A Fax Decoder on the XC6200

XAPP 085 July 25, 1997 (Version 1.0)

Application Note by Douglas M Grant

Summary

Part of a fax decoder circuit is designed in VHDL which, with the aid of with some simple software, can decode fax-format data. The circuit is mapped onto a XC6216 FPGA within XC6000DS development system PCI board to accelerate the most cycle-intensive parts of the decoding algorithm. This note describes the design of the accelerator circuit for the XC6216 and demonstrates simple and effective codesign.

Xilinx Family

- XC6200

Demonstrates

- VHDL
- Hardware/Software Co-design
- XC6000DS
- Reconfigurable Computing

Table of Contents

Introduction

It is practical to implement applications on FPGAs which were formerly only possible with ASIC technology. The benefits of using an FPGA include the low NRE, low risk and independence of shifting standards. In the communications world especially, the accelerating improvements in bandwidth and world-wide interconnectivity mean that new standards are an almost daily occurrence. In order to keep pace with these technological advances it is no longer possible to design and develop ASICs in the time available and so the use of FPGAs becomes a necessity in many systems. Their use allows new standards to be implemented in just a few days and the hardware updated accordingly in fractions of a second.

The ITU (formerly the CCITT) group 3 fax transmission standard is a standard which, while quite stable, will need to be improved to exploit the growing ISDN-2 communications infrastructure. There is already a draft for an update in progress. The standard describes both the transmission protocols themselves as well as the binary coding and compression methods used. The standard is also used in applications other than actual fax transmission, for instance as a compression/decompression method for sending data to a printer over a network. Processing power and memory required to implement decompression in real time can be very expensive, and an FPGA-based accelerator can reduce these costs and make real-time operation at video rates possible.

An important application of this compression technology is in frame buffer controllers where the memory requirement can be minimized. This gets used extensively in printer controllers.

The XC6000 Development System

The XC6000 Development system comprises some low-level software and a PCI board containing an XC6216, an XC4013E to implement the PCI protocols and some other functions. It defines the hardware/software interface.

With the XC6000DS software provided, Reads and Writes of up to 32 bits can be made to any 32 registers in a column of the XC6216 (this could be more using wildcarding). The control registers on the XC6216 may also be written and read with this software. The value on any gate, multiplexer or register output may be read from the circuit, which to the host processor looks like an SRAM. The hardware can be clocked by the software. This makes for a development system that enables hardware to be tested as part of the software development process.

CCITT G3 Fax Standard [1]

Each line of 1728 pixels scanned from the source document is transmitted, encoded as a pulse stream lasting a minimum of 20ms. Lines with little or no features may be "padded" with some silence during transmission (silence is equivalent to sending '0's).

The coding part of the standard describes two sets of 91 Huffman codes, from 2 bits to 13 bits in length. A Huffman code is one that does not form the beginning of any other code. If the code '11' represents some number of consecutive pixels then no other code may begin with a '11'. The two sets of 91 codes are for Black pixel data and White pixel data. Each set is made up of 64 codes to represent a run of 0 to 63 pixels - the so-called Termination codes; and 27 codes to represent runs of 1 to 27 blocks of 64 pixels each - the so-called Makeup codes.

There is an extra 12 bit code that represents an end-of-line (EOL) and finally any spare '0's in the transmission are considered to be padding, or FILL bits. An end-of-page (EOP) is coded as two consecutive EOLs and an end-of-transmission (EOT) as six consecutive EOLs.

Every line starts with one or two codes for white pixel data, and the colour of the data then alternates every one or two codes. A pixel run of one colour is represented by either a single Termination code or a by Make-up code followed by a Termination code. The bit stream given in **Figure 1** would represent a blank white line. Huffman coding also allows us to send shorter codes for more common pixel data, and in general around 80% of codes transmitted will be 6 bits or less.

```
010011011      00110101     000000000001
27*64 pixels   0 pixels      EOL
```

Figure 1: Example of transmitted bit stream

Circuit architecture

Partitioning and architectural choices.

The most cycle-intensive part of fax decoding in software is the detection of valid codes. A variable-length decoding circuit for Huffman codes, as required here, is most efficiently implemented in hardware by a binary decoding tree. Since 80% of fax codes transmitted have a run-length of 6 bits or fewer, it is natural to build a 6-bit decoding tree to begin with. Also, because there are no 1-bit codes, the tree may be constructed from two, 5-bit decoding trees. These trees are fed with the LSB, either directly or via an inverter to generate a 1', which then filters through the tree structures, guided by the values of the more significant (or later arriving) bits. At certain points in the trees, valid codes are marked with a flag bit. If the current stream of bits entering the tree contains a valid code, this is reflected by a '1' bit on the output which is associated with that number of bits. Software may then slice that many bits from the input stream and look up the corresponding number of white or black pixels to be output to paper. **Figure 2** shows the simple structure of a decoding tree, and shows how a code of 1110 (from LSB-Bit0..Bit3) is detected. By applying the incoming bit stream to the columns of cells and ORing the outputs of nodes in a column, the number of bits in a valid code, or "hit", can be detected. Only the number of bits in a valid code is important and ORing together pairs of the so-called "hit" lines from the two 5-bit trees produces 5 outputs representing a code length of 2 to 6 bits.

With a 6-bit decoding tree chosen, more decisions on the architecture may be made. Since the data received in a fax transmission relates alternately to runs of white and black pixels, it seems natural to have a decoding tree for each one. This will avoid reprogramming a single tree with new valid code points after each white or black code is detected. Chip area is not a concern at this point, since it costs almost nothing on a reconfigurable FPGA such as the XC6216 unlike on an ASIC where area considerations can be critical to yield and cost.

The longest valid fax codes are 13 bits long. By adding a second 6-bit decoding tree for both white and black data, codes of length 12 bits may be detected. In order to detect a code of 7 to 12 bits, the second tree must be dynamically programmed. The programmed codes are dependent on the value of the first six bits of the codes. The approach taken here is to have the software determine which valid codes to program the second tree with, whenever no 2 to 6 bit code is detected. 13-bit codes only exist for black pixel data and a following section describes the simple circuit required to detect these in hardware. The logic to detect an EOL code and a FILL bit are also described.

If no valid code is detected then the software discards the LSB and code detection continues. Otherwise the software slices the requisite number of bits from the input stream, looks up their equivalent pixel run-length code from a table, and outputs that many pixels of the given colour to paper (or file).

It only remains to develop the input and output hardware for the trees. In this case the input takes the form of a parallel-load shift register, which can be written to via the processor interface. The following section describes the circuit in detail.

Input shift register.

The shift register is based on the FDC D-type flip-flop component which uses a single cell in the XC6200 family. The architecture is simply a chain of these flip-flops, the lower 32 of which are multiplexed with RPFDC parts. The latter are registers which can only be written to by the processor interface (The "RP" stands for "register protected"). **Figure 3** shows the circuit. It should be noted that the LSB of the parallel-input RPFDCs is used to generate the clock for the shift register, by attaching a CBUF_OUT primitive to that register and then feeding the pulse generated by a read from or write to this cell onto the global G1 clock routing with a BUFGP part. Separate RPFDCs are also used to generate the control for the multiplexers and the clear signal for the shift register. The shift register forks into two 16-bit shift registers,

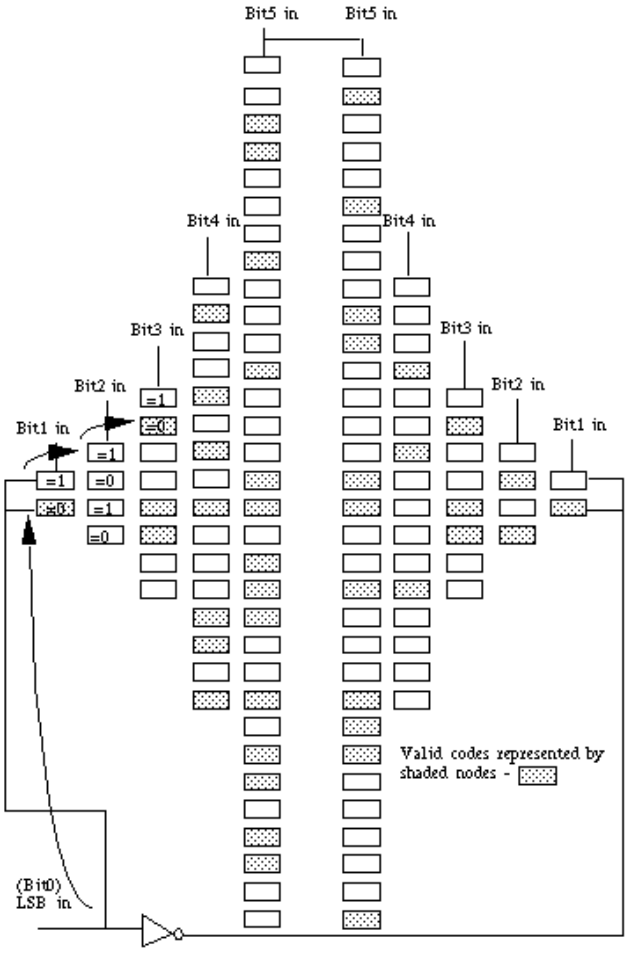

Figure 2: Six-bit binary decoding tree at work

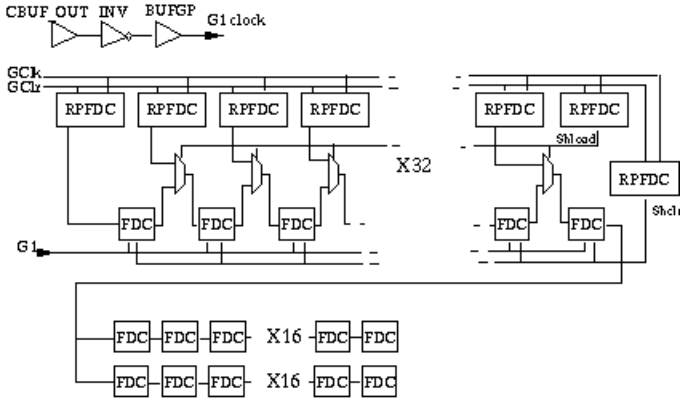

Figure 3: Input shift register

one for white pixel data and one for black.

Decoder tree cell.

The decoder trees are made up from simple cells which have the circuits shown in **Figure 4**. The exact circuit depends on the location in the tree, as intimated in the diagram. The RPFDC is loaded (at run time) with a '1' if the cell is a node in the tree that represents a valid code word. A single decoder cell can be mapped to 3 cells on the XC6216 array.

5-bit decoder tree.

The 5-bit decoder tree is made up of 62 of the cells, with 63 being required for the second decoder trees. These are connected as shown in **Figure 5**.

6-bit decoder tree.

A 6-bit decoding tree may be constructed from two 5-bit trees. **Figure 6** shows a decoding tree for the first 6 bits of data. The trees for the next 6 bits have a '1' constant replacing the bit0 input, and bits 6..11 instead of bits1..5 as the other inputs. It has 6 output OR gates instead of 5.

12-bit decoder tree.

A 12-bit decoding tree has 12 inputs and 11 outputs, representing valid code lengths of 2 to 12 bits. **Figure 7** shows such a tree alongside part of the input shift register which is used to feed it parallel data.

13th bit decoder.

The CCITT standard defines 13 bit codes for black pixel data only. These codes are paired, sharing a common first 12 bits within each pair. So if one of those 12-bit codes is detected then whatever the value of the 13th bit, a valid 13-bit code is detected. The 12-bit codes are detected by monitoring the outputs from the appropriate leaves of the second, black decoding tree with a network of OR gates. A '1' on any input will propagate through to a '1' on the "13-bit code found" output of the tree. Since all the 13-bit codes also have bit6 = '1', only the leaves of one half of the 6-bit tree need to be monitored. **Figure 8** shows how this works.

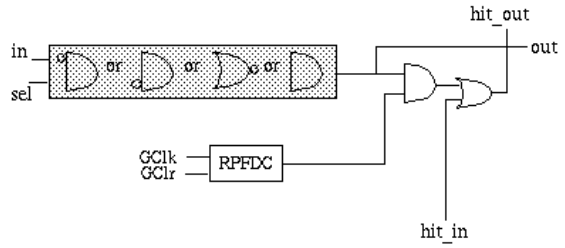

Figure 4: Decoder tree cell

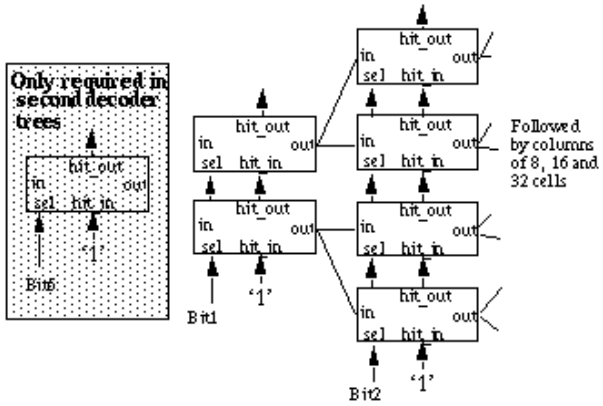

Figure 5: 5-bit decoder tree

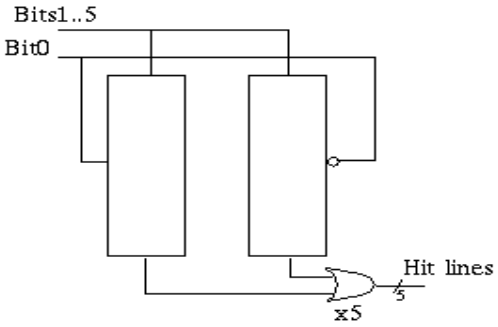

Figure 6: 6-bit decoder tree

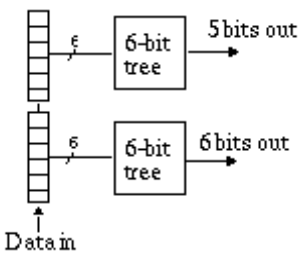

Figure 7: A 12-bit decoder tree

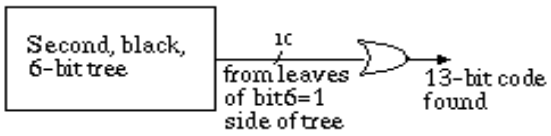

Figure 8: 13th bit decoder

FILL bit and EOL detector.

To re-iterate, an EOL code is defined as 11 '0's followed by a '1'. By NORing together the first 11 bits of the input stream, and ANDing the result with the 12th bit an EOL detector is designed. A FILL bit is defined as any spare '0's in the sequence. These may only appear just before a EOL code is transmitted, so detecting a run of 12 '0's is enough to be sure that the first '0' is a FILL bit. The circuit for FILL bit detection can be combined with that for the EOL, resulting in the circuit shown in **Figure 9**.

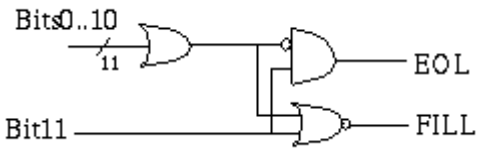

Figure 9: EOL and FILL detector

Further partitioning

At this stage of the design it is apparent that there is still be a lot of chip area left unused, so a further section of software may be selected for implementation in hardware. The act of slicing the correct number of bits from the input remains a software task, since the software has to clock the shift register the correct number of times after each valid code is detected. The software, which looks up the pixel run-lengths that correspond to a valid code, is simple to replace with some hardware.

Run-length LUTs

Each of the 6-bit decoding trees has a corresponding LUT created in hardware, which takes the appropriate, raw bit-stream values as inputs and outputs the correct number of pixels (or blocks of pixels). This output can be read back by the software.

The LUTs are defined as truth-tables, with each having 6 output bits to describe a pixel run-length of up to 63, and an extra output bit which signals a make-up code as opposed to a termination code.

Outputs

The outputs from each 6-bit tree are the OR gates which combine the “hit” lines. These may be read from directly by software. The LUT outputs are extracted from wherever in the logic block they are produced, to a column of buffers which software may also read from.

Complete architecture

The complete architecture is shown in **Figure 10**.

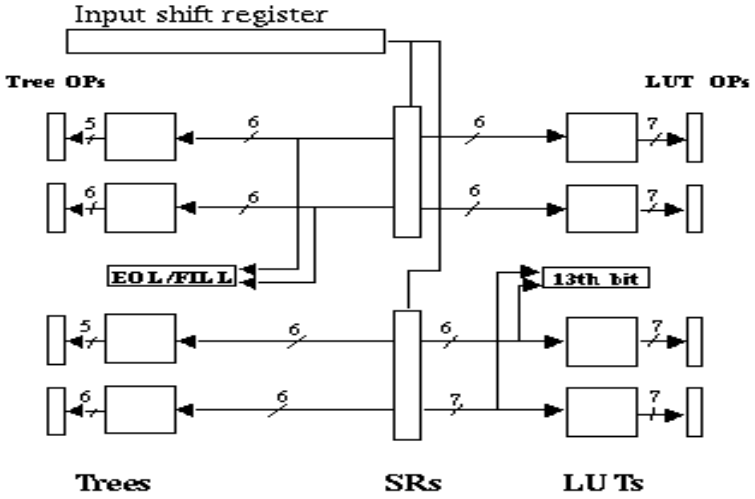

Figure 10: Complete architecture of fax decoder

Layout - Thinking ahead

Now that the details of the architecture have been fleshed out to a register-level design, and before starting to describe the circuit in VHDL, it is important to make an initial floorplan. Since there are fewer than 64x3 cells in a 5-bit decoder tree, it is possible to pack these into an area of 32x6 cells. This is best done by interleaving the nodes of the tree in the order suggested by the structure in **Figure 2**. The 32 leaves make up one column of 32x3 cells. The first node will be placed about half way up the other column, the next two nodes above and below that, the next four nodes spread as equally as possible between those, and so on. It looks as if the tree has been "squashed" into 2 columns instead of 5 or 6. Two of these trees make up a 6-bit detector and the two 6-bit detectors for each of black and white data are placed on the left and right sides of the IC, with the black hardware above the white. The most complex LUT will not require more than around 100 gates (and therefore 100 cells), This will fit easily into a 32x15 cell block which is available beside each decoding tree. **Figure 11** shows this initial floorplan.

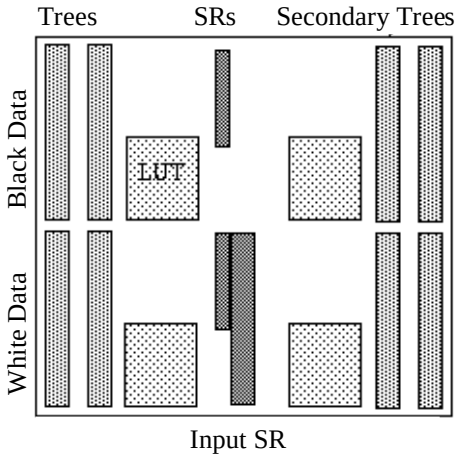

Figure 11: Initial floorplan on XC6216.

VHDL design flow

The entire design has been captured as a mix of VHDL and truth-tables. The latter are synthesised into logic - a netlist which is made available as more VHDL code which can then be incorporated with the rest of the design. The trees, shift registers and other hardware rely heavily on the use of the generate construct in VHDL. The initial floorplan can also be incorporated into the VHDL description as RLOC attributes which can be attached to the various levels of hierarchy during elaboration. **Figure 12** describes the design flow.

As an alternative to using Synopsys for elaboration, a more targeted elaborator, 'Velab' [3], can be used. This has been written specifically for the XC6200 family, and includes the propagation and evaluation of parametrisable attributes in VHDL'93. (Velab is available as Freeware on the Xilinx web page.) Timing analysis may be done before and after place and route (P&R), and delays may be back annotated into the VHDL in SDF format, which the back-end P&R tool produces.

Program design

Application code must be written, which allows the PC to use the design on the XC6000DS [2]. This can be done concurrently with the hardware design. The place and route constraints in the VHDL are used to determine the location of cells that will be used as I/O ports by the software. It is a simple matter to trade off hardware for software and vice-versa at a late stage in the design flow.

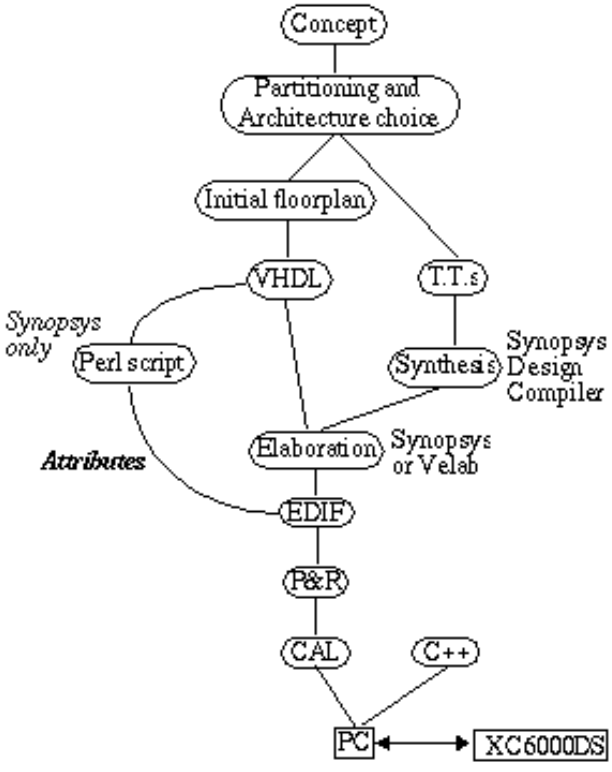

Figure 12: VHDL design flow

Figure 13 shows the program flow that implements all the control and sequencing parts of the decoding algorithm.

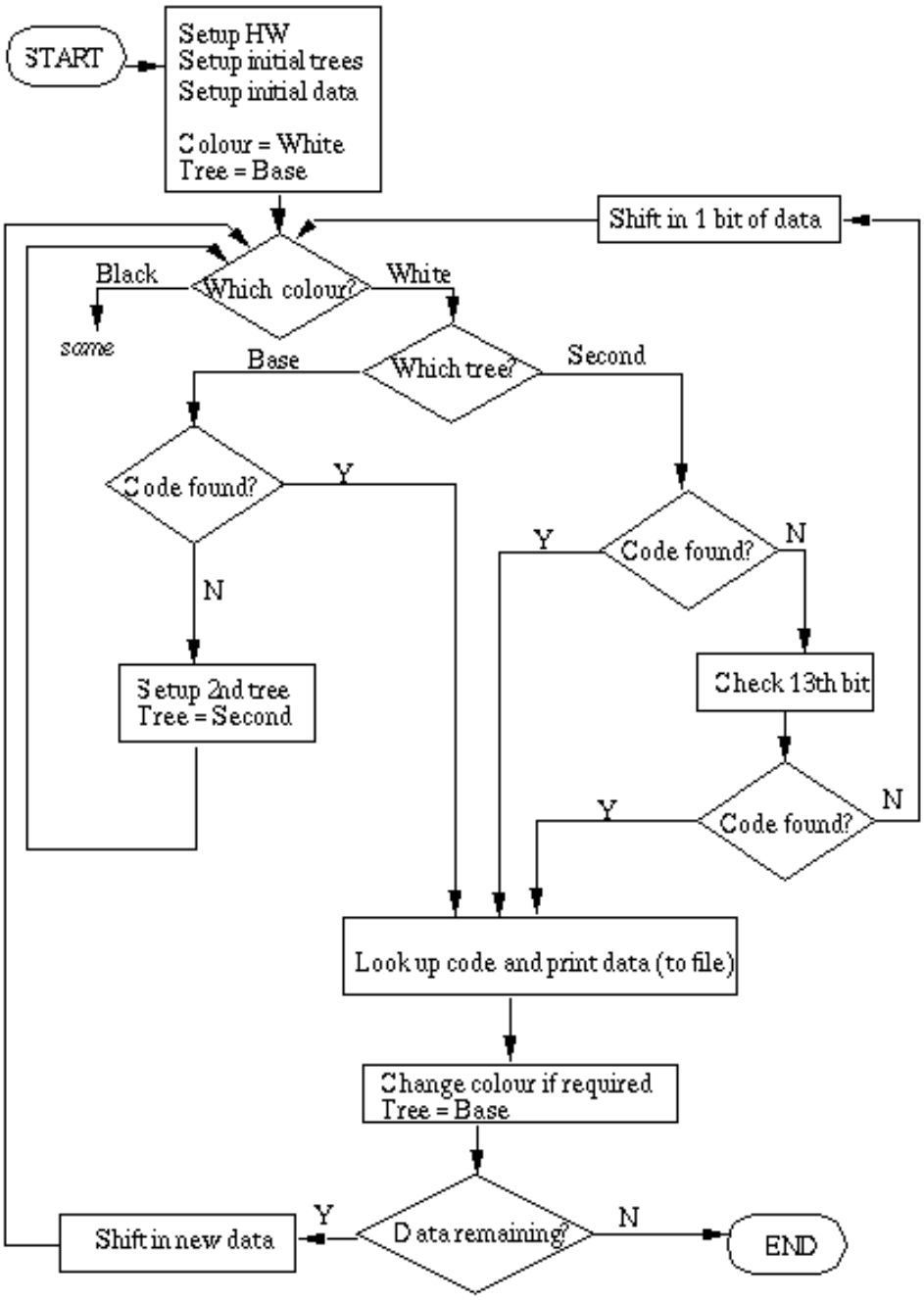

Figure 13: Software flow diagram

Reprise

The application has been wrapped in a Windows application, allowing a bitmap to be made for the decoded data and displayed on screen. The performance of the accelerator has been measured to be some four times better than a wholly software-based approach. This would be greatly enhanced if the XC6216 was directly addressed by the host processor rather than via the PCI bus. Burst-mode access on the PCI bus can increase performance.

This exercise proves the ease of hardware/software co-design for applications based on the XC6000DS. It can be used as a template for mapping more applications to a mixture of hardware and software, based around the XC6000DS.

The use of VHDL as a design methodology is critical to obtaining working designs in a short time. Floorplanning at an early stage in the design flow is also critical.

References

[1]: “Standardization of Group 3 facsimile apparatus for document transmission”, CCITT Draft Recommendation T.4
[2]: “XC6200 co-design for fax decoder”, D.M.Grant, Xilinx Scotland, 1997.
[3]: "Velab Release Notes", D.M.Grant, Xilinx Scotland, 1997

Limitations And Restrictions