    a = 56 % sizeof(typeof(f));
    a = sizeof(f);
    a = 12 * (-34 && sizeof(typeof(char))) + 52;
    a = 12 * (-34 || sizeof(typeof(char))) + 52;
    a = (34 & 23) / 45;
    a = (34 | 23) / 45;
    a = -34 * 3;
    a = 35 ^ 4;
    a = a + 1;
    a = a + (b * 62);
    a = a << 3;
    a = a >> 2;
    a /= (7 + b);
    a = a * 7;
    a = 4 + ~7 + !a + ~a;
    a = 4 + !7 + !0 + !a;
    a = --b;
    b = 2346;
    for (a = 1; a < 10; b = b + 4, a++) {
        b = b / 3;
    }
    for (a = 1; a < 10; b = b + 4, a++)
        b = b / 3;
    if (a < b) {
        a = 3;
        b++;
    } else {
        b = 4;
        a++;
    }
}
```

Figure 6 - Original C code

```
ARCHITECTURE behaviour OF C2VHDL IS
BEGIN
   PROCESS
      VARIABLE ftell : long;
      VARIABLE i : int;
      VARIABLE h : varpointer;
      VARIABLE j : short;
      VARIABLE r : char;
      VARIABLE matt : unsignedchar;
      VARIABLE g : int;
      VARIABLE a : long;
      VARIABLE b : int := 46;
      VARIABLE d : char;
      VARIABLE e : varpointer;
      VARIABLE t : long;

      VARIABLE as_1 : int4;
      VARIABLE as_2 : int4;
      VARIABLE as_3 : int4;
      VARIABLE as_4 : int4;
      VARIABLE as_5 : int4;
      VARIABLE as_6 : int4;
      VARIABLE as_7 : int2;
      VARIABLE as_8 : int4;
      VARIABLE as_9 : int4;
      VARIABLE as_10 : int2;

   ... and so on ...

      VARIABLE as_142 : int4;
      VARIABLE as_143 : int4;
      VARIABLE as_144 : int4;

   BEGIN
      as_1 := 97;
      as_2 := 3;
      matt := as_1 + as_2;

      as_3 := i;
      as_4 := 4;
      b := as_3 + as_4;

      i := 1;

      b := 1;

      as_5 := i;
      as_6 := b;
      i := as_5 + as_6;

      as_7 := j;
      as_8 := 5;
      j := as_7 * as_8;

      as_9 := 5;
      as_10 := j;
      j := as_9 * as_10;

      g := 0;

      CASE g IS
         WHEN 1 =>
            a := a + 1;

         WHEN 2 =>
            a := a + 2;

         WHEN 3 | 9 =>
            a := a + 3;

         WHEN 4 | 8 =>
            a := a + 4;

         WHEN OTHERS =>
            NULL;
      END CASE;

      CASE g IS
         WHEN 5 =>
            as_11 := b;
            as_12 := 3;
            b := as_11 / as_12;

         WHEN OTHERS =>
            NULL;
      END CASE;

      CASE g IS
         WHEN 5 =>

            t := 0;

            as_13 := t;
            as_14 := 10;
            WHILE as_13 < as_14 LOOP
               a := a * 4;

--             EXIT;
               as_15 := t;
               t := t + 1;
            END LOOP;
         WHEN OTHERS =>
            NULL;
            as_16 := a;
            a := a + 1;
      END CASE;

      as_17 := '1';
      WHILE (as_17 = '1') LOOP
         as_18 := t;
         t := t - 1;
--       NEXT;
         as_19 := g;
         as_20 := 1;
         g := as_19 + as_20;

         as_21 := t;
         as_22 := 0;
         IF (as_21 > as_22) THEN
            as_17 := '1';
         ELSE
            as_17 := '0';
         END IF;
      END LOOP;

      as_23 := t;
      as_24 := 0;
      WHILE as_23 > as_24 LOOP
         as_25 := t;
         t := t - 1;
         as_26 := t;
         t := t + 1;
--       EXIT;
         as_27 := g;
         as_28 := 1;
         g := as_27 + as_28;
```

```
END LOOP;

as_29 := g;
as_30 := 10;
IF as_29 > as_30 THEN
   g := 4;

ELSE
   g := 5;

END IF;

as_31 := g;
as_32 := 10;
IF (as_31 > as_32) THEN
   g := 4;

ELSE
   g := 5;

END IF;

as_33 := 10;
as_34 := 4;
a := as_33 + as_34;

as_35 := 15;
as_36 := 4;
a := as_35 - as_36;

as_37 := 24;
as_38 := 1;
a := as_37 * as_38;

as_39 := 24;
as_40 := 4;
FOR as_j IN 0 TO 31 LOOP
   IF ((as_39 / 2) /= ((as_39 + 1) / 2)) THEN
      as_41(as_j) := '1'
   ELSE
      as_41(as_j) := '0'
   END IF;
   IF ((as_40 / 2) /= ((as_40 + 1) / 2)) THEN
      as_42(as_j) := '1'
   ELSE
      as_42(as_j) := '0'
   END IF;
END LOOP
a := as_44;

as_45 := 56;
as_46 := 157;
a := as_45 MOD as_46;

a := 157;

as_47 :=  - 34;
as_48 := 1;
IF NOT(as_47 = 0) AND NOT(as_48 = 0) THEN
   as_49 := 1;
ELSE
   as_49 := 0;
END IF;
as_50 := 12;
as_51 := (as_49);
as_52 := as_50 * as_51;
as_53 := 52;
a := as_52 + as_53;

as_54 :=  - 34;
as_55 := 1;
IF NOT(as_54 = 0) OR NOT(as_55 = 0) THEN
   as_56 := 1;
ELSE
   as_56 := 0;
END IF;
as_57 := 12;
as_58 := (as_56);
as_59 := as_57 * as_58;
as_60 := 52;
a := as_59 + as_60;

as_61 := 34;
as_62 := 23;
IF as_61 < 0 THEN
   as_61 := as_61 + 2147483647;
   as_61 := as_61 + 1;
END IF;
IF as_62 < 0 THEN
   as_62 := as_62 + 2147483647;
   as_62 := as_62 + 1;
END IF;
as_63 := 0;
as_64 := 1;
FOR as_j IN 0 TO 31 LOOP
   IF ((as_61 / 2) /= ((as_61 + 1) / 2)) AND
((as_62 / 2) /= ((as_62 + 1) / 2)) THEN
      as_63 := as_63 + as_64;
   END IF;
   as_61 := as_61 / 2;
   as_62 := as_62 / 2;
   as_64 := as_64 * 2;
END LOOP;
as_65 := (as_63);
as_66 := 45;
FOR as_j IN 0 TO 31 LOOP
   IF ((as_65 / 2) /= ((as_65 + 1) / 2)) THEN
      as_67(as_j) := '1'
   ELSE
      as_67(as_j) := '0'
   END IF;
   IF ((as_66 / 2) /= ((as_66 + 1) / 2)) THEN
      as_68(as_j) := '1'
   ELSE
      as_68(as_j) := '0'
   END IF;
END LOOP
a := as_70;

as_71 := 34;
as_72 := 23;
IF as_71 < 0 THEN
   as_71 := as_71 + 2147483647;
   as_71 := as_71 + 1;
END IF;
IF as_72 < 0 THEN
   as_72 := as_72 + 2147483647;
   as_72 := as_72 + 1;
END IF;
as_73 := 0;
as_74 := 1;
FOR as_j IN 0 TO 31 LOOP
   IF ((as_71 / 2) /= ((as_71 + 1) / 2)) OR
((as_72 / 2) /= ((as_72 + 1) / 2)) THEN
      as_73 := as_73 + as_74;
   END IF;
   as_71 := as_71 / 2;
```

```
            as_72 := as_72 / 2;
            as_74 := as_74 * 2;
         END LOOP;
         as_75 := (as_73);
         as_76 := 45;
         FOR as_j IN 0 TO 31 LOOP
            IF ((as_75 / 2) /= ((as_75 + 1) / 2)) THEN
               as_77(as_j) := '1'
            ELSE
               as_77(as_j) := '0'
            END IF;
            IF ((as_76 / 2) /= ((as_76 + 1) / 2)) THEN
               as_78(as_j) := '1'
            ELSE
               as_78(as_j) := '0'
            END IF;
         END LOOP
         a := as_80;

         as_81 :=  - 34;
         as_82 := 3;
         a := as_81 * as_82;

         as_83 := 35;
         as_84 := 4;
         IF as_83 < 0 THEN
            as_83 := as_83 + 2147483647;
            as_83 := as_83 + 1;
         END IF;
         IF as_84 < 0 THEN
            as_84 := as_84 + 2147483647;
            as_84 := as_84 + 1;
         END IF;
         as_85 := 0;
         as_86 := 1;
         FOR as_j IN 0 TO 31 LOOP
            IF ((as_83 / 2) /= ((as_83 + 1) / 2)) XOR
((as_84 / 2) /= ((as_84 + 1) / 2)) THEN
               as_85 := as_85 + as_86;
            END IF;
            as_83 := as_83 / 2;
            as_84 := as_84 / 2;
            as_86 := as_86 * 2;
         END LOOP;
         a := as_85;

         as_87 := a;
         as_88 := 1;
         a := as_87 + as_88;

         as_89 := b;
         as_90 := 62;
         as_91 := a;
         as_92 := (as_89 * as_90);
         a := as_91 + as_92;

         as_93 := a;
         as_94 := 3;
         a := as_93 * (2 ** as_94);

         as_95 := a;
         as_96 := 2;
         a := as_95 / (2 ** as_96);

         as_97 := 7;
         as_98 := b;
         as_99 := a;
         as_100 := (as_97 + as_98);
         a := as_99 / as_100;
```

```
         as_101 := a;
         as_102 := 7;
         a := as_101 * as_102;

         as_103 := 4;
         as_104 := (2147483647 - 7);
         IF a = 0 THEN
            as_105 := 1;
         ELSE
            as_105 := 0;
         END IF;
         as_106 := as_103 + as_104;
         as_107 := as_105;
         as_108 := as_106 + as_107;
         as_109 := (2147483647 - a);
         a := as_108 + as_109;

         IF 7 = 0 THEN
            as_110 := 1;
         ELSE
            as_110 := 0;
         END IF;
         as_111 := 4;
         as_112 := as_110;
         IF 0 = 0 THEN
            as_113 := 1;
         ELSE
            as_113 := 0;
         END IF;
         as_114 := as_111 + as_112;
         as_115 := as_113;
         IF a = 0 THEN
            as_116 := 1;
         ELSE
            as_116 := 0;
         END IF;
         as_117 := as_114 + as_115;
         as_118 := as_116;
         a := as_117 + as_118;

         b := b - 1;
         a := b;

         b := 2346;


         a := 1;

         as_119 := a;
         as_120 := 10;
         WHILE as_119 < as_120 LOOP
            as_124 := b;
            as_125 := 3;
            FOR as_j IN 0 TO 31 LOOP
               IF ((as_124 / 2) /= ((as_124 + 1) / 2))
THEN
                  as_126(as_j) := '1'
               ELSE
                  as_126(as_j) := '0'
               END IF;
               IF ((as_125 / 2) /= ((as_125 + 1) / 2))
THEN
                  as_127(as_j) := '1'
               ELSE
                  as_127(as_j) := '0'
               END IF;
            END LOOP
            b := as_129;
```

```
            as_121 := b;
            as_122 := 4;
            b := as_121 + as_122;

            as_123 := a;
            a := a + 1;
         END LOOP;

         a := 1;

         as_130 := a;
         as_131 := 10;
         WHILE as_130 < as_131 LOOP
            as_135 := b;
            as_136 := 3;
            FOR as_j IN 0 TO 31 LOOP
               IF ((as_135 / 2) /= ((as_135 + 1) / 2))
THEN
                  as_137(as_j) := '1'
               ELSE
                  as_137(as_j) := '0'
               END IF;
               IF ((as_136 / 2) /= ((as_136 + 1) / 2))
THEN
                  as_138(as_j) := '1'
               ELSE
                  as_138(as_j) := '0'
               END IF;
            END LOOP
```

```
            b := as_140;

            as_132 := b;
            as_133 := 4;
            b := as_132 + as_133;

            as_134 := a;
            a := a + 1;
         END LOOP;

         as_141 := a;
         as_142 := b;
         IF as_141 < as_142 THEN
            a := 3;

            as_143 := b;
            b := b + 1;
         ELSE
            b := 4;

            as_144 := a;
            a := a + 1;
         END IF;

         WAIT ON clk UNTIL (in_rdy = '1');
      END PROCESS;
   END behaviour;
```

Figure 7 - Converted VHDL code