XC6200 Fax Decoder Co-design

XAPP 086 July 22, 1997

Application Note by Douglas M Grant

Summary

This applications note describes the development of co-designs in software and hardware for a code decompression application. The target platform for the application is the XC6200 part within the XC6200DS development system - A PCI-interfaced, reconfigurable processing platform.

Xilinx Family

XC6200

Demonstrates

- Hardware/software co-design
- Reconfigurable Computing
- XC6000DS

Table of Contents

Introduction

This applications note describes the development of co-designs in software and hardware for a code decompression application. The target platform for the application is the XC6200 part within the XC62OODS development system - a PCI-interfaced, reconfigurable processing platform.

It is fast becoming practical to implement applications on FPGAs that were formerly only possible with ASIC technology. The benefits of using an FPGA include the low NRE, low risk and independence of shifting standards (future proofing). In the communications world especially, the accelerating improvements in bandwidth and worldwide 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, is likely to be improved to exploit the growing ISDN-2 communications infrastructure. There is already a draft for an update on the table. 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, reducing the size and cost of local, page buffer memory. Processing power and memory required to implement decompression in real time can be very expensive, and an FPGA-based accelerator as a reconfigurable processor can reduce these costs and make real-time operation possible.

The Development System

The XC6000 Development system comprises low-level software and a PC board containing an XC6216, an XC4013E to implement the PCI protocols and some other functions. This is a platform for implementing complete applications. 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 testing of the hardware 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 Make-up 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. The color of the data then alternates every one or two codes. A pixel run of one color is represented by either a single Termination code or by a 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 transmission of shorter codes for more common pixel data. 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

Co-design: Partitioning

There is a natural partitioning of the application into parts which software can do well: Control and sequencing, and parts which hardware can do best: Code detection, look-up tables (LUTs). Co-design may be envisaged as writing a program that "calls" hardware extracodes to execute the application. The hardware extracodes are ASIC designs that may be dynamically activated within the FPGA. As in all engineering problems there are a range of solutions that trade software complexity, hardware complexity and performance. The following sections describe two possible designs:

1. Using dynamically programmed decoding trees and discrete run-length LUTs.
2. Using a single code and run-length LUT.

Design #1

ASIC parts

The hardware in the first design consists of 3 groups of parts: I/O cells, decoding trees to detect the codes and LUTs to translate these to pixels [2].

I/O cells
Data is written by the software to a 32-bit, parallel-load shift register. Software drives the clock for the shift register via the PCI interface. This shift register forks after 32 bits into two 16-bit shift registers, one for feeding the black pixel decoder on the top half of the cell array, and the other feeding the white pixel decoder on the lower half. In the design there are separate Black and White decoding parts.

The outputs from the decoding trees are simple gates, and these signal the detection of a code from 2 to 13 bits and also the arrival of a FILL bit and of an EOL. The outputs from the LUTs are fed to columns of buffers in a single column that may also be read from. Two control registers in the design clear the shift registers (although this is not really necessary), and control the shift or load behavior of the 32-bit input shift register.

The hardware was described in VHDL and synthesized into an EDIF netlist. All the gates in the I/O blocks were fully pre-placed on the cell array by annotating the VHDL with RLOC attributes.

Decoding trees
The decoding trees are duplicated for White and Black data, two 5-bit decoding trees are placed back-to-back to make a 6-bit tree. With two 6-bit trees, codes of up to 12 bits may be detected (a few more gates detect the few 13-bit codes which are defined by the standard).

The trees are gate level designs with a single node taking three XC6216 cells. It is possible to fit a 5-bit tree into a 6 cell wide by 32 cell high bounding box. Cells are all pre-placed with RLOC attributes in the VHDL. The duplication of trees was aided by the REF9O transform, which reflects a design across a vertical axis.

It should be noted that a height of 32 cells for the trees was chosen purposely to reprogram the trees for detecting codes of length 7 to 13 bits. Depending on the values of the first 6 bits, this is most efficiently done with 32-bit writes to the array by the software.

The three cells in each node comprise a code detector, a register and an OR gate. The code detector is a single gate that outputs a '1' if the input data is of the correct value for that position in the tree. In this way the inputs "steer" a token along a single path through the tree. At certain points in the tree, at nodes which represent a valid code, an RPFDC primitive is used to store a '1' (which is written there by the software). This triggers a '1' to ripple up a chain of OR gates across the tree, to one of the "hit" outputs. The nodes are placed in an interleaved arrangement to reduce routing overhead, using a complex, parameterized RLOC attribute in the VHDL.

Run-length LUTs
The LUTs which translate the codes to pixel run-lengths were synthesized from PLA input format. Area-based optimizations were applied during synthesis. Timing driven structuring (the default) was applied first, followed by Boolean optimization.

Physical design

Figure 2 shows the layout of this hardware on a XC6216 FPGA, where the light-colored vertical stripes indicate I/O cells, the light-colored boxes surround the LUTs and the highlighted blocks of the other used cells form the eight 5-bit trees (or four 6-bit trees.)

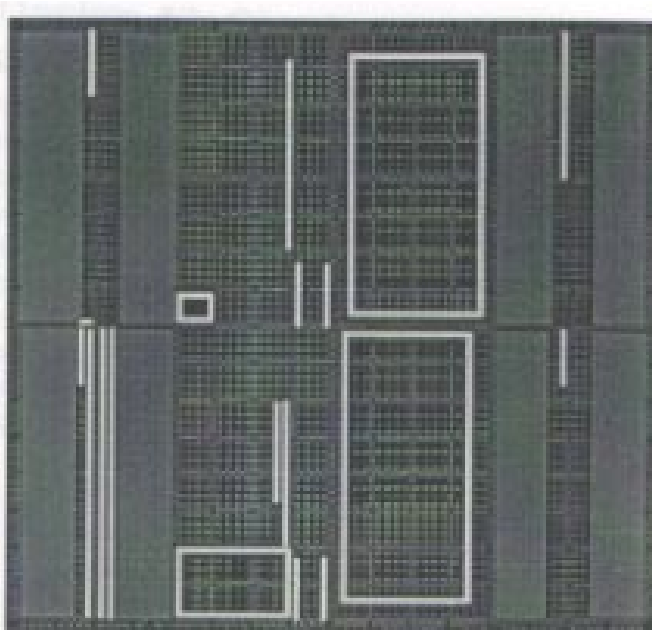

Figure 2: Layout of hardware on XC6216

Software part

Raw fax data is read from a file and stored in an array ready for processing, as in a wholly software-based approach. Data is written to the circuit 32 bits at a time, with software controlling this process. Processed data is read back and then output to file as characters, resulting in a 1728-character wide text file. The software flow is depicted in **Figure 3** and is described thus:

I. Read data from file.
II. Download the design to the XC6216.
III. Program the trees for detecting 2 to 6 bit codes (this only happens once).
IV. Set the multiplexer control to "load"
V. Write the first 32 bits of data to the column of RPFDCs
VI. Reset the mux control to "shift"

Writing the data generates a clock to the shift register to accept that data. By attaching a special primitive, the CBUF_OUT, it is possible to "drag" a control signal from the underlying logic of the FPGA. In this case that control signal is the word line which selects a column of cells for reading or writing. This signal is inverted and routed onto the global G1 clock network, which then clocks the shift register separately from the input registers.

VII. Repeat reading from the input register column, shifting the data into the two shift register forks, where it's ready to be decoded, which is done (almost) immediately.
VIII. Read from the appropriate base tree output to tell the software if a 2 to 6 bit code, a FILL bit or an EOL has been detected.
IX. If the software sees a '1' on the EOL output bit then a newline is written to the output file, and the data in the shift register is shifted on 12 bits. If a FILL bit is detected then a single shift happens. If however, a 2-to-6 bit “hit” is found, the translation to pixels has already been completed and the answer is waiting on the outputs of the appropriate LUT.
X. 6 bits of this result pass the values 0 to 63 and the 7th bit signals if this is a make-up code, rather than a termination code, which the software then multiplies by 64 to get the true number of pixels to print out.
XI. Then input data is shifted the appropriate number of bits.

XII. If a termination code is found then the software switches to decode the opposite color, on the other half of the IC.

If no code is detected on the base tree, the secondary tree is loaded with the valid codes given those first 6 bits. This is done by software.

XIII. Program the second tree with 4 more Writes, and immediately read the outputs of the second tree, and process that data accordingly:
XIV. Either no code will be detected, in which case there is a transmission error, or read the translated pixel data from the appropriate LUT output, as before.

The next code, which is waiting on the inputs of the trees and LUTs, may then be decoded. It may be necessary to write another 32 bits to the input registers and load these into the shift register. Software keeps track of how empty the hardware buffer is, and if necessary during the shifting process between code detections, the next word of data can be written.

This process of decoding continues until the data is all gone, or until an EOT is detected. Software again keeps track of how many consecutive EOLs have been received.

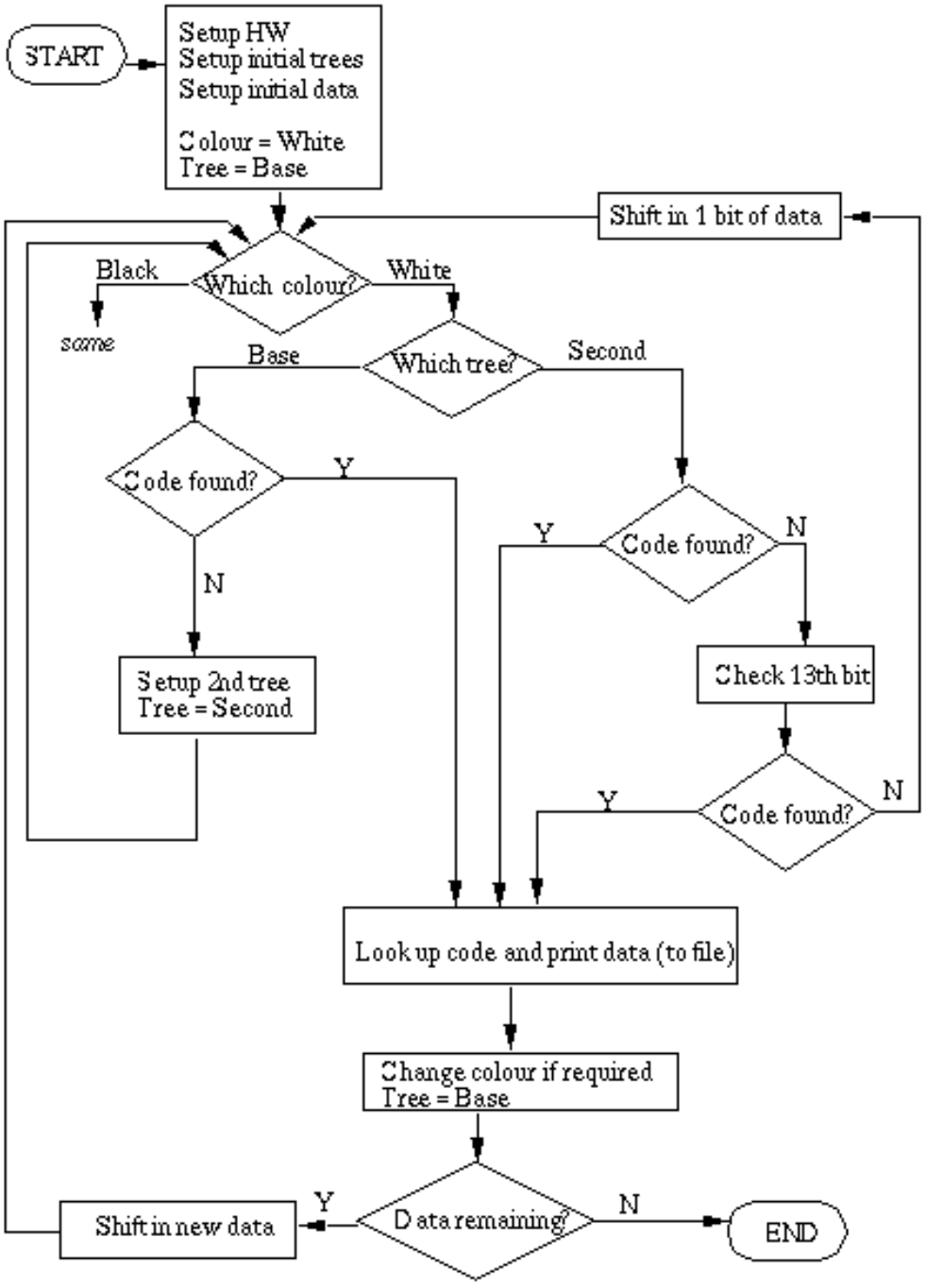

Figure 3: Software flow diagram.

Design #2

Single detection and run-length LUT

An alternative to this design is to follow a simpler approach and implement the whole detection and decoding phase as a single LUT. As well as translating codes to pixels runs, this LUT outputs how many bits were in any valid codes detected, and whether the codes were termination codes or make-up codes. It is more complex than the four LUTs in the previous design combined. There are two benefits in this approach:

1. It is no longer necessary to reprogram as much hardware during translation.
2. Conceptually very simple, this approach may appeal to software designers with little hardware design experience.

In **Figure 4** the LUT-based design takes the 13 bits from the shift register, plus a single extra bit which flags White or Black pixel data. It outputs the number of pixels, along with a bit to flag make-up codes and 4 bits telling the software how long the detected code was. The 14th bit of the LUT input is required since the Huffman codes are only unique within each pixel color, and not globally. The software needs to write to the register (in the bottom right corner) whenever the pixel color changes, i.e. after a termination code or an EOL code. The software that drives this design is much simpler, and does far fewer writes to the IC, saving many cycles.

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 is some four times better than a wholly software-based approach. Performance would be greatly enhanced if the XC6216 was directly addressed by the host processor rather than via the PCI bus. Burstmode access on the PCI bus can increase performance.

These designs expose the ease of hardware/software co-design for applications based on the XC6000DS can be used as templates for mapping more applications to a mixture of hardware and software based around the XC6000DS.

References

[1]: *"Standardization of Group 3 facsimile apparatus for document transmission"* - CCITT Draft Recommendation T.4
[2] *"A Fax decoder on the XC6216"* - D.M.Grant, Xilinx Scotland 1997.
[3]: "Reconfigurable processing - The 4th paradigm", University Video, San Jose, 1996.

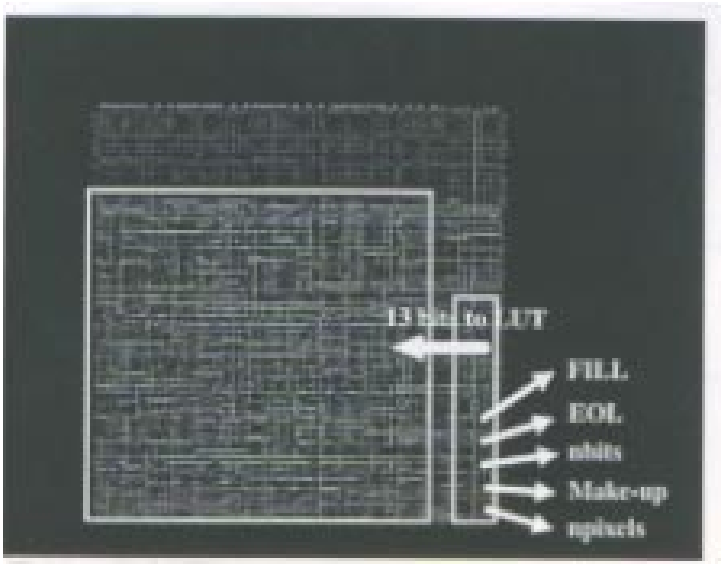

Figure 4: LUT-based design

Limitations And Restrictions

Warning: THIS IS AN UNTESTED DESIGN.

Xilinx, Inc. does not make any representation or warranty regarding this design or any item based on this design. Xilinx disclaims all express and implied warranties, including but not limited to the implied fitness of this design for a particular purpose and freedom from infringement. Without limiting the generality of the foregoing, Xilinx does not make any warranty of any kind that any item developed based on this design, or any portion of it, will not infringe any copyright, patent, trade secret or other intellectual property right of any person or entity in any country. It is the responsibility of the user to seek licenses for such intellectual property right where applicable. Xilinx shall not be liable for any damages arising out of or in connection with the use of the design including liability for lost profit, business interruption, or any other damages whatsoever.