Use of Reconfigurability in Variable-length Code Detection at Video Rates

Gordon Brebner[1] and John Gray[2]

[1] Department of Computer Science
University of Edinburgh
Mayfield Road
Edinburgh EH9 3JZ
Scotland
[2] Xilinx Development Corporation
52 Mortonhall Gate
Edinburgh EH16 6TJ
Scotland

Abstract. This paper shows how properties of the data produced by a particular compression scheme lead to an elegant paged logic implementation of a decoder. The implementation uses the new Xilinx XC6200 FPGA family, which supports dynamic use of programmable logic with special hardware. The work illuminates an eternal verity of computing that, when a resource is limited, its use can be extended by paging. In this case, the resource is logic.

1 The Group Three Fax Compression Scheme

In this work, properties of the data produced by a particular compression scheme are used to derive a paged logic implementation of a decoder. The compression scheme considered is the one used in the ITU-T (formerly CCITT) T.4 standard for group 3 fax transmission [3]. This combines run-length coding with Huffman coding [1]. The essence of Huffman coding is that it encodes symbols from an alphabet into binary codewords, with frequently occurring symbols having short codewords and less frequently occurring codewords having longer codewords. Given this, the encoding of a sequence of symbols is a binary string consisting of concatenated codewords. In the case of the ITU-T T.4 fax standard, the 'symbols' are the different lengths of alternating white and black runs in scan lines, and the binary codewords range in length from 2 to 13 bits. By this means, scan lines are encoded into bit streams that can be transmitted by fax apparatus.

The decoding problem is that codewords must be identified within a binary string. This is made possible by a key property of Huffman codes: no codeword is an initial substring of another. Thus, decoding can be performed by extracting codewords, in sequence, using a scan of the binary string from beginning to end. One method of detecting codewords in hardware is to allow the successive bits of the input to 'steer' a signal through a binary decoding tree, from the root to a leaf. The leaves of the tree represent codewords, and the arrival of a steered signal indicates the detection of a codeword in the input stream. Figure 1 shows a simple decoding tree for a code with the codewords **00**, **1000**, **1001** and **101**. Since decoding trees grow exponentially with the maximum size of codeword, there is usually a severe limitation to the size of a tree that can be built in fixed hardware or using PAL technology. However, if the hardware can be reprogrammed dynamically, it is possible to extend the steered signal through an arbitrarily large tree.

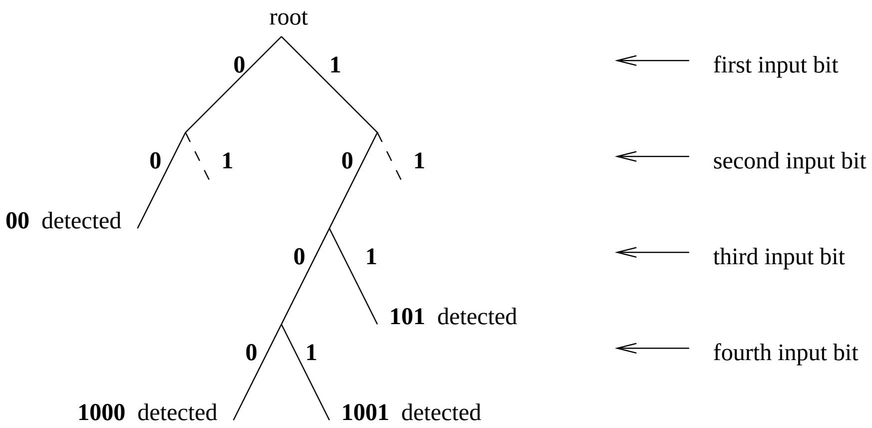

Fig. 1. Simple Decoding Tree

The systems context for designs of this type is usually a memory to memory transfer of data as compressed data is expanded. In this example, compressed fax data is being expanded for printing or bitmap manipulation. Binary strings from memory are decoded to give lengths of white and black runs in scan lines, which are then used to construct a bitmap.

2 Properties of Input Data

Group 3 fax apparatus operates at a normal resolution of 200 dots per inch. To accommodate paper of A4 width, a horizontal scan line contains 1728 pixels. Each scan line is usually handled independently, and consists of alternating runs of white and black pixels. The transmitted data for a scan line is an encoding of the sequence of run lengths. Since certain run lengths occur far more frequently than others, the T.4 fax standard defines two sets of variable-length codewords for the integers between 0 and 1728 (the possible run lengths), one set for white and a different one for black.

The codewords chosen in the T.4 standard were based on an analysis of 'typical documents'. Frequently observed run lengths were allocated short binary codewords, whereas less frequent lengths were given longer binary codewords. The allocation procedure used Huffman's algorithm to select codewords based on occurrence frequencies, in a way that ensures no codeword is an initial substring of any other. To illustrate the code, Table 1 shows the encodings of the first few run lengths, from 0 up to 10.

White run length	Codeword	Black run length	Codeword
0	00110101	0	0000110111
1	000111	1	010
2	0111	2	11
3	1000	3	10
4	1011	4	011
5	1100	5	0011
6	1110	6	0010
7	1111	7	00011
8	10011	8	000101
9	10100	9	000100
10	00111	10	0000100

Table 1. Codewords for Run Lengths 0 to 10

Since the data used by ITU-T to determine the choice of codewords was not readily available, experiments were performed to measure the frequency of different run lengths. Two types of 'typical document' were chosen: pages taken from a technical report, and pages taken from letters. The former had a higher density of black, and a more regular distribution of black. The latter differed from this because of more spacing out and the presence of logos etc. It was found that the distribution of run lengths for both white and black, for both types of document, was a reasonable match to the distribution expected from examining the code chosen by ITU-T. Table 2 summarises the distribution of *codeword lengths* that resulted when the test pages were encoded.

Codeword	White		Black	
length	Reports	Letters	Reports	Letters
2	—	—	54.8%	48.4%
3	—	—	17.4%	20.1%
4	59.1%	41.8%	11.8%	10.9%
5	13.0%	15.9%	3.0%	3.0%
6	13.3%	14.8%	4.7%	4.8%
7	4.7%	7.9%	5.2%	4.8%
8	8.1%	14.4%	1.3%	2.5%
> 8	1.8%	5.2%	1.8%	5.5%

Table 2. Distribution of Codeword Lengths for Test Data

From the table, it can be seen that the pages taken from the technical report have a shorter expected codeword length for both white and black. At least 85% of codewords have length at most six, and at least 90% of codewords have length at most seven. The same statistics for the pages taken from letters are not so impressive, being 72% and 80% respectively. This reflects the fact that these pages are probably 'less typical' (i.e., have a more random structure) than the pages used to derive the Huffman code. It is clear that there can be no single best code for all fax data. However, the codeword length frequencies obtained experimentally here give an indication of what is achievable in a realistic setting.

3 Architecture of Design

3.1 Systems Level

The overall architecture of the decoding system is shown in Figure 2. The idea is that there is a standard unit tree that can detect codewords of size up to a particular length. One or more 'fixed trees' are permanently resident in the FPGA, and other 'swap trees' are paged into the FPGA from primary memory as necessary, under the control of the processor.

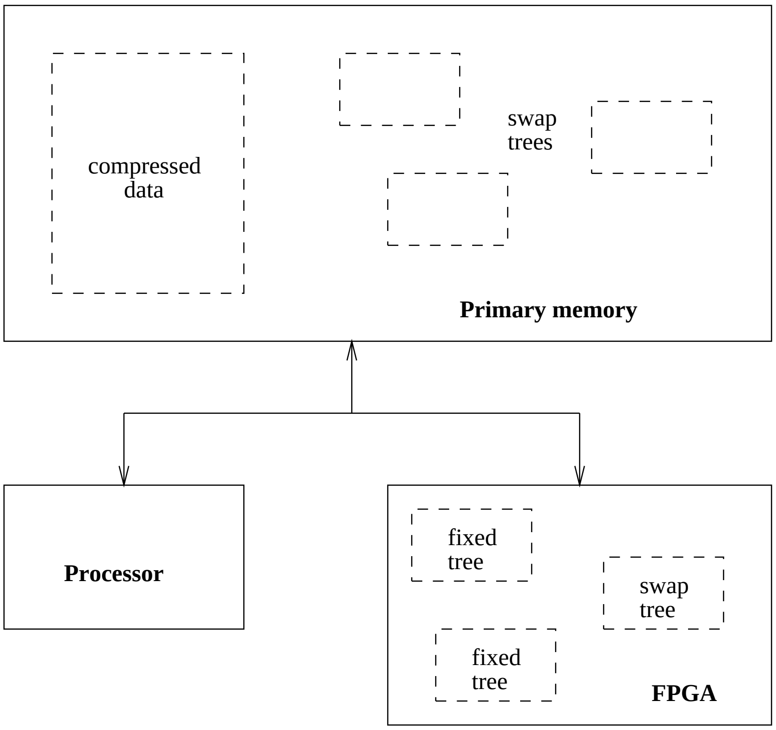

Fig. 2. Decoding System Architecture

When decoding is in progress, one of the resident trees is active. If it completes the detection of a codeword, a success indication is given to the processor. Otherwise, an "incomplete" indication is given to the processor. Then, the processor may either complete the codeword detection by using a table stored in memory, or swap in the next (lower) sub-tree of the decoding tree.

There are two key design decisions at this level. The first concerns the optimum size of a unit tree. The statistics included in the previous section point to possible answers for this question. For example, for textual documents, a six-bit decoding tree would detect approximately 85% of codewords directly. In fact, this tree size was used for the design described here, since it represented a reasonable compromise between having a high chance of successfully decoding within a single unit tree, and having moderate overheads when swapping trees. A further advantage is that the end-of-line code, which is a guaranteed feature of each scan line, is 12 bits long, and so can be detected using exactly two six-level trees.

The second decision concerns the number and arrangement of fixed trees included in a complete system. Two fixed trees — one for white, the other for black — are likely to be present, otherwise swapping is necessary for each run of white or black. In the case of black runs, all except four codewords begin with two zero bits. If the four special cases are detected separately, then a fixed six-bit tree can be used to detect all codewords of length up to eight bits. In the case of white runs, the decoding tree is significantly more balanced, so this approach is not fruitful. However, an alternative possibility is to include a pair of fixed trees, one for each of the two top-level sub-trees of the decoding tree. This idea could be further extended to having four six-bit sub-trees, which would allow detection of all codewords of length up to eight bits.

The final design decisions are driven by the performance specification of the whole system, and the resources available on the FPGA. Here, the prototype design was targeted at the XC6215 part, which offers a 64×64 cell array on one chip. Since a six-bit unit tree would fit into a 32×21 cell array, this allowed four unit trees to be present simultaneously on a single chip, together with any necessary glue logic. Therefore, using the above ideas, three fixed trees for decoding all codewords of length up to seven bits, together with one swap tree for longer lengths, could be comfortably accommodated. With a tighter design, it is likely that five, or even six, unit trees could be accommodated, depending on the amount of extra logic needed for interfacing.

3.2 Logic Level

To accommodate word-size transfers between the processor and the FPGA, a back to back arrangement of two five-bit decoding trees was used to construct a six-bit unit tree. Each five-bit tree is one of the two top-level sub-trees of the unit tree.

The XC6200 family allows direct register access to the FPGA. Every cell may be configured to contain both a function unit and a flip-flop, if required. The output of any cell's function unit can be read, and the flip-flop within any cell can be written. Registers can be configured within each column of the array, with the cells forming an 8-bit, 16-bit or 32-bit register selected using a map register for the column. To exploit this capability in the design here, a 32-bit shiftable I/O register was inserted between the five-bit decoding trees to supply the input bit stream and to collect codeword detection information. This register eliminates the need for wiring to the I/O pins of the array, at no performance penalty. The general layout is illustrated in Figure 3.

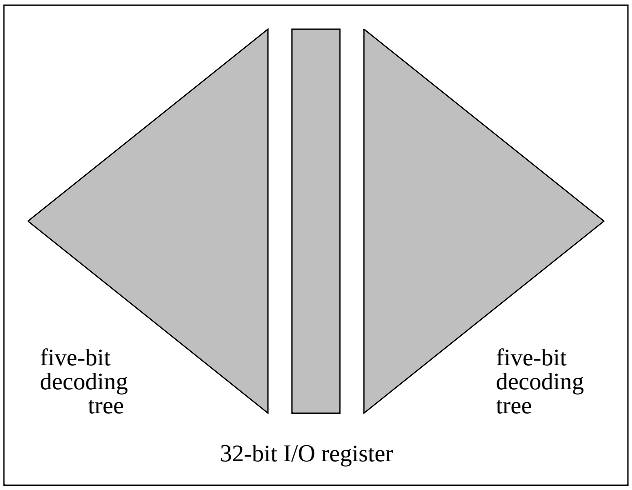

Fig. 3. Six-bit Unit Tree Layout

The input bit stream is loaded, one byte at a time, into 24 bits of the I/O register. The decoding sub-trees are driven from this register, the choice of sub-tree being made by the first bit, and then the steering at each of the five levels of the chosen sub-tree being driven by the next five bits. The decoding sub-trees have a simple B-tree form that interconnects detector cells. A dense layout is possible for an XC6200 series FPGA, given the assistance of the associated design tools, in particular the Camelot mapping, placement and routing software. The minimum size layout for a unit tree occupies a 32×15 cell area; a layout with better performance occupies a 32×21 cell area.

The detector cells represent nodes in the tree, and embody structural information: (a) whether the incoming tree edge is labelled **0** or **1**; and (b) whether the node is a leaf. This information may be 'soft-wired' for minimum area and maximum detection performance (but lower swap performance). Alternatively, the information may be register written, to maximise swap performance, at some space and detection performance overhead. A schematic diagram of a detector cell is shown in Figure 4. When a steering signal reaches a detector cell, and the detector's input bit matches its incoming tree edge label, the cell is activated. If it is a leaf cell, a 'codeword detect' signal is output; detect signals at each level of the tree are collected together using an OR chain. Otherwise, a steering signal is output to the next tree level, if there is one.

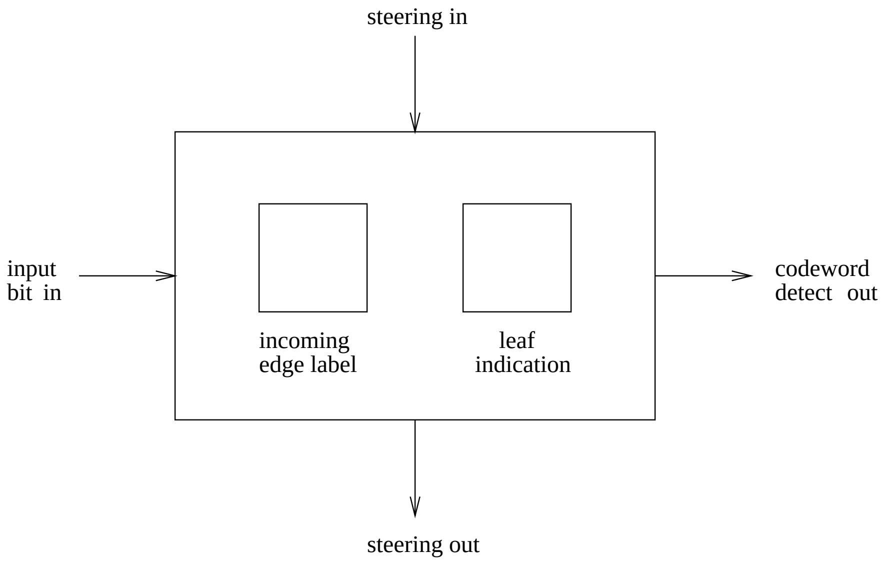

Fig. 4. Detector Cell Schematic

The I/O register contains a six-bit output from the codeword detection circuitry, each bit used to indicate a codeword detection at one of the six levels of the tree. If all six bits are zero, then the detection is incomplete. In this case, the first six bits in the input part of the I/O register indicate the appropriate next stage for the decoding process, whether this is an index of a swap tree or a pointer to a lookup table. When a codeword (or part of a codeword) has been detected and the output read by the processor, the input part of the I/O register is shifted by the appropriate amount using additional circuitry, and is replenished with further data bytes by the processor when required.

The current design does not detect invalid codes. However, this feature could be obtained easily by adding the generation of a 'code error' signal output to existing leaf cells which correspond to the detection of bit strings that are not prefixes of valid codewords.

3.3 Performance Considerations

With this architecture, performance when detecting a codeword of length at most six bits is determined by two factors. First, the number of gate delays before detecting the codeword, which will be in the range one to six for a single unit. Second, the time required to 'carry' a detect signal out of a level of the decoding tree, which will be 1 to either 15 or 31 gate delays, depending on wiring strategy. Therefore, codeword detection times would be in the range 20 to 70 ns for typical gate delays.

For codewords with length greater than six bits, one of 64 sub-trees must be swapped in. For the XC6200 FPGA family, register writes to the FPGA are executed at SRAM speeds. If detection cells are hard-wired to give a minimum space solution, 30 writes are required but, if detection cells are register written, only 10 writes are needed. At first sight, these figures suggest that the paged tree solution might be slower than a straightforward software solution using a lookup table. However, the fact to remember is that the paging scheme has good locality. For example, if 90% of codewords are detected by a fixed tree, then the cost of swapping in a sub-tree can be amortised: 10 writes incurred for a page fault would average out to one write per codeword.

Given the above remarks, a reasonable estimate for the expected detection time for a codeword is 150 ns. Since each codeword represents a run length, the final output rate of the overall decoding system, measured in bits per second, depends on the length. The experimental data, for report pages and letter pages, had average white run lengths of 46 and 121 respectively, and average black run lengths of 4.5 and 5.3 respectively. Thus, expected bit output rates would be 30 Mbps for black, and at least an order of magnitude more for white. These are above video rates, and are substantially better than the rates obtainable from finely-tuned software implementations,

4 Exploitation of Reconfigurability

The discussion in the previous section has shown that there is significant scope to exploit configurability in order to achieve the best performance from the fax decoding system. This can be summarised under three headings:

Logic Level

At the logic level, there is the option of configuring detection cells either by 'soft wiring' or by register writing. The former saves area, and gives improved performance. The latter gives improved swap performance.

Systems Level

At the system level, there are various options for choosing the total number of unit trees that are simultaneously resident, and then for choosing the balance between fixed trees and swap trees. A further consideration is the choice of unit tree size. These options involve trade-offs between area and paging behaviour.

Processor Level

At the overall system level, there is a basic trade-off between using table lookup and using the detection circuitry. At one extreme, there is the conventional software decoding approach; at the other extreme, there is an approach that uses the circuitry exclusively. The middle option is the use of the detection circuitry as a first stage, then the use of lookup tables to avoid logic paging.

The previous section gives timing estimates derived under the assumptions of a 2 ns gate delay, and a register write time of 40 ns per transfer. The latter figure depends on the efficiency of the FPGA control store performance. With the imminent availability of XC6200 parts, it will be possible to test these timings experimentally. This will also allow investigation of the various logic level and systems level trade-offs.

The fundamental question arises from the processor level trade-off. That is, whether logic paging can be more efficient than just handling infrequent cases in software. As the previous section indicates, the cost of handling a page fault can be as little as 10 memory writes. The circuitry can then complete the lookup in the time that would be needed for only one or two memory operations. This total time should be reasonable compared with a software solution, when instruction fetches, bit-twiddling and table lookups are taken into account. However, the full benefit might only be derived if paging did not require the intervention of the processor, by using DMA transfers. Whether or not this is done, some processor time is freed for other, more general purpose, work. Only experimentation, with the circuitry embedded in a realistic overall system will allow verification of these estimates. However, the expectation is that higher continuous rates of throughput will be possible, certainly compared with total software implementations, and also compared with hybrid implementations.

5 Conclusions

The work described in this paper indicates that reconfigurability can be exploited in several ways for an application with suitable properties. In this case, the major properties are that the detection circuitry has a regular (tree-like) structure, and that probabilities can be derived for the use made of different parts of the circuitry. This forms the classic basis for effective paging. Some of the trade-offs possible are exactly those found in the management of virtual memory by operating systems. Other, new, trade-offs arise from the fact that the hardware itself can be changed.

The issues raised in this paper apply in other circumstances. The total approach is clearly applicable to any Huffman decoding circuitry, and so would be of use for applications where very high speed operation is more significant than it is between group 3 fax apparatus. One particularly interesting application is to dynamic Huffman codes, where individual compressed files (or other units of data) carry details of their tailored Huffman code as extra information. Assuming that the tailored code guarantees short codewords with high probability (and the Huffman coding algorithm does guarantee the smallest expected codeword length possible), then the configurable hardware will perform well.

In general, the approach is applicable to any tree-structured circuit with predictable localised behaviour. By allowing customisation of the function in each tree cell, it is possible to envisage a general-purpose paged tree system. Further, the ideas are applicable to other types of planar graph. The most obvious such type of graph is the two-dimensional array, where rectangles form a natural paging unit. Two examples of this style of reconfiguration, used for artificial neural network implementation, can be found in [2] and [5]. An example of gross paging of an entire array, to give a multifunction progammable logic device, can be found in [4].

The interaction between the configurable hardware and the processor is also very important, since it takes place only via registers that are read and written by the processor. No additional wiring is required to interface the hardware, which is a major advantage in simplifying the management of FPGA real estate. The overall scheme brings configurable hardware into the mainstream of computing, and points the way to effective and flexible acceleration of software in the future.

References

1. Cormen, Leiserson and Rivest, *Introduction to Algorithms,* MIT Press 1990, pp.337–344.
2. Eldredge and Hutchings, "RRANN: A HArdware Implementation of the Back-propagation Algorithm Using Reconfigurable FPGAs," Proc. IEEE International Conference on Neural Networks, 1994.
3. International Telecommunication Union, "ITU-T Recommendation T.4: Standardization of Group 3 Facsimile Apparatus for Document Transmission," ITU Geneva 1993.
4. Ling and Amano, "WASMII: a Data Driven Computer on a Virtual Hardware," Proc. IEEE Workshop on FPGAs for Custom Computing Machines, 1993.
5. Lysaght, Stockwood, Law and Girma, "Artificial Neural Network Implementation on a Fine-grained FPGA," Proc. 4th International Workshop on Field-Programmable Logic and Applications, 1994.