Programming the Harp-2 using Handel-C

Andrew Bailey
University of Oxford

June 23, 1996

Disclaimer: The Handel-C compiler has yet to be extensively test in programming the Harp-2 board. This note aims at helping the novice get started in programming the board but may be inadequate for some tasks. I hope to rectify this soon, however, there is one known problem which is described in the text.

1 Introduction

In order to program the Harp-2 using Handel-C requires the use of the Occam Toolset (referred to as OTS) and the Xilinx tools (which will be collectively referred to as XACT). It is assumed that the reader has set these two systems up correctly and is able to use them at least for small examples. There are two other documents that contain relevant information: the Handel-C manual and "The HARP Software Library and Utility Packages", both of which should be in the directory where this document was found. Chapter 5 of the Handel-C manual contains some information relevant to programming the Harp-2 cards but only the descriptions given in this document have been known to generate working programs on a Harp-2. These are described by way of an example: `num.c`.

2 Example: num.c

All the source files described in this section can be found in `ex/num`.

`num.c` is probably one of the simplest of Handel-C programs, being a one place buffer between an output and input channel on the Harp-2 transputer. Chapter 5, particularly section 5.2.1, describes the intention of the `spec` declaration, whilst the behaviour is captured by the `while` loop. The constant `dw` is the data width of the integers used throughout the program.

```
/* HCC File
 * File:                num.c
 * Purpose:             Reads value from transputer and send it back
 * Author:              Andrew Bailey, Hardware Compilation Group, OUCL
 * Date:                6 June 1996
 * Documentation:
 * Related Files:       num.occ, harpnum.occ, num.pgm
 */

const spec harp2 = {
  fpga_type       = "Xilinx3000",
  fpga_chip       = "3195APQ160-3",
  clock_pad       = "P160",
  not_error_pad   = "P55",
  finish_pad      = "P44",
  clock_divider   = "1",
  carry_weight    = "50",
  critical_weight = "100"
};

const dw = 8;
const forever = 1;

void main(target = harp2,
          chan (in) cin : dw,
          chan (out) cout : dw)
{
  int num : dw;
  while (forever){
    cin ? num;
    cout ! num;
  }
}
```

If this file is processed by `hcc` (the Handel-C compiler) by the invocation `hcc num.c` a file `num.xnf` is generated. This is suitable for input to the XACT tools, however, it also contains code for two other systems within it as USER comments. These are an Occam program, for the harp transputer end of the communication channels specified in the Handel-C program, and a list of commands for the XACT layout editor to place and route the time critical components of the implementations of the channels in the FPGA. Both these fragments should be extracted in to separate files, either using an editor, or using the supplied PERL script, `extract.pl` (`extrac.pl .occ` xnffile gets the Occam fragment and `extract.pl .mac` xnffile gets the LCA commands and puts them in a file with the XNF file root name with `.occ` and `.mac` extension respectively).

Using `num.xnf` and `num.mac` we can now generate the FPGA configuration as follows:

1. Start the XACT editor and select expert mode and the appropriate FPGA part and Speed (this example uses the 3195APQ160 part and -3 speed). Select the new file option and give it a name (I

use xnum.lca) and select editlca. Once the screen is ready, type execute num.mac at the command line and wait a few minutes while the layout and routing takes place. The resulting layout can be found at the top right of the FPGA. So far only the time critical nets have be routed, the balance of those possible at this time (the design is still only partially complete) can be done by issuing the `route *` command. Once this is completed the partial design can be saved for use as a guide file.

2. Process the handel-C xnf in order to make a rawbits file as follows:

```
xnfmerge num

xnfprep num

lca2xnf -b xnum num.pgf

xnfmap -k num

ppr num guide=xnum

makebits -b -t num
```

The use of `lca2xnf` on `xnum.lca` generates a guide file for `xnfmap` (-k option); given the messages from `xnfmap` it's not clear whether this is having an effect but this is merely following guidelines in the XACT documentation. From this sequence of tools a rawbits file, `num.rbt`, is generated which will be used to generate an Occam code fragment (a procedure described later) to configure the FPGA to behave as described by the Handel-C program.

We can now generate programs for the Harp-2 transputer and associated network. The network used for this example consists of a T805 root transputer and a Harp-2 card attached to link 2 of the root via link 1. You should specify your network by editing `network.inc` in `harp/libs`.

For the root transputer a very simple program is used that takes and send values from the host system and sends and receives values to the Harp-2 transputer, as shown below:

```
#INCLUDE "hostio.inc"

PROC num (CHAN OF SP fs, ts,
          CHAN OF INT to.harp.tp, from.harp.tp)
  #USE "hostio.lib"
  INT inval, htval:
  BOOL err:
  SEQ
    so.write.string (fs, ts, "Enter Clock Freq (MHz): ")
    so.read.echo.int(fs, ts, inval, err)
    to.harp.tp ! inval
    WHILE TRUE
      SEQ
        so.write.string (fs, ts,"Enter a number: ")
        so.read.echo.int (fs, ts, inval, err)
        so.write.nl (fs,ts)
        to.harp.tp ! inval
        from.harp.tp ? htval
        so.write.string(fs,ts, "Number is: ")
        so.write.int(fs, ts, htval, 8)
        so.write.nl(fs,ts)
:
```

The program for the Harp transputer is as follows:

```
PROC harp.tp.num(CHAN OF INT from.root.tp, to.root.tp)
  #INCLUDE "harp1dec.inc"
  #INCLUDE "harp1lpc.inc"
  -- #USE "harp1all.lib"   -- causes problems with imakef
  #USE "harp1ba.lib"       -- Harp basic library
  #USE "harp1f.lib"        -- Harp frequency library
  #INCLUDE "fpgaconf.inc"  -- configuration of FPGA; from xbits
  #INCLUDE "hcnum.occ"     -- Occam generated by Handel-C

  VAL MHz IS 1000000:

  PROC setup.FPGA(VAL []INT config, VAL INT length.count, VAL INT freq)
    BOOL programmed:
    INT val:
    VAL clock.off IS pll.data \/ (pll.notstb \/ (pll.mx.ss \/ pll.block)):
    SEQ
      configure(config, length.count, clock.off, programmed) -- from harp1ba.lib
      CHAN OF INT frequency:
      CHAN OF report reply:
      PAR
        set.for.frequency(frequency,reply)      -- from harp1f.lib
        SEQ
          frequency ! (freq * MHz)
          reply ? CASE pll.value; val
      FPGA.clock.control (val)                  -- from harp1ba.lib
  :

  -- Abbreviations to undo Adrian's/Handel's nosensical ones!
  read.ready IS ready.C.cout.0:
  read.data IS FPGA.chan.ports[1] :
  write.ready IS ready.ChanOut.Bus.3:
  write.data IS FPGA.chan.ports[0]:
  INT inval,rtval:
  SEQ
    from.root.tp ? inval
    setup.FPGA(config, length.count, inval)
    PAR
      Event.Handler(write.data,[read.ready,write.ready])
      WHILE TRUE
        SEQ
          from.root.tp ? inval
          write.ready ! TRUE
          write.data ! inval
          read.ready ! TRUE
          read.data ? rtval
          to.root.tp ! (rtval /\ #FF)
:
```

The two INCLUDEd files contains the hardware and port configuration of the connection between the transputer and FPGA (`harp1plc.inc` actually includes these two thus you need actually include just that file).

`harpl1all.lib` contains all the other supplied harp libraries. However, because of the way OTS `imakef` works, the resulting makefile requires editing because of a line length problem. In this case I have elected to use just the two libraries required for this program to avoid this problem.

`fpgaconf.inc` is generated from `num.t90` by the transputer program `xbits` which is in `harp/utils`. Note that this program appends to the specified output file so it is best to delete old version before running `xbits`. A program `rbttoocc` is also available in the sun4 `bin` but I have not used this in anger yet.

`hcnum.occ` is the file extracted from the Handel-C generated XNF file. As generated it contains some declarations for use in the users Occam programs. The principle object is an event handler process which controls data transfer on each of the channels specified and is specific to the number of channels between the FPGA and Transputer (see the Libraies manual). Unfortunately this cannot be used as is because it contains two versions for the event handler - I have used version 1. The actual resulting file can be found in `ex/num`.

`setup.FPGA` is, as the name suggests, a procedure to setup the FPGA and clock generator. I am currently at a loss to describe how much of this procedure works but it is probably adequate for most situations. Detail examination of the Harp hardware manual and library and utilities manual is probably required before experimenting with changes to this procedure.

The initial `SEQ` of the program receives a value from the host via the root transputer to set the clock frequency for use in the call of `setup.FPGA`. The parameters `config` and `length.count` come from `fpgaconf.inc` being an Occam table with the FPGA configuration and its size respectively.

Once the FPGA is configured, the event handler is started in parallel with the main program. This loop demonstrates the necessary steps to read and write to the FPGA. To write, the program must wait until TRUE is received on the ready channel, similarly for reading. The last thing to note that although the FPGA is configured to receive bytes, the transputer sends and receives (reads) INTs (32 bits). Hence it is necessary to mask the result to the bitwidth of the channel.

The pgm file for `num` is quite straight forward:

```
#INCLUDE "network.inc"

#INCLUDE "hostio.inc"
#USE "num.c9h"
#USE "harpnum.c9h"

CONFIG
  CHAN OF SP fs, ts :
  PLACE fs, ts ON hostlink :
  CHAN OF INT root.to.harp, harp.to.root:
  PLACE root.to.harp, harp.to.root ON blink:
  PLACED PAR
    PROCESSOR root.t
      num(fs,ts,root.to.harp,harp.to.root)
    PROCESSOR harp1
      harp.tp.num(root.to.harp,harp.to.root)
:
```

`network.inc` describes the transputer network as described earlier.

A point to note is that the event handler code is parameterised on the number of read and write channels between the transputer and FPGA, but not the channel widths. Thus the same code can be used for different values of `dw` in the Handel-C program (remember to change the mask value in `harpnum.occ` as appropriate). Although this can be as large as 32 we are currently getting incorrect results with 24 and 32. We know that 4, 8 and 16 work and guess that all channel widths less than 16 will work. We believe this may be a software problem (possibly Xilinx layout).