FPGA implementation of 1D wave equation for real-time audio synthesis

Article *in* IEE Proceedings - Computers and Digital Techniques · October 2005

DOI: 10.1049/ip-cdt:20045178 · Source: IEEE Xplore

David M Howard
Royal Holloway, University of London
384 PUBLICATIONS **2,368** CITATIONS
SEE PROFILE

Andy Tyrrell
The University of York
203 PUBLICATIONS **2,522** CITATIONS
SEE PROFILE

The IEE

Engineering the future

IEE Proceedings

Computers & Digital Techniques

Volume 152 Number 5 September 2005

ISSN 1350-2387

ICDTEA 152 (5) 537–696

Contents

FPGA implementation of 1D wave equation for real-time audio synthesis

J.A. Gibbons, D.M. Howard and A.M. Tyrrell

Abstract: The paper addresses the potential benefits of using a field programmable gate array (FPGA) as opposed to a traditional processor for music synthesis. The benefits result from the use of a cellular design, with each cell performing identical operations on its own state and the states of its neighbours. This gives advantages of design simplicity through inherent parallelism. A cellular model which has previously been used for music synthesis is the mass spring paradigm, and this model is implemented on an FPGA. On a sequential processor, the clock speed requirements of this model increase as N^2 (where N is the number of cells), whereas the clock speed requirement increases as N on an FPGA (if the cells can be made small enough that available area on the chip is not a constraint). To make the cells small enough, a bit-serial design was used. This work was performed to advocate the FPGA as a stand-alone live performance music synthesis platform, which has the potential to be reconfigured for example in performance, between synthesis techniques as different sound patches are selected. The paper considers in particular the mass-spring model, because it demonstrates the type of music synthesis technique for which FPGAs provide the greatest potential benefit. The capability to reconfigure combined with the high performance that can be achieved using cellular design suggests that the FPGA is an ideal platform for a live performance hardware music synthesiser, combining the flexibility of software with the speed of a custom ASIC.

1 Introduction

An FPGA can be considered a compromise between the flexibility of software and the processing speed of a custom ASIC. The FPGA device is configured before run-time, and whilst this configuration can be considered analogous to software for a sequential processor, practically, it is a circuit design. Since the FPGA is programmed in this manner, designs placed on it are limited only by the size of the device itself in terms of the number of operations that can be performed in parallel. FPGAs therefore perform best when configured by algorithms that can be easily designed as a parallel circuit. One method of achieving this is through cellular design. If a useful algorithm can be designed as an array of homogenous cells, then the design process requires the design of just one cell. This cell is then replicated to create a homogenous array from which the function of the algorithm is an emergent property. In the world of music synthesis, the mass-spring algorithm represents a good candidate for this approach.

The roots of cellular implementations can be found in the work of Von Neumann [1] who devised the cellular automaton. A cellular automaton is a device that has an internal state and determines its next state from its current state and the state of its local neighbours. Wolfram [2] defines eight basic defining characteristics of cellular automata. These are listed below, along with an explanation of why the cells presented herein conform to the specifications:

1. *Discrete in space:* The model calculates displacements at discrete points of a string.
2. *Discrete in time:* The displacements are calculated at discrete time intervals.
3. *Discrete states:* Each cell contains a finite number of registers and as such is still limited to a finite number of states.
4. *Homogenous:* Every cell in the design is identical.
5. *Synchronous updating*: Every cell updates simultaneously on an FPGA.
6. *Deterministic rule:* The rule that updates each cell is deterministic and constant.
7. *Spatially local rule:* Each cell calculates its next state from its own state and those of its neighbours. However, the cells may be affected by an external input.
8. *Temporally local rule:* The rules to update the cells depend on values from the previous time step.

This design matches every defining characteristic of a cellular automata array, yet it is a direct discrete implementation of the wave equation. Wolfram [2] suggests that finite difference models (from which the mass-spring model is derived in Section 2.1) differ from cellular automata because the site values are continuous. However, the displacement values used here are clearly bounded and discrete.

The work described in this paper forms part of a larger project in which a novel run-time reconfigurable fault tolerant FPGA architecture is being designed. This new architecture is a self-contained, flexible and physical substrate designed to interact with the environment through spatially distributed sensors and actuators to develop and dynamically adapt its functionality through the processes of

IEE Proceedings online no. 20045178
doi: 10.1049/ip-cdt:20045178
Paper first received 25th June 2004 and in revised form 21st January 2005

The authors are with the Department of Electronics, University of York, York, UK
E-mail: amt@ohm.york.ac.uk

evolution and growth to a dynamic and partially unpredictable environment, and to self-repair parts damaged by ageing or environmental factors in order to remain viable and perform the same functionality. This device is to use a cellular architecture, with strong metaphors taken from biological systems (e.g. biological cellular development). Such biologically inspired designs require each electronic cell, within the array, to hold appropriate information regarding the construction and functionality of the system (as for DNA). These requirements, while giving the overall system extra reliability through self-repair and self-reconfiguration, also put constraints on the functionality available in each electronic cell in an array, and minimising designs has considerable advantages. For the work described in this paper, this necessitated the use of bit-serial design. Since the device is to be a cellular parallel processor, the cellular nature of the design presented here was considered advantageous. We show in this paper that such minimal designs can be achieved for such architectural models, in this case FPGAs, without loss of overall functionality.

The application under consideration here is real-time music sound synthesis. Reconfiguration offers the potential for a host of algorithms to be offered when a music performer wants to change the sound synthesis algorithm being used, for example for a new movement of a piece, or when there is a need to change the timbre rapidly (for example, through context switching). Currently, this would typically involve switching between separate music synthesis systems. There are a number of different techniques that are used for music synthesis, including additive synthesis, subtractive synthesis, sampling synthesis, frequency modulation (FM) synthesis, wavetable synthesis and physical modelling. Each technique makes use of a number of signal processing elements, many of which are different. Contemporary approaches to music synthesis are often capable of implementing more than one of these techniques by means of a library consisting of the individual signal processing elements, each generally referred to as a 'unit generator process' or 'UGP' (e.g. Dodge and Jerse [3]), which can be interconnected as required. The specific process to be considered in this paper is physical modelling synthesis, which is a popular technique that produces sounds of interest to today's performers, since they are often considered to be quite natural sounding or 'organic'.

2 Algorithm

Many implementations exist for modelling the wave equation for music synthesis. The most popular is the digital waveguide by Smith [4]. It is a distributed model that uses delay lines to propagate the travelling waves of the D'Alembert solution to the wave equation. The length of the delay line is proportional to the frequency produced at the output. This model has properties of massively reduced computational requirements compared to lumped models when the algorithm is implemented on a sequential processor. However, an FPGA can only provide a processing speed-up through parallelism since these devices tend to operate at lower clock speeds than dedicated sequential processors (e.g. an Intel Pentium) using equivalent process technologies. Delay lines cannot be made parallel, so an FPGA can only improve the performance of a waveguide by executing many of them simultaneously. For example, a guitar has six strings and the FPGA could process all six at the same time. If multiple waveguides are not required then an FPGA cannot provide any speed-up over a traditional CPU. Lumped models are more appropriate for an FPGA, since their implementation of a single string is cellular and can be processed in parallel. They have further benefits of design simplicity, since only one cell needs to be designed. A waveguide would require dedicated filters for accurate frequency selection and damping. Secondly, the model presented here can easily be operated in any number of dimensions, while a waveguide is limited to one. The Cymatic software by Rimmel *et al.* [5] uses this model to create virtual musical instruments that could not exist in reality, such as an instrument that occupies four dimensions.

Hiller and Ruiz [6, 7] show how the model can be extended such that it accounts for stiffness and internal friction. The model used here accounts only for viscous damping, following the work of Rimmel and co-workers[5, 8, 9]. However, the design of a single cell can be extended to account for these other factors. This shows that the simplicity of cellular design allows modelling of complex systems for which designing a waveguide would be very difficult.

Lumped models can be divided into those that model masses and springs and those that use finite differences, both of which are described herein. The following derivation is used to make predictions of the output frequencies produced by different inputs presented to the model. This is used to show how to design the model to meet a specification of 'the maximum allowable difference in frequency produced by two numerically adjacent inputs to the model'. This is important for music synthesis, since it must be less than the just noticeable difference (JND) for pitch across all audible frequencies to enable the design to be used in a musical context. This enables a design to be created that is not audibly different in terms of frequency selectivity to a floating point solution that would be implemented on a traditional processor. The output fundamental frequency predictions are compared to those obtained from the FPGA in Section 5 to prove that the model operates correctly.

2.1 Derivation of model using finite differences on wave equation

The wave equation (1) states that the acceleration of a string at a point equals the curvature of the string at that point multiplied by the square of the speed at which waves propagate along the string:

$$\frac{\partial^2 y(x,t)}{\partial t^2} = c^2 \frac{\partial^2 y(x,t)}{\partial x^2} \tag{1}$$

$y(x,t)$ is the displacement in the y-axis at location x and time t, x is the location in the x-axis, t is time and c is wave propagation velocity.

This equation does not allow real-time input and is not damped, so its output will not decay with time. To make the sound decay, a term to represent a damping force due to atmospheric viscosity is included. This is due to Hiller and Ruiz [6, 7]. A function is added which represents the real-time input and is therefore known for all x and t. The new governing equation is given by (2):

$$\frac{\partial^2 y(x,t)}{\partial t^2} + d\frac{\partial y(x,t)}{\partial t} = c^2 \frac{\partial^2 y(x,t)}{\partial x^2} + p(x,t) \tag{2}$$

$p(x,t)$ is the input from the user at location x and time t and d is the level of damping.

To model the string digitally, time and space are made discrete. Therefore ∂t and ∂x have tangible values rather than being infinitesimally small. The model is described as 'lumped' for this reason. In this model, the string is represented as 32 'cells'. Each cell contains the displacement and velocity of an infinitesimally small section of the string, separated from its neighbouring cells by ∂x. The displacement of the string is only known at multiples of ∂x. Its value is given by (3). The ends of the string have zero displacement (they are hinged) and are not considered to be cells:

$$\partial x = \frac{L}{(N+1)} \tag{3}$$

where N is the number of cells to be used in the model and L is the length of the string.

The term ∂t is related to the sampling frequency, f_s as shown by (4). Because the model is discrete in time, its value is only known at multiples of ∂t,

$$\partial t = \frac{1}{f_s} \tag{4}$$

where f_s is the sampling frequency.

To express the fact that x and t are only known for multiples of ∂x and ∂t, we can write (5),

$$x = l\partial x \quad t = n\partial t \quad n = l = 0, 1, 2, 3, \ldots \tag{5}$$

Equation (6) uses n and l rather than x and t to show that it is only valid for multiples of ∂t and ∂x. The velocity value used to calculate the damping is found at an earlier time so that future values will not be required to calculate it when the finite difference approximation is applied. This is called a 'half sample delay error' by [10]

$$\frac{\partial^2 y(l,n)}{\partial t^2} + d\frac{\partial y(l,(n-0.5))}{\partial t} = c^2\frac{\partial^2 y(l,n)}{\partial x^2} + p(l,n) \tag{6}$$

Applying the finite difference approximation to each term (apart from p, which is known) gives (7)–(9). This use of finite differences is due to [10]:

$$\frac{\partial^2 y(l,n)}{\partial t^2} = \frac{y(l,n-1) - 2y(l,n) + y(l,n+1)}{\partial t^2} \tag{7}$$

$$d\frac{\partial y(l,(n-0.5))}{\partial t} = d\frac{y(l,n) - y(l,n-1)}{\partial t} \tag{8}$$

$$c^2\frac{\partial^2 y(l,n)}{\partial x^2} = c^2\frac{y(l-1,n) - 2y(l,n) + y(l+1,n)}{\partial x^2} \tag{9}$$

Substituting (7)–(9) into (6) gives the finite difference solution shown by (10). It will now be used to derive the mass-spring solution,

$$y(l,n+1) = y(l,n) + (1 - d\partial t)(y(l,n) - y(l,n-1)) + \frac{c^2\partial x^2}{\partial t^2}\begin{pmatrix} y(l-1,n) \\ -2y(l,n) \\ +y(l+1,n) \end{pmatrix} + p(l,n) \tag{10}$$

By rearranging (8) and rewriting velocity as $v(l,(n-0.5))$ rather than $\partial y(l,(n-0.5))/\partial t$, we obtain (11),

$$v(l,n-0.5)\partial t = y(l,n) - y(l,n-1) \tag{11}$$

$v(l,n)$ is the velocity at time n and position l.

By the same method, (12) can be found:

$$v(l,n+0.5)\partial t = y(l,n+1) - y(l,n) \tag{12}$$

The first of the mass-spring equation (13) can be found by substituting (10) into (12). The second mass-spring equation (14) is derived by substituting (13) into (10):

$$v(l,n+0.5)\partial t = (1 - d\,\partial t)v(l,n-0.5)\,\partial t + \frac{c^2\partial x^2}{\partial t^2} \times \begin{pmatrix} y(l-1,n) \\ -2y(l,n) \\ +y(l+1,n) \end{pmatrix} + p(l,n) \tag{13}$$

$$y(l,n+1) = y(l,n) + v(l,n+0.5)\partial t \tag{14}$$

These equations are similar to the mass-spring paradigm used by Rimmel and co-workers [5, 8, 9] and Pearson and Howard [11]. The mass-spring model iteratively performs numerical integration, and as such can be written in a more compact form given by (15):

$$y(l,n+1) = \sum_{i=0}^{n}\sum_{j=0}^{i}\left((1-d\partial t)^{i-j}\left(\frac{c^2\partial x^2}{\partial t^2}\begin{pmatrix} y(l-1,j) \\ -2y(l,j) \\ +y(l+1,j) \end{pmatrix} + p(l,j)\right)\right) \tag{15}$$

This shows that the mass-spring and finite difference iterative solutions are both simply iterative methods of performing numerical integration at a set of equally spaced points. The mass-spring model was chosen for implementation in hardware since it has been used previously for music synthesis [5, 8, 9, 11]. Previous work has already implemented the finite difference method in hardware [12–14] to accelerate the modelling of electromagnetic phenomena. The architecture used for this work is very different from that described here. The design by Schneider *et al.* [12] is very similar to the wave digital filter described by Bilbao and Smith [15], which is based on the bilinear transform [16]. It benefits from better stability against overflow errors than the model used here. The downside is that it requires two multipliers per cell. The multiplier is by far the largest component in this design. Since minimisation of the size of a cell is one of the goals of this work, the mass-spring model was chosen. Smaller cells allow more to be placed on the FPGA and hence more complex instruments to be designed. Durbano *et al.* [14] store data in off-chip ram and implement floating point 'solvers' on the FPGA, which is in marked contrast to the implementation presented here.

2.2 Selecting sampling frequency

The wave equation is now in a form that can be solved iteratively. The size of ∂t required such that the string can produce all audible frequencies must now be found. The range of human hearing is approximately 20 Hz to 20 kHz. The pitch produced by the string is controlled by varying c, the speed at which waves may propagate along its length. The Courant–Friedrichs–Lewy stability condition states that the 'domain of dependence of the numerical method must include the physical domain of dependence' [17]. This implies (16), given by [17]:

$$c \le \frac{\partial x}{\partial t} \tag{16}$$

This is because the model is not continuous in time and space, such that the maximum distance information can travel in time ∂t is ∂x. If c is set such that it exceeds this, then information will be lost and the string cannot sustain a fundamental frequency (this is explained in terms of the Courant limit by [12]). Therefore, we can evaluate the maximum speed that waves can propagate along the string (equation 17) in terms of the number of cells, the length of the string and the fundamental frequency by substituting (3) and (4) into (16):

$$c_{\max} = \frac{Lf_s}{(N+1)} \tag{17}$$

where $c_{\max}$ is the maximum propagation velocity of waves on the string.

The sound that is produced by plucking the string is dominated by its fundamental frequency of oscillation, f_0, which is found using (18),

$$f_0 = \frac{c}{2L} \tag{18}$$

$f_0$is fundamental frequency.

Therefore, the maximum output frequency occurs when the propagation velocity is set to its maximum, giving (19),

$$f_{0_{\max}} = \frac{f_s}{2(N+1)} \tag{19}$$

where $f_{0_{\max}}$ is maximum fundamental frequency.

This shows that increasing the number of cells reduces the maximum fundamental frequency.

2.3 Optimising model for digital hardware

The term $c^2\partial t^2/\partial x^2$ in (13) is a user controlled input that sets the fundamental frequency. Its minimum value is zero, which gives a fundamental frequency of zero. It cannot exceed one and still obey the Courant–Friedrichs–Lewy stability condition. This can be understood by substituting (16) into the term. By substituting (3), (4) and (18) into this term, it can be written in terms of the sampling frequency, the fundamental frequency and the number of cells. This is shown by (20),

$$0 \leq \left(\frac{4(N+1)^2 f_0^2}{f_s^2} = \frac{c^2\partial t^2}{\partial x^2}\right) \leq 1 \tag{20}$$

For efficient implementation in digital hardware, integer arithmetic is used. Floating point multipliers and adders are too large to be implemented in a cellular manner, and as such the parallelism and simplicity of the design would be lost. For this reason, the input to select f_0 must be an integer. Secondly, numbers may only be divided by a power of two (a shift right operation), since integer division uses a lot of hardware and is difficult to pipeline since it is most efficiently performed MSB first. Therefore, to multiply a number by a value between 0 and 1, one must first divide it by a power of two and then multiply it by an integer, as shown by (21):

$$\frac{4(N+1)^2 f_0^2}{f_s^2} = \frac{i}{2^b} \quad 0 \leq i \leq 2^b \tag{21}$$

where i is the input to the model and must be an integer, and b is the number of bits by which the displacement must be shifted such that $c^2\partial t^2/\partial x^2$ does not exceed 1.

In this implementation, the damping coefficient, $d.\partial t$ is allowed to have eight different levels; the first is 0, which corresponds to no damping. The next levels are powers of two that were selected through experimental trials. The 'no damping' level allows a note to be sustained indefinitely. The next level causes the notes to decay such that they are inaudible after approximately 6 s. Notes last approximately 3 s for the next level, and decrease with each further level until complete decay occurs almost instantaneously as the string is struck. The use of levels was chosen because it enables the implementation to use a multiplexer. Precise control would require a multiplier, which would substantially increase the size of a cell. Secondly, the damping could be rapidly switched to simulate a damping setting between actual levels:

$$d.\partial t = 2^{-a} \tag{22}$$

where a controls the level of damping.

The exact equations implemented in this work are given by (23), where (21) and (22) have been substituted into (12):

$$\begin{aligned} v(l,n+0.5)\partial t =& (1-2^{-a})v(l,n-0.5)\partial t \\ &+ \frac{i}{2^b}\begin{pmatrix} y(l-1,n) \\ -2y(l,n) \\ +y(l+1,n) \end{pmatrix} + p(l,n) \\ y(l,n+1) =& y(l,n) + v(l,n+0.5)\partial t \end{aligned} \tag{23}$$

If the design is to be used as a hardware synthesiser, it is important that it should be able to reproduce the equal tempered tuning values for fundamental frequency of the western orchestral scale accurately. By rearranging (21), the frequency produced by each input level can be calculated as

$$f_0 = \sqrt{\frac{f_s^2 i}{2^{b+2}(N+1)^2}} \tag{24}$$

To enable accurate selection of frequencies, the difference between the fundamental frequencies produced by different inputs levels should be small. The difference between the frequency produced by i and $i+1$ (remembering that i may only have an integer value) is called f_g and is defined by (25), which derives directly from (24):

$$f_g = \frac{f_s}{2^{0.5b+1}(N+1)}\left(\sqrt{i+1} - \sqrt{i}\right) \tag{25}$$

where f_g is the frequency gap.

Figure 1 shows the variation of f_g with i when the sampling frequency is set to 44.1 kHz, $b = 11$, $2^{-a} = 0$ and $N = 32$. This shows that f_g decreases with i. It compares the frequency gap to the just noticeable difference (JND) approximation given by Zwicker and Fastl [18]. This states that the JND equals 1 Hz for all frequencies less than 500 Hz, and one-500th of the frequency for all values greater than 500 Hz.

It is therefore possible to tune the design to a desired specification of the maximum allowable frequency gap within a desired range. Therefore the frequency variation due to adjacent inputs can be guaranteed to be less than the JND for a selected range. A frequency f_{JND} is defined such that when i is incremented by one, f_{JND} increases by 1 Hz. This is shown by (26). To design a cell, the value of b required for a desired f_{JND} and $f_{0_{\max}}$ must be found:

$$\frac{4(N+1)^2 f_{JND}^2}{f_s^2} = \frac{i}{2^b} \quad \frac{4(N+1)^2(f_{JND}+1)^2}{f_s^2} = \frac{i+1}{2^b} \tag{26}$$

$$\frac{1}{2^b} = \frac{i+1}{2^b} - \frac{i}{2^b} = \frac{4(N+1)^2(f_{JND}+1)^2}{f_s^2} - \frac{4(N+1)^2 f_{JND}^2}{f_s^2} \tag{27}$$

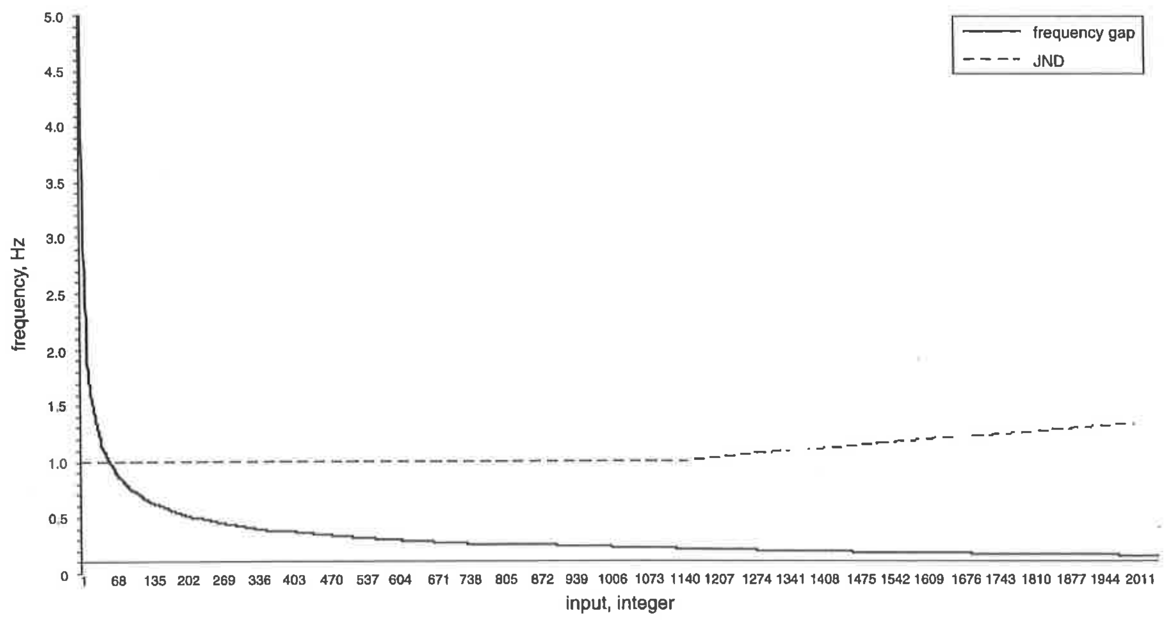

Fig. 1 *Frequency resolution of evaluation model against just noticeable difference*
Sampling frequency = 44.1 kHz

$$\frac{1}{2^b} = \frac{4(N+1)^2}{f_s^2}\left((f_{JND}+1)^2 - f_{JND}^2\right) = \frac{4(N+1)^2(2f_{JND}+1)}{f_s^2} \qquad (28)$$

$$b = \log_2\left(\frac{f_s^2}{4(N+1)^2(2f_{JND}+1)}\right) \qquad (29)$$

Substituting (19) into (29) gives (30), which defines the required value of b in terms of f_{JND} and $f_{0_{\max}}$:

$$b = \log_2\left(\frac{f_{0_{\max}}^2}{(2f_{JND}+1)}\right) \qquad (30)$$

3 Implementation

The algorithm was implemented in hardware using Xilinx ISE 5.1 design tools to synthesise, place-and-route the design. ModelSim 5.6a was used to verify the design in simulation before it was placed on a Xilinx Virtex 1000 FPGA. The FPGA was on a Celoxica RC1000 card, which fits in a PCI slot on a desktop PC. It has four 2 Mb banks of SRAM, which buffer communications between the host PC and the FPGA and allow the FPGA to store data locally. Celoxica provide functions that the host PC can use to DMA data to the SRAM so that the FPGA can process it and vice versa. This is a limiting factor on the performance of the board as a synthesiser. It was found that stable real-time operation could only be achieved by retrieving data from the board in blocks of 1024 values. This limits the real-time control of the synthesiser on the development board. However, this work is not intended to advocate development boards as synthesisers. It is an investigation as to the feasibility of the FPGA as the central processing device for hardware synthesis where this bottleneck will not exist.

This hardware design is simply an implementation of an algorithm. As such, the structure of each cell is fully described by (23). Two designs were implemented which both use 32 cells. The first design was used to evaluate the model and operates on 32-bit integers for comparison with software. The second is optimised to show the performance levels the design can achieve.

3.1 Bit-serial design

The equations require three mathematical operations: add, negate and multiply. These are built from logic and registers and used as components to construct a cell. Each of the components is bit-serial and is based on those found in [19]. This means that they process data one bit at a time, and as such one clock cycle of the FPGA does not correlate to one sample output. The three mathematical components are shown by Fig. 2, with a fourth component called 'latch sign extension'. This is used when shift registers are tapped to perform a division by a power of two, as shown by Fig. 3. They latch the tap while the MSB is being output from it. This ensures correct sign extension of the data. The control signals for these latches are shown by 'damping latch', 'displacement latch 1' and 'displacement latch 2' in Fig. 4.

The area occupied by a bit-serial design is approximately $1/N$th that of an N-bit parallel design. The tradeoff is that the parallel design is N times faster. If every component in this design were made parallel, it would therefore be 32 times larger. Since even the simplest design uses 16% of the FPGA, it would not fit. The solution that best optimises the FPGA's performance for a specific number of cells would be digital serial [20], which maximises parallelism for given resources and design size. The goal of this work was to minimise the resources required to implement a string, to maximise the potential number of cells. This enables multiple strings to potentially be implemented on the same chip. For example, a virtual violin would require four strings. Secondly, if the string were to be embedded in a virtual environment, it may need to be modelled by a large number of cells, so the propagation of the sound through air can be accurately represented. However, in this case digit serial may be more appropriate, as it allows further reductions in the required clock frequency. Section 4 shows that the required clock speed of the FPGA increases

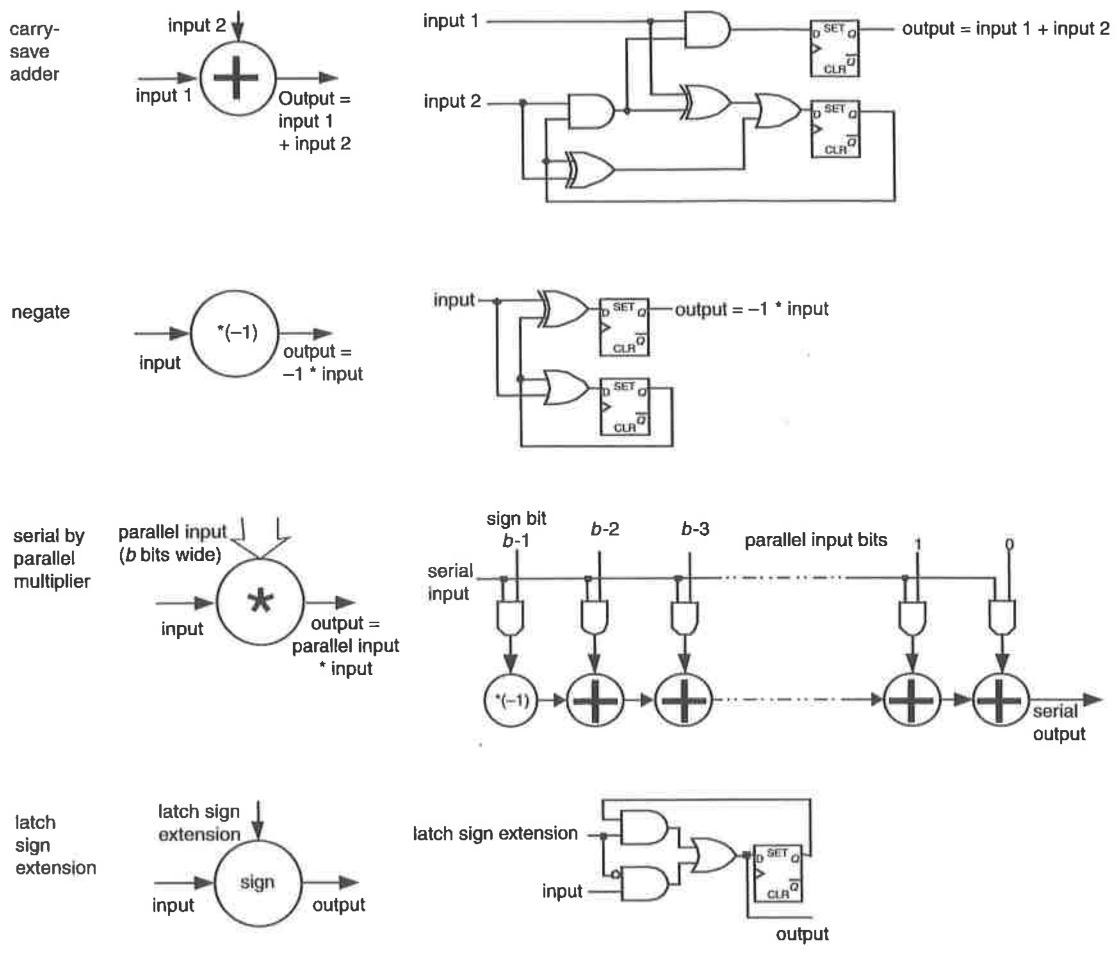

Fig. 2 *Bit serial components used in design*

linearly with the number of cells in a single string. For example, a digit size of 2 would halve the required clock frequency at the expense of doubling the area of the circuit. The results section shows that an optimised bit-serial design with a good range of output frequencies uses only 23% of the chip, so this is a feasible option.

If floating point hardware were to be used (see Durbano *et al.* [14]), then each set of mathematical operations would have to be responsible for more than one cell because they are so large. For example, a floating point solution would match the performance of the design presented here if it could be pipelined such that it calculates the next displacement of each cell in one clock cycle. This would take 32 clock cycles, while the duty cycle of this design is 34 clock cycles. However, this would require a much more complex, larger and less flexible design, for a minimal gain in performance and no audible improvement. The advantage of bit-serial cellular design is that each memory location can have its own set of mathematical operators so that it does not need to be addressed, which saves routing resources and makes the pipeline easier to design.

3.2 Evaluation model

The architecture to implement (23) is shown in Fig. 3. This cell is fully pipelined. It processes 32-bit signed integers, producing an output every 34 clock cycles. The remaining two clock cycles are used to reset the mathematical operators and to set them so that they are ready to process the LSB of the next sample. The numbers on Fig. 3 show the position of each data bit at the start of a duty cycle. The output of the operators designated by 'reset stage 1' is defined as 'set'. This means they have been reset on the previous clock cycle and are now active, since the LSB is being presented at their inputs. 'Reset stage 2' is currently being reset, while every other part of the circuit holds valid data.

Figure 4 shows the duty cycle of the evaluation design, including the reset times for every stage. It produces an output after 32 clock cycles from the position marked 'output displacement' in Fig. 3. This feeds a shift register, which, at the time shown by 'write displacement to memory' on Fig. 4, writes the data to a 32-bit buffer in parallel. Figure 5 shows the complete circuit including all the control structures required to operate the cells for the evaluation circuit. This provides the user with six controls. The pitch of the output can be selected. The cell to use as the output can be selected (the 'pickup'). The cell to pluck may be chosen (the 'pluck' location). The final three inputs allow for control over the damping level, the length of time to pluck and the pluck input itself. The optimised circuit is identical apart from down-sampling the output, which uses a 4-bit counter. The output is a 32-bit number that conforms exactly to the IEEE specification of an 'int' data type. This allows the data to be passed to the host PC without modification. The data are output to SRAM, which is used purely to buffer communications with the host PC. Owing to the time overhead of setting up a DMA transfer, the data have to be passed to the host in blocks of 1024 samples. Two SRAMs are used for this task, with a third providing the control inputs from the host (e.g. the pluck and pickup locations). While one output SRAM is transferring data to the host, the other builds up the next block to be transferred so that no data are lost. This work is not intended to advocate development boards as synthesisers because of

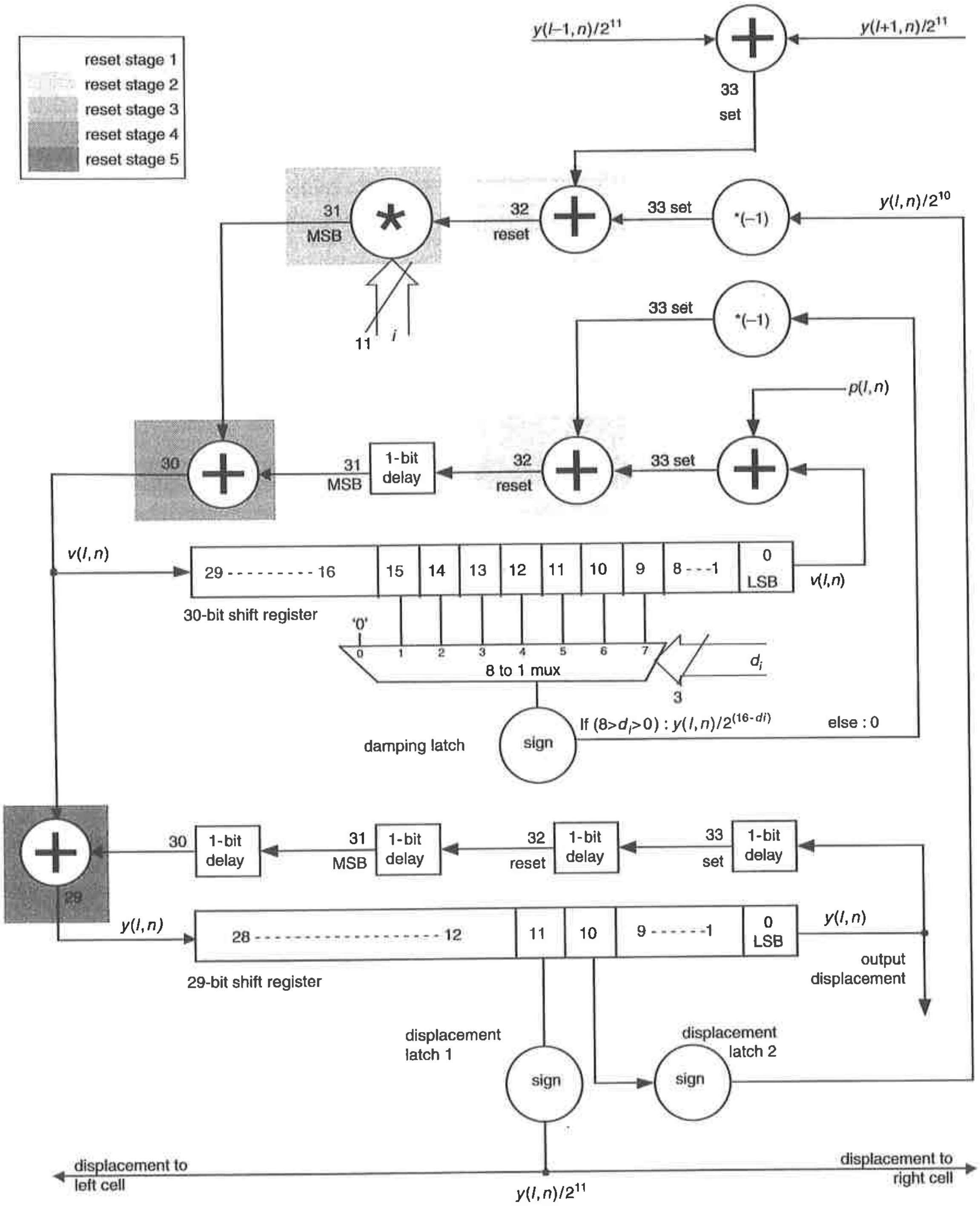

Fig. 3 *Single cell of evaluation model*

the overhead involved in DMA transfers. A development board was used purely because it is the simplest way to develop FPGA applications.

The FPGA must operate at a frequency equal to the duty cycle multiplied by the desired sampling frequency, which in this case is 44.1 kHz. With a duty cycle of 34, this implies an operating frequency of 1.4994 MHz. These are the exact parameters used to create Fig. 1, which shows the frequency gap between the 2048 possible input levels of the design. The maximum output frequency of the design can be found using (19). This is shown by

$$f_{0_{\max}} = \frac{f_s}{2(N+1)} = \frac{44100}{2.(32+1)} \approx 668\,\text{Hz} \qquad (31)$$

The frequency above which the gap in output frequencies produced by adjacent inputs is less than the pitch JND (effectively less than 1 Hz) can be found by rearranging (30). The value of f_{JND} is determined by b, which was set to 11. This was chosen as a compromise between lowering f_{JND} and retaining an acceptable dynamic range d_r. The dynamic range represents the number of bits available to describe the amplitude of the output waveform. It is equal to the total number of bits available minus b. This is because any input with a magnitude less than 2^b cannot cause oscillation in the string. The 21 bits available here should be ample, since CD audio uses only 16 bits per channel. Therefore, the design can be considered equivalent to a floating point solution in terms of dynamic range. The resulting equation and calculations for this model are shown by (32):

$$d_r = 32 - b = 21\,\text{bits}$$

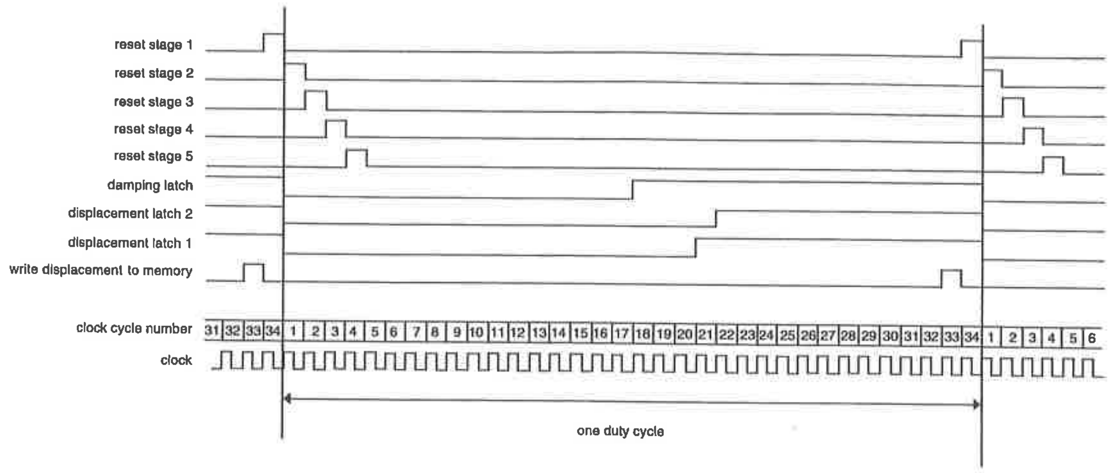

Fig. 4 *Waveform showing global duty cycle for evaluation model with damping = 2*

$$f_{JND} = \frac{f_{0_{\max}}^2}{2^{b+1}} - \frac{1}{2} = \frac{668^2}{2^{11+1}} - \frac{1}{2} \approx 109\,\text{Hz} \qquad (32)$$

where f_{JND} is the frequency above which the frequency gap is less than the pitch JND.

Since this implementation operates at the same sampling frequency as the soundcard on the host PC, the output does not need to be down-sampled. This work advocates the use of FPGAs as the central processor in dedicated hardware synthesisers because of the potential performance gains that

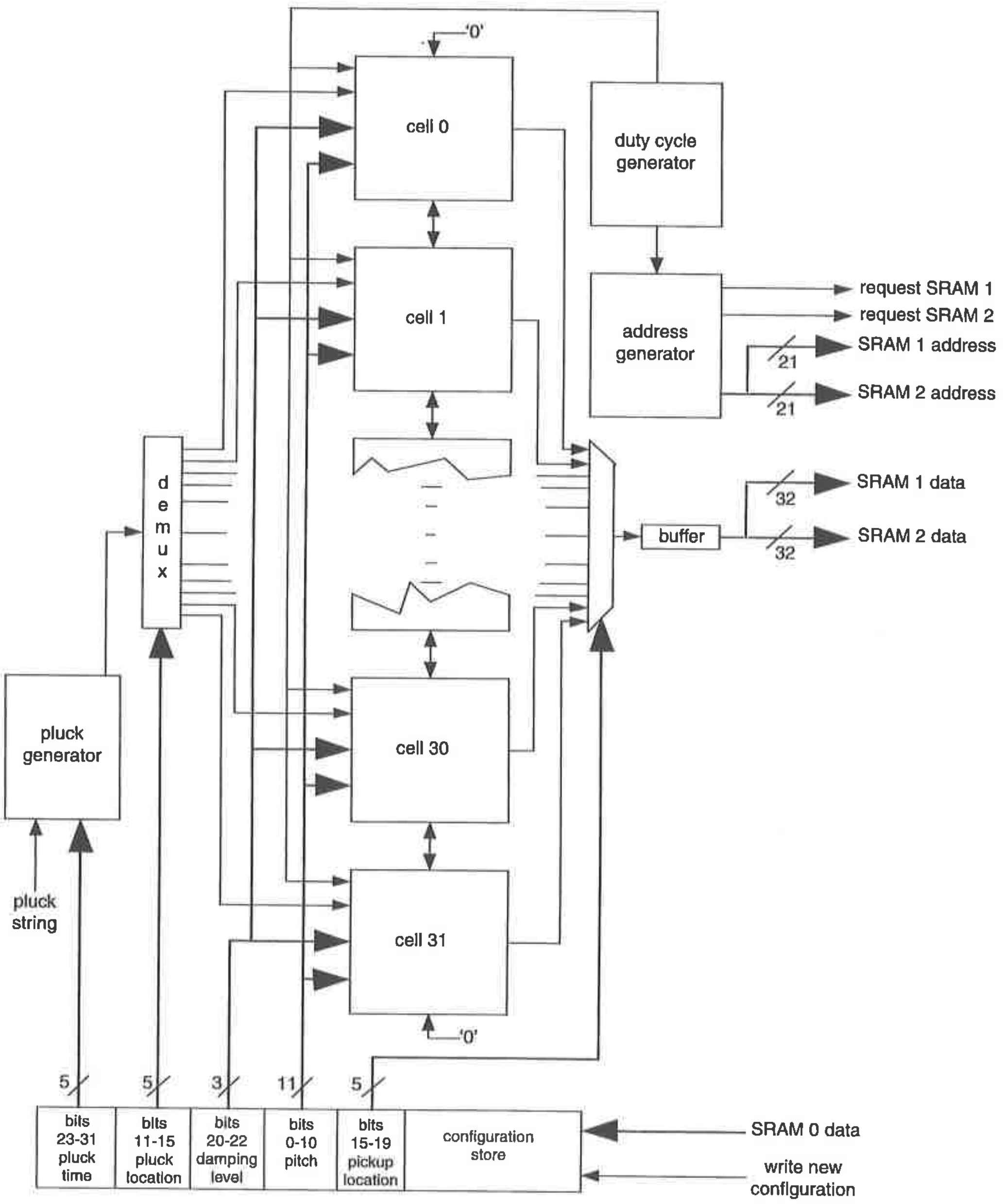

Fig. 5 *Complete circuit of evaluation model*

can be achieved with cellular designs (see Section 4). A dedicated synthesiser could include a fast digital to analogue converter (DAC), allowing the system to produce a full range of frequencies without down-sampling.

3.3 Optimised model

This model was designed without the constraint of being directly comparable to software. It down-samples the output in hardware, outputting every 16th sample. This allows a 16-fold increase in internal operating frequency compared to the previous model, providing a proportional increase in maximum fundamental frequency as shown by (33):

$$f_{0_{\max}} = \frac{f_s}{2(N+1)} = \frac{44100 \times 16}{2(32+1)} = 10691\,\text{Hz} \qquad (33)$$

Next, we desire the frequency gap to be less than the JND for all frequencies greater than 20 Hz, which is approximately the lowest audible frequency. Equation (30) is applied in (34) to calculate the value of b required to achieve this. Since b must be an integer, it is rounded up to 22:

$$b = \log_2\left(\frac{f_{0_{\max}}^2}{(2f_{JND}+1)}\right) = \log_2\left(\frac{10691^2}{[(2\times 20)+1]}\right) \qquad (34)$$
$$= 21.41 \approx 22\,\text{bits} \quad d_r = 23\,\text{bits}$$

The evaluation model was prone to overflow errors when no damping was applied since the amplitude of oscillation increases every time the string is plucked. To partially alleviate this, two extra bits were added to the dynamic range d_r, in addition to the extra 11 required by the increase

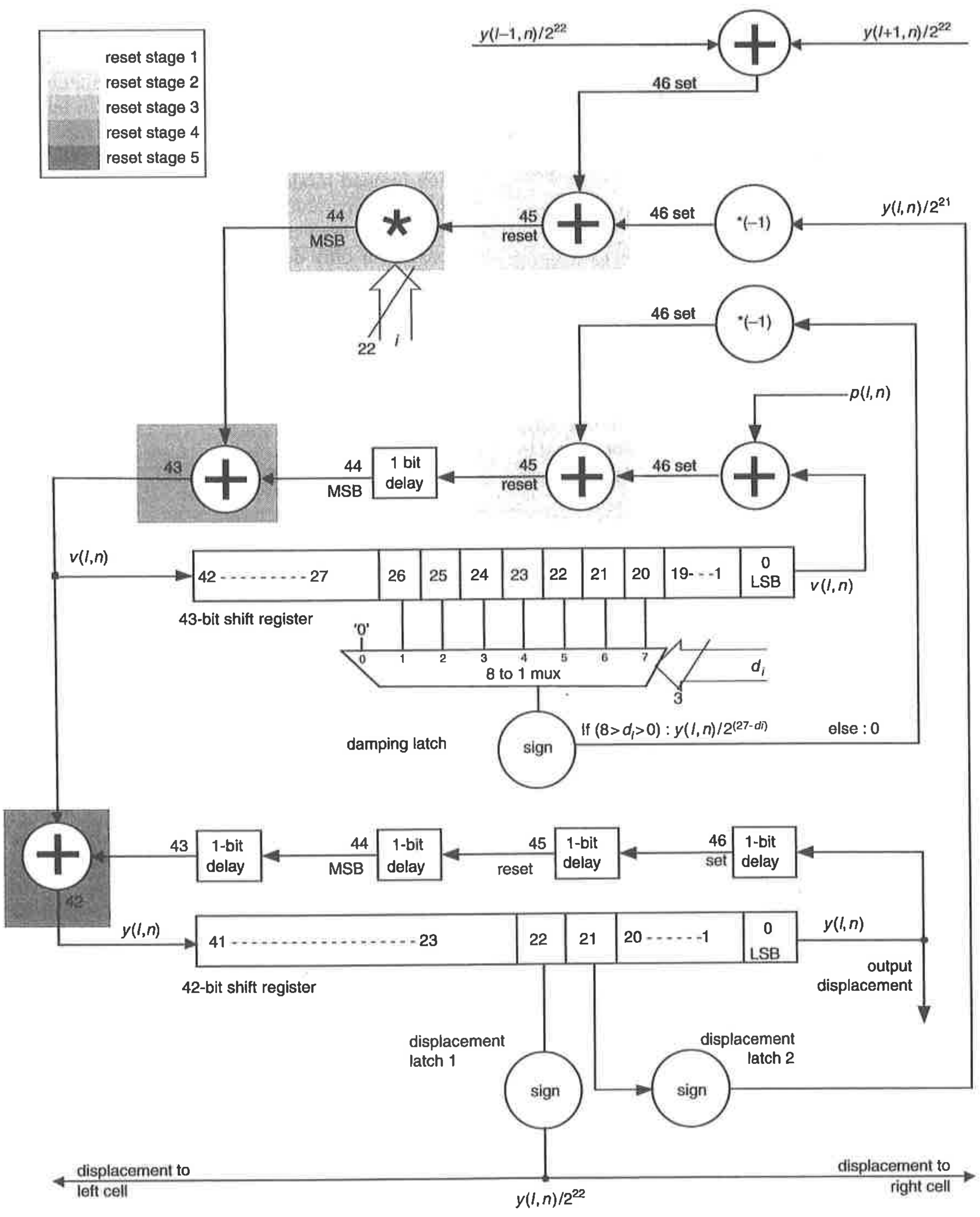

Fig. 6 *Single cell of optimised model*

of b. The resulting design has a duty cycle of 47 clock cycles operating on 45 bit numbers. The design is shown in Fig. 6. To retain compatibility with the host PC, the 32 most significant bits were output as integer data. The required operating frequency of the FPGA is equal to the sampling frequency multiplied by the duty cycle, which is 33.1632 MHz.

A simple software interface written using the windows API was used to control both of the models. The output was played in real time using the synthesis toolkit (STK), which can be found in [21].

4 Theoretical comparison of performance of FPGA design against a sequential processor

Rearranging (19) gives (35), which shows that the required sampling frequency is a function of the desired maximum fundamental frequency and the number of cells:

$$f_s = 2f_{0_{\max}}(N+1) \tag{35}$$

To achieve this sampling frequency on a sequential processor, each cell must be solved sequentially. Therefore the required clock frequency is given by (36):

$$f_{clk} = 2O_c f_{0_{\max}}(N^2+N) \tag{36}$$

O_c is the number of clock cycles required to solve one iteration of a single cell.

Using (13) and (14) O_c is estimated to be 8. The desired $f_{0_{\max}}$ is 22.05 kHz. Therefore, the variation of clock frequency with N for this model on a sequential processor can be estimated by (37):

$$f_{clk} = 352800(N^2+N) \tag{37}$$

The FPGA solves every cell simultaneously in one duty cycle. The required operating frequency is therefore equal to the sampling frequency multiplied by the duty cycle, as shown by (38):

$$f_{clk} = 2f_{0_{\max}}(N+1)(d_r+b+2) \tag{38}$$

Substituting (30) into (38) gives (39):

$$f_{clk} = 2f_{0_{\max}}(N+1)\left(d_r + \log_2\left[\frac{f^2_{0_{\max}}}{(2f_{JND}+1)}\right]+2\right) \tag{39}$$

Selecting a dynamic range of 32, a maximum output frequency of 22.05 kHz and a 'just noticeable difference' frequency of 20 Hz gives (40). Comparing this with (37), it can be seen that the required clock frequency is a function of N^2 on a sequential processor, but it is linear with respect to N on an FPGA. The larger N becomes, the greater the gains that are achieved by using an FPGA. Figure 7 shows the performance gains that are theoretically possible on an FPGA using this design with up to 250 cells. With that many cells, the FPGA would need to operate at 642 MHz, while the sequential processor would have to run at 22.1 GHz:

$$f_{clk} = 2557800(N+1) \tag{40}$$

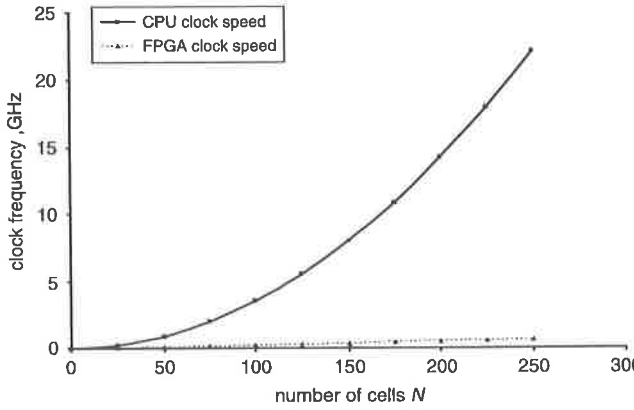

Fig. 7 *Increase of required clock speed against cell numbers for an FPGA and a sequential processor*

5 Results

The place-and-route report provided by Xilinx ISE showed that the evaluation design used 16% of available slices on the chip. This is a measure of the use of available resources of a Virtex 1000 FPGA. As a comparison with the digital waveguide, a dual delay line was implemented that used the same amount of space. This compares the lumped model including all control circuitry and I/O buffers to just two delay lines. The evaluation model was equivalent to two 32-bit delay lines of length 63. To match the optimised model, the delays lines were increased in length to 88. This is demonstrated by the post-synthesis reports provided by Xilinx ISE software shown by Fig. 8. It shows the reports for each lumped model and each waveguide. The two designs use a comparable number of slices and flip-flops. The waveguide does not require any look-up tables since it consists of only delays. The IOBs referred to in Fig. 8 are input/output blocks which the FPGA uses to interface with its SRAM. The lumped design requires more because it has had all its external control implemented. If the waveguide were implemented as an equivalent synthesiser, it would need just as many. The waveguide operates at 332 MHz because in this design it consists of just delays and as such

Post-synthesis summary for the evaluation lumped model
Number of slices: 2027 out of 12288 16%
Number of slice flip-flops: 3565 out of 24576 14%
Number of 4 input LUTs: 1859 out of 24576 7%
Number of bonded IOBs: 183 out of 408 44%
Minimum period: 15.352 ns (maximum frequency: 65.138 MHz)

Post-synthesis summary for the optimised lumped model
Number of slices: 2870 out of 12288 23%
Number of slice flip-flops: 5123 out of 24576 20%
Number of 4 input LUTs: 2621 out of 24576 10%
Number of bonded IOBs: 183 out of 408 44%
Minimum period: 14.072 ns (maximum frequency: 71.063 MHz)

Post-synthesis summary for the waveguide using two 32-bit delay lines of length 63
Number of slices: 2021 out of 12288 16%
Number of slice flip-flops: 3968 out of 24576 16%
Number of bonded IOBs: 2 out of 408 0%
Minimum period: 3.009 ns (maximum frequency: 332.336 MHz)

Post-synthesis summary for the waveguide using two 32-bit delay lines of length 88
Number of slices: 2869 out of 12288 23%
Number of slice flip-flops: 5632 out of 24576 22%
Number of bonded IOBs: 2 out of 408 0%
Minimum period: 3.009 ns (maximum frequency: 332.336 MHz)

Fig. 8 *Post-synthesis reports to show that the designs use an equivalent area to a digital waveguide*

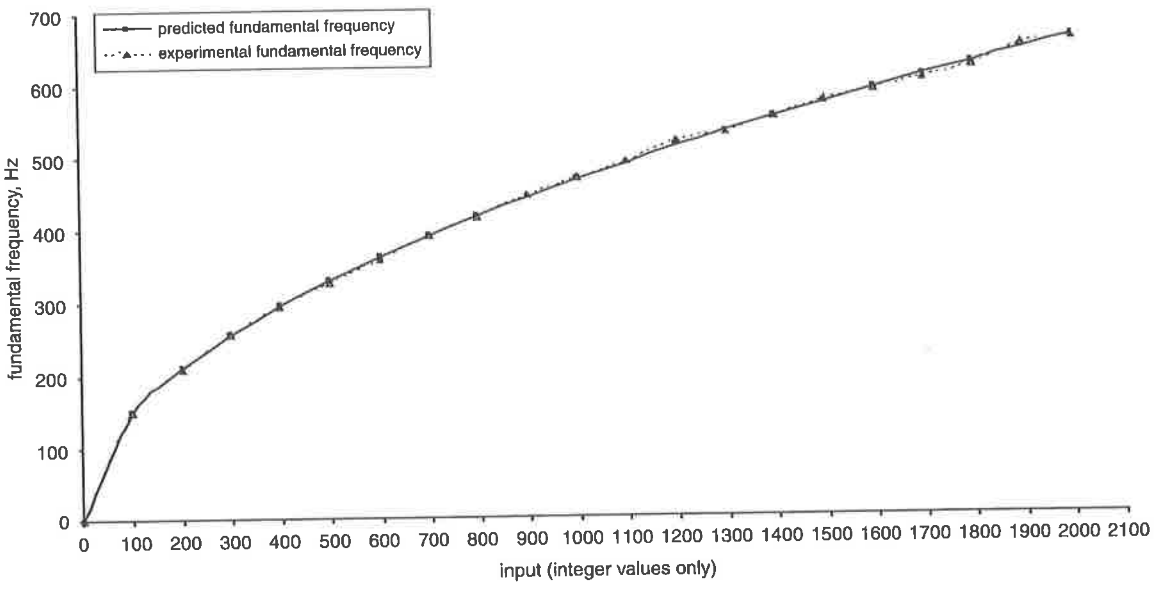

Fig. 9 *Evaluation model output compared to predictions*

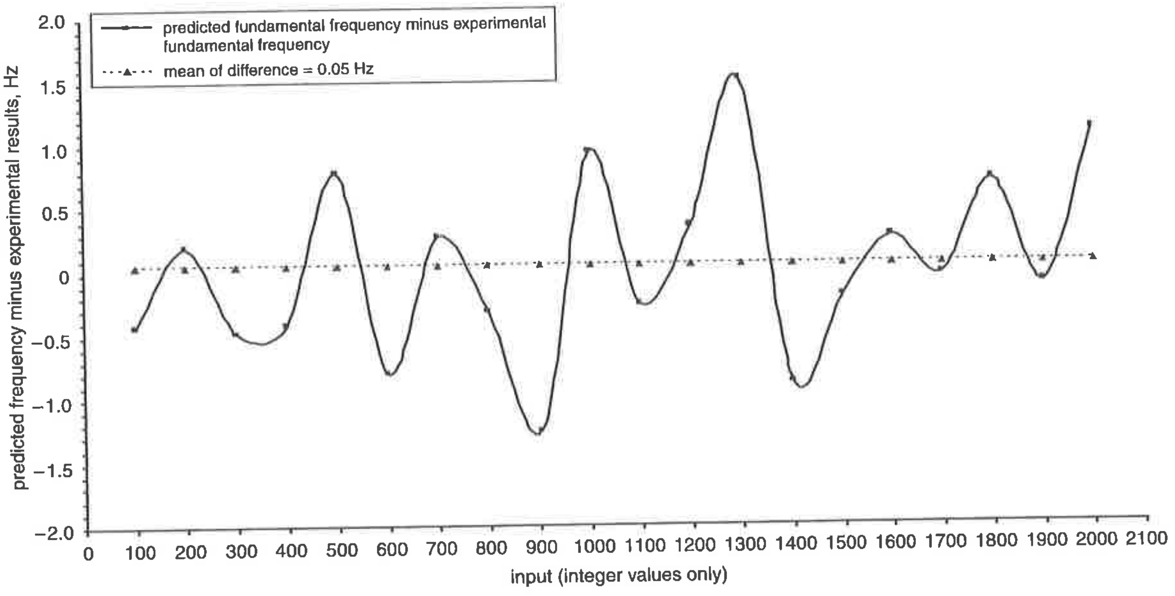

Fig. 10 *FPGA outputs from evaluation model compared to predictions with no damping*

the minimum delay path can be minimised to a much greater extent than the lumped model. This estimate should not be considered to represent how fast a real waveguide would operate. A waveguide that is appropriate for synthesis would require an adder at every location where an output is desired and filters at each end to simulate damping. These would require longer delay paths and massively reduce the speed of the design.

The maximum operating frequency of the lumped models was estimated to be about 65–70 MHz, as shown by Fig. 8, although this is just a synthesis estimate. This slow clock speed is because the array of cells occupies a significant area of the FPGA, but each cell receives pitch and damping information from the same source. This causes some very long connections, which limits the speed of the design. A solution would be to give each cell its own configuration bit string, or put extra delays on the signal path to each cell. This was not implemented since the design operated fast enough to demonstrate real-time synthesis, so it was unnecessary to further optimise the model for speed. As mentioned in the implementation section, the evaluation model was clocked at 1.4994 MHz and the optimised model at 33.1632 MHz. Therefore, the maximum operating frequency estimated by the synthesis report was never approached by the models implemented in hardware and must be considered an estimate.

Figure 9 shows the variation of the fundamental frequency with the input for the evaluation model with the damping level set to 0. The frequencies predicted by (24) are compared to the frequencies output by the FPGA. The difference between the predictions and the output at each measured frequency is shown by Fig. 10. This shows that the average error is very close to zero. The frequencies were measured by calculating the wavelength of the output using linear interpolation, which is presumed to be the source of the error in Fig. 10. The first complete wavelength after the 32768th sample was measured for each frequency. The string was plucked for 50 clock cycles in each frequency

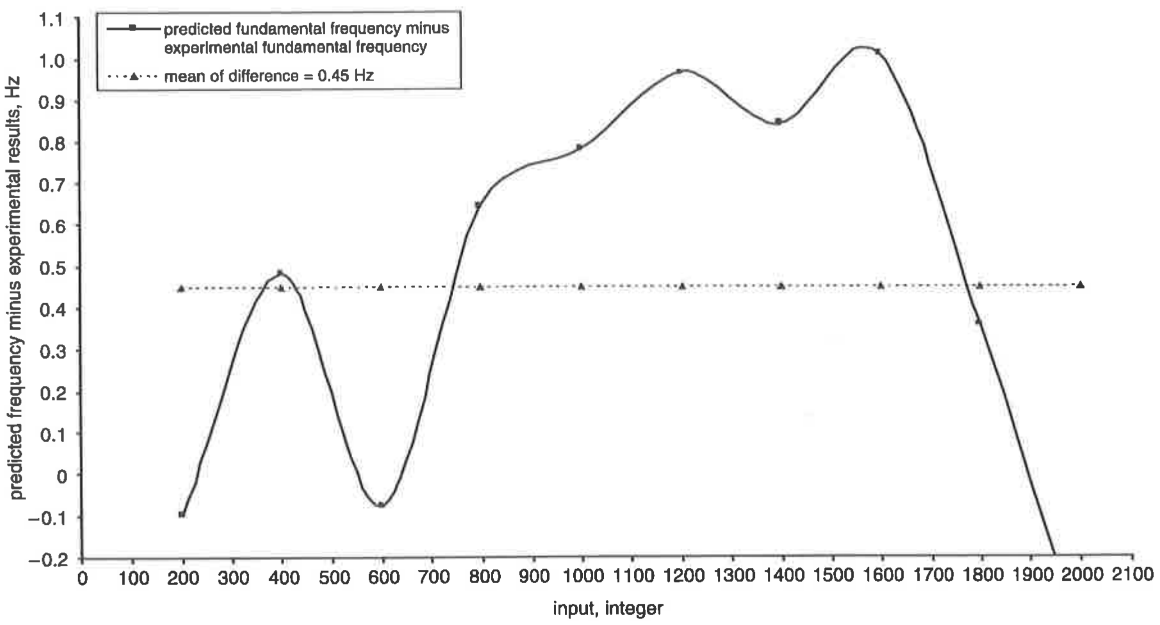

Fig. 11 *FPGA outputs from evaluation model and predictions with damping*

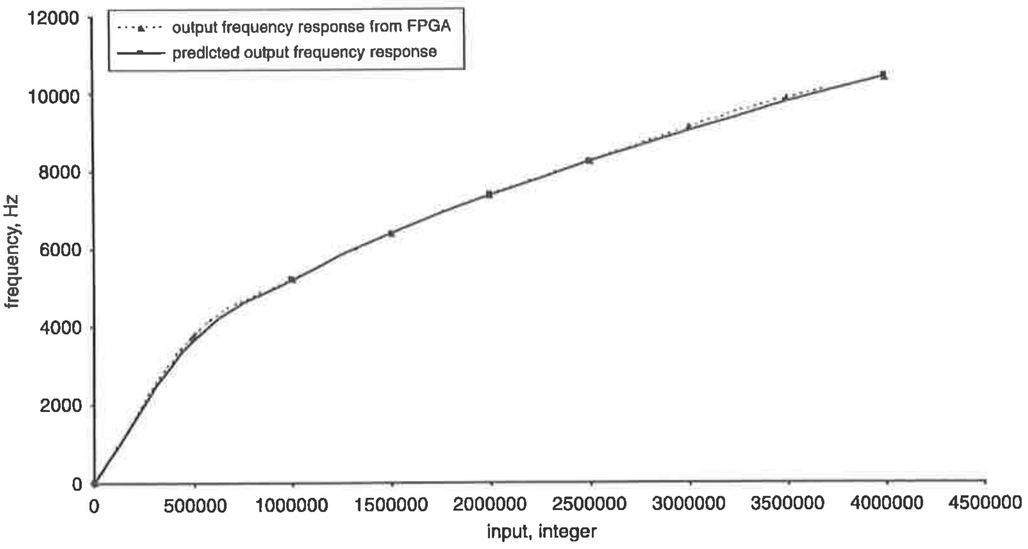

Fig. 12 *Optimised model output compared to predictions*

measurement. The plucking force in each case was proportional to the input pitch setting. This was found to help in avoiding overflow errors. Figure 11 repeats the experiment with the damping level set to 2 and shows that the output frequencies are slightly lower than the predictions, which is an effect of the damping. Figure 12 repeats the experiment of Fig. 9 on the optimised model to prove that it operates correctly.

6 Conclusions

The goal of this work was to investigate whether an FPGA could perform lumped synthesis while retaining a level of output frequency selectivity that would make the model useful to a musician. The results show that it is achievable on the device used with bit serial design. Minimal hardware area was required to implement the design on a fault tolerant FPGA. This was achieved by minimising the number of multipliers that are needed by the algorithm and by minimising the requirements of routing resources by using bit-serial design, which exploits the plentiful local connections in the FPGA. Future work dedicated purely to FPGAs should use digital serial design. For a doubling of the required area, the required clock frequency can be halved. In this way, a very large number of cells can be implemented in a single string such that the potential benefits shown by Section 4 can be truly realised.

This work is seen as a demonstration of the potential power of FPGAs in music synthesis. The design was implemented on a Virtex 1000 chip, which is relatively small by current standards, yet the optimised design required only 23% of available resources. The maximum operating speed of the design was reported by the synthesis tools as 71.063 MHz. This is fast enough to produce a range of frequencies up to 21.38 kHz whilst retaining an acceptable JND at 20 Hz if the optimised model is run at 66.33 MHz with 32 times down-sampling and a duty cycle of 47. This allows 21 bits of dynamic range. This was not tested, since it would still not represent the best the chip could achieve. The designs implemented in hardware should be considered a proof of principle and do not exploit the full potential of the FPGA. There is considerable margin for increasing the clock rate by eliminating long nets from the inputs, and using digit serial design could reduce the duty cycle.

If operated at 44.1 kHz, the waveguide that uses an equivalent area to the optimised model would provide a frequency range of 250 Hz to 22.05 kHz. This compares to the range of 20 Hz to 10.688 kHz provided by the optimised model. The waveguide in the form presented here allows only 88 different frequencies to be produced, which means it cannot be tuned. This could be improved by adding extra circuitry, but it would then use a larger area than the model presented here.

The design is stored in a file (called a.bit file) which is downloaded to the FPGA. For the part used in this work, each.bit file uses 749 kb of memory. Therefore, if an FPGA were packaged with a permanent storage device, it would constitute an open hardware synthesiser architecture. Different instruments could be kept on the storage device (as.bit files) and downloaded to the FPGA as required. The instrument could even be changed in real time during a performance if context switching is available, since this can change the configuration of the FPGA in one clock cycle. Secondly, instruments could be added or removed as needed or even upgraded as improved designs appear.

Further work has been performed into extending the number of variables in the model to include control over stiffness of the string and damping due to internal friction. The potential of extending the model to 2D to simulate drum skins has been investigated and found to be feasible. Possible improvements to the model include a digit serial implementation to maximise the power of the FPGA for processing a single string. Otherwise, the remaining space could be filled by a wave table to generate realistic pluck inputs to simulate a bow. Multiple strings could be implemented on one chip using bit serial design. Given that the control circuitry could be shared, four instances of the optimised model could be run simultaneously on a Virtex 1000. This is all possible on a single FPGA because it can be reconfigured. This makes the chip as flexible as a sequential processor running software. Secondly, Section 4 shows that the parallelism available on a large FPGA means designs can be conceived that would be impossible to operate in real time on a traditional processor using today's technology.

7 Acknowledgments

This work was funded by the EPSRC under grant number GR/R74512/01. The authors would like to thank the anonymous referees for their insightful comments and recommendations.

8 References

1 In: von Neumann, J., and Burks, A.W., (Ed.): 'Theory of self-reproducing automata' (Univ. of Illinois Press, 1966)
2 Wolfram, S.: 'Theory and applications of cellular automata Advanced series on complex systems - volume 1' (World Scientific Publishing, 1986), ISBN 9971-50-123-6, p. 1
3 Dodge, C., and Jerse, T.: 'Computer music' (Schirmer Books, 1985)
4 Smith, J.O.: 'Physical modelling using digital waveguides', *Comput. Music J.*, 1992, **16**, (4), pp. 74–91
5 Rimell, S., Howard, D.M., Tyrrell, A.M., Kirk, P.R., and Hunt, A.D.: 'Cymatic: Restoring the physical manifestation of digital sound using haptic interfaces to control a new computer based musical instrument'. Int. Computer Music Conf. ICMC-02, 2002
6 Hiller, L., and Ruiz, P.: 'Synthesizing musical sound by solving the wave equation for vibrating objects: Part 1', *J. Audio Eng. Soc.*, 1971, **19**, (6), pp. 462–470
7 Hiller, L., and Ruiz, P.: 'Synthesizing musical sound by solving the wave equation for vibrating objects: Part 2', *J. Audio Eng. Soc.*, 1971, **19**, (7), pp. 542–551
8 Howard, D.M., and Rimell, S.: 'Real-time gesture controlled physical modelling music synthesis with tactile feedback', *EURASIP. J. Appl. Signal Process.*, 2004, 7, pp. 1001–1006
9 Howard, D.M., Rimell, S., Tyrrell, A.M., Kirk, P.R., and Hunt, A.D.: 'Tactile feedback in the control of a physical modelling music synthesiser'. Int. Conf. of the Perception and Cognition of Music, Sydney, Australia, July 2002
10 http://www-ccrma.stanford.edu/ ~ jos/waveguide/Finite_Difference_Approximation.html, Physical audio signal processing: Digital waveguide modeling of instruments and audio effects, Julius O Smith III, *Center for Computer Research in Music and Acoustics (CCRMA)*
11 Pearson, M., and Howard, D.M.: 'A musician's approach to physical modelling'. Proc. Int. Computer Music Conf., ICMC-95, pp. 578–580
12 Schneider, R.N., Laurence, E.T., and Okoniewski, M.M.: 'Application of FPGA technology to accelerate the finite difference time domain method'. FPGA, Moterey, california, USA, 24-26 February 2002, pp. 97–105
13 Marek, J.R., Mehalic, M.A., and Terzuoli, A.J.: 'A dedicated VLSI architecture for finite difference time domain calculations'. 8th annual review of progress in applied computational electromagnetics, Monterey, CA, March 1992, Vol. 1, pp. 546–553
14 Durbano, J.P., Fernando, O.E., Humphrey, J.R., Prather, D.W., and Mirotznik, M.S.: 'Implementation of three-dimensional FPGA-based FDTD solvers: An architectural overview'. Proc. 11th Ann. IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM'03), 2003
15 http://ccrma.stanford.edu/ ~ jos/WaveDigitalFilters/, Stefan Bilbao and Julius O. Smith III (E-mail: jos@ccrma.stanford.edu) Center for Computer Research in Music and Acoustics (CCRMA), 2003.
16 http://www-ccrma.stanford.edu/ ~ jos/lumped/Finite_Differences_vs_Bilinear.html, Julius O. Smith Online Publications.
17 http://personalpages.umist.ac.uk/staff/david.d.apsley/lectures/numeric/pde_methods.pdf, methods for second order PDEs, page 2, spring 2004, David Apsley.
18 Zwicker, E., and Fastl, H.: 'Psycho-acoustics facts and models', Second Updated Edition(Springer Series in Information Sciences, Springer Verlag, 1999), ISBN 3-540-65063-6, pp. 185–187
19 http://www.andraka.com/files/fir.pdf Andraka, R.J. "FIR filter fits in an FPGA using a bit serial approach" Raytheon company, Missile Systems division, Tewksbury MA 01876.
20 Parhi, K.K.: 'A systematic approach for design of digit-serial signal processing architectures', *IEEE Trans. Circuits Sys.*, 1991, **38**, (4), pp. 358–375
21 http://ccrma.stanford.edu/software/stk/, Perry R. Cook, Gary P. Scavone, Synthesis ToolKit in C++ (STK), 1995-2004.