THE EFFECTS OF PIPELINING FEEDBACK LOOPS IN HIGH SPEED DSP SYSTEMS

Steven W. Alexander [1], *Robert W. Stewart* [2]

[1] Institute for System Level Integration, Alba Centre, Alba Campus, Livingston, EH54 7EG, UK
steven.alexander@sli-institute.ac.uk

[2] EEE Department, University of Strathclyde, 204 George Street, Glasgow, G1 1XW, UK
r.stewart@eee.strath.ac.uk

ABSTRACT

Many of today's Electronic Design Automation (EDA) tools include Intellectual Property (IP) cores that are fully pipelined to increase data throughput. Using these cores to implement data paths that do not involve feedback can result in fast, efficient designs. However, if they are used within a feedback loop this is not always the case. This paper examines the effects that using pipelined cores in feedback loops can have on a design. By considering two designs that implement a Givens rotation using feedback, which is used in QR decomposition [1], it is shown that, even though a pipelined design can be clocked faster, its data throughput is less than a non-pipelined design. Also, the non-pipelined design is shown to be smaller and consumes less power. Finally, a suggestion for a more efficient use of pipelining in feedback loops is presented, based on channel interleaving [2].

1. INTRODUCTION

Increasingly, many high sampling rate DSP algorithms are implemented on dedicated hardware rather than DSP processors. This has led to the development of EDA tools that allow engineers to design and simulate DSP systems before automatically generating an equivalent hardware design. Field Programmable Gate Arrays (FPGAs) are ideal for this type of rapid development and thus, in the current market, many of the DSP design EDA tools have been developed by FPGA manufacturers [3][4][5].

Much of the DSP IP that exists for FPGA implementation is designed to take advantage of the resources available on these devices. One common resource is the simple data register, which exists in large quantities throughout the fabric. To utilise this resource, many of the cores that exist within the EDA industry are pipelined to increase data throughput. Pipelining on FPGAs effectively costs nothing as the registers already exist within each configurable cell. Thus, there is a tendency to use them whenever possible. For data paths without feedback this can result in fast, efficient designs. However, this is not the case when data paths with feedback are considered.

This paper starts by considering the issues concerned with pipelining a simple feedback loop. In Section 3 the consequences of doing so are examined. Two hardware designs using feedback, one with pipelining and one without, which are algorithmically identical, are presented. This comparison is used to show differences in speed, area and power consumption and ultimately to reveal which approach is best. In Section 4, a scenario where pipelining a feedback loop offers some benefits is presented. This is based on channel interleaving for a suitable multichannel scenario and offers a low cost hardware solution for low data-rate applications. Section 5 reviews the generic outcomes of the work before the conclusions are presented in Section 6.

2. PIPELINING A FEEDBACK LOOP

It is possible to pipeline a simple feedback loop while still preserving the original algorithm. Figure 1 shows a feedback loop with an adder and some logic containing no registers. A single delay (reg 2) is required to synchronise the feedback data with the new data as it arrives.

The maximum speed at which this device can be clocked at is determined by the *greatest delay* between any

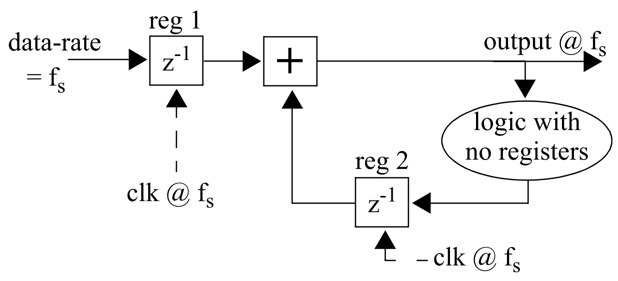


Figure 1: Simple feedback loop

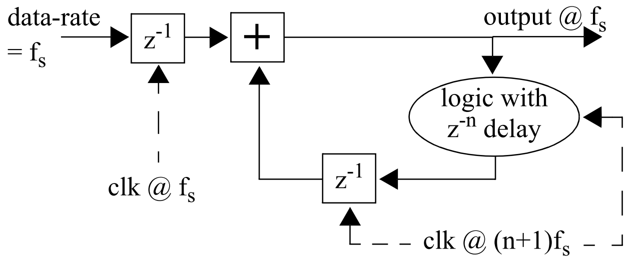


Figure 2: Simple feedback loop with pipelining

two connected registers. This delay represents the time it takes for data to travel between two registers and thus limits the clock speed. If the device is clocked faster than this limit, data to be captured by the receiving register will arrive after the clocking signal and so will not get registered. Thus, the longest delay between two registers in a design is known as the *critical path*. In Figure 1, the critical path is either the connection between reg 1 and reg 2 or it could be the feedback path connecting the output of reg 2 with its own input.

Figure 2 illustrates a pipelined version of the feedback loop shown in Figure 1, where it is assumed that pipelining registers in the feedback logic block will reduce the critical path. This circuit still represents the same algorithm as the non-pipelined design. To achieve this the registers in the feedback loop must be clocked at $(n+1) \times f_s$. The pipelined registers need to be clocked rate so that the feedback result arrives at the adder at the same time as the next sample. However, now the critical path will have been reduced as there are more registers in the data path. This means that the maximum clock speed is now increased. To determine the maximum data-rate at which this circuit can operate with, the maximum clock rate must be obtained and divided by *n+1*.

The simple question is then which of the two designs is faster and what are the respective power and logic resource requirements?

3. TO PIPELINE OR NOT TO PIPELINE?

To try to answer these questions, two circuits were designed using HDL Design Studio. This tool is used for the design, simulation and implementation of DSP systems on FPGAs. The methodology is based on the professional DSP design software, SystemView by Elanix [6], and uses a bit true fixed-point library (FXP-*Lib)* which maps directly to synthesiseable HDL code.

3.1. Givens Rotation With Feedback

The circuits designed were based on logic found in a QR decomposition (QRD) using Givens rotations [1]. This technique is used for RLS optimisation and has a wide range of application to adaptive filtering. The QR algorithm can be implemented using a parallel array of cells as illustrated in Figure 3. Each cell in the array performs a Givens rotation according to Eq. 1. However, the boundary cell on each row differs from the other cells in that it must calculate θ_i and pass it along the row to the other cells (the Givens Generation), as well as perform a Givens rotation itself.

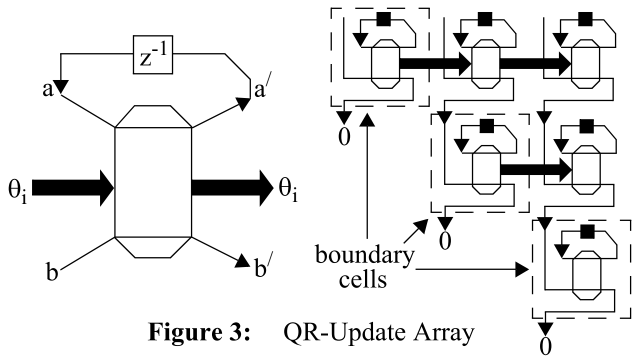


Figure 3: QR-Update Array

$$\begin{aligned} a^{/} &= a\cos\theta_i + b\sin\theta_i \\ b^{/} &= b\cos\theta_i - a\sin\theta_i \end{aligned} \qquad \text{Eq. 1}$$

For the purpose of the experiment, a pipelined and a non-pipelined implementation of this particular cell was chosen. A schematic illustrating one implementation of a boundary cell is shown in Figure 4. In this implementation, rather than compute θ_i to pass onto the other cells in the row, $\cos\theta_i$ and $\sin\theta_i$ are computed and passed on. The other cells must then perform 4 multiplies, 1 addition and 1 subtraction to complete a Givens rotation. Eq. 2 shows the relationship between the input vector *(a,b)* and the calculated values $\cos\theta_i$ and $\sin\theta_i$.

$$\begin{aligned} \tan\theta_i &= b/a \\ \cos\theta_i &= \frac{1}{\sqrt{1+\tan^2\theta_i}} = \frac{1}{\sqrt{1+(b/a)^2}} \\ \sin\theta_i &= \tan\theta_i \times \cos\theta_i = \frac{b/a}{\sqrt{1+(b/a)^2}} \end{aligned} \qquad \text{Eq. 2}$$

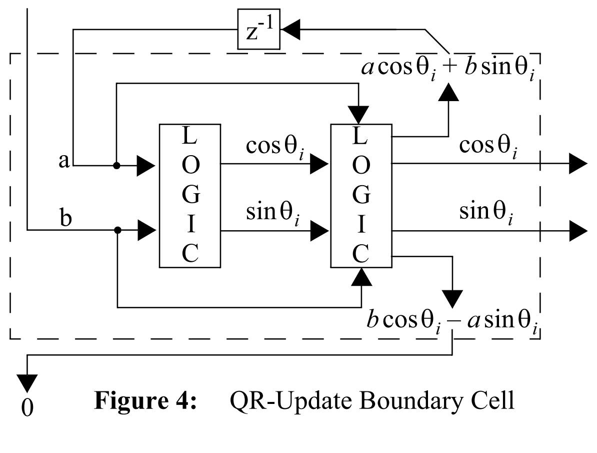
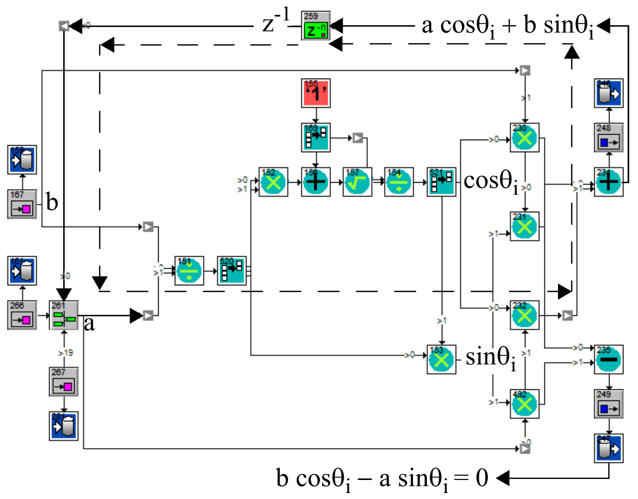


Figure 4: QR-Update Boundary Cell



z^{-1}
$a\cos\theta_i + b\sin\theta_i$
b
a
$\cos\theta_i$
$\sin\theta_i$
$b\cos\theta_i - a\sin\theta_i = 0$

Figure 5: Non-Pipelined Boundary Cell

3.1.1. Non-Pipelined Design

Figure 5 illustrates the non-pipelined boundary cell implemented using HDL Design Studio. The $\cos\theta_i$ and $\sin\theta_i$ components are computed using a combination of division, multiplication, addition and square root tokens in accordance with Eq. 2. The only register in the cell occurs in the feedback loop and is used to synchronise the feedback data with the next sample arriving. Note that the critical path is highlighted by dashed arrows and follows the path of the feedback loop.

3.1.2. Pipelined Design

The pipelined boundary cell is shown in Figure 6. Only the two dividers and the square root tokens are pipelined and combine to produce an overall delay of 86 samples. Thus, the pipelined stages are clocked at $(n+1 = 87) \times f_s$. Clocking at this speed is required to synchronise the feedback result with the next sample arriving at the f_s

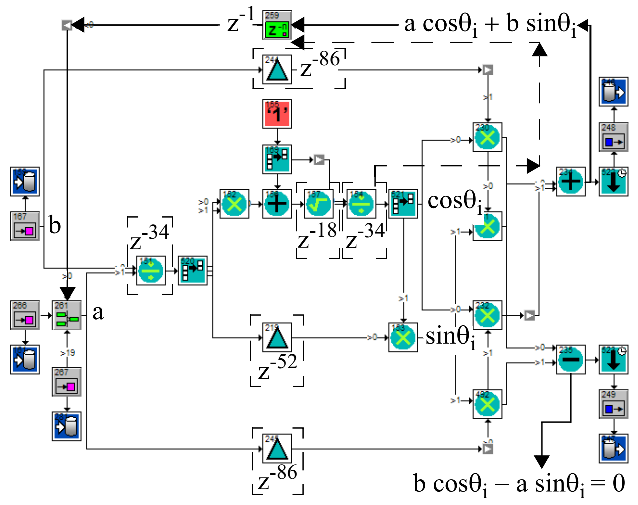


Figure 6: Pipelined Boundary Cell

data-rate thus maintaining the algorithm. The critical path in this design is also highlighted by dashed arrows. Note how much shorter it is compared to the non-pipelined design thus allowing the clock speed to be higher.

3.2. Synthesis Results

The results from synthesising both boundary cell designs are presented in Table 1. ISE 6.2.03i was used to target a Virtex-II XC2V8000 device.

Table 1: Synthesis Results

Design	Slices	Max. Clk Speed	Max. Data Rate	Total Est. Power Cons.
Pipelined	4672	45.55 MHz	523 kHz	902 mW
Non-Pipe	2046	2.755 MHz	2.755 MHz	787 mW

From these results the non-pipelined design is faster (in terms of data throughput), uses less logic and consumes less power than the pipelined device.

4. FILLING THE PIPELINE

The results from synthesis would suggest that there is no reason to pipeline a feedback loop. However, there is redundancy [2] in this structure that can be exploited, making pipelining viable under certain conditions. The pipeline, in the situation considered so far, never fills up because a new sample must wait on the result of the previous one before it can enter the pipeline. Thus, for a single input data channel, a sample will enter the pipeline and clock through each stage with nothing following it until it has passed through completely. This means that the majority of the logic is redundant for the majority of the time.

An approach that exploits this redundancy is to share the same structure with more than one input data channel (i.e channel interleaving as in Figure 7). Here we assume that the feedback loop has n pipeline stages. This means that up to n independent input channels can share this hardware. By multiplexing each input channel into the hardware, the pipeline can be filled. As soon as the first sample enters the pipeline and clears the first stage, a sample from another channel can enter.

This architecture offers a low cost hardware solution under the correct circumstances. Unless n input channels exist to fill n pipeline stages then there will still be some redundancy. Also, as n grows, the data-rate reduces, which means that this structure is only useful for low data-rate applications.

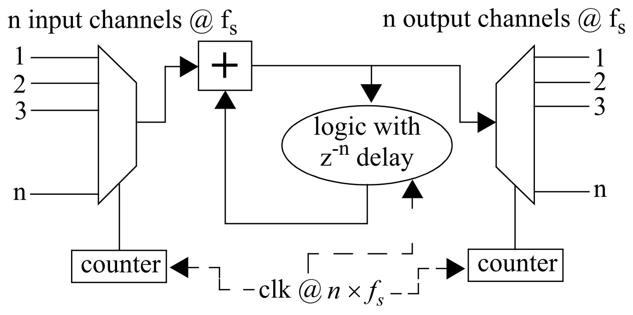


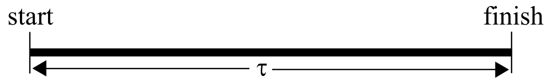
Figure 7: Channel Interleaving

5. DISCUSSION

The synthesis results have demonstrated that with a single data channel there is no reason to use pipelining in a feedback loop. The purpose of pipelining is to speed up data throughput, but clearly it has the opposite effect in this case. Not only is the pipelined version slower, but it uses more logic because of the extra registers and it consumes more power because of the extra clocking requirements.

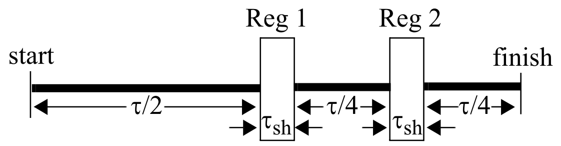
These findings can be explained further by considering Figure 8. Here, a simple wire is shown, where τ represents the time it takes for a signal to travel from the start to the finish. Figure 9 shows the same length of wire but this time it has been pipelined with two registers. The first register splits the wire in half so the time taken for the signal to travel from the start to Reg 1 is $\tau/2$. Reg 2 then splits the remainder of the wire in half, thus $\tau/4$ is the time taken for a signal to travel through each of the last two sections. So far we have accounted for the time the signal takes to travel through the 3 sections of wire. However, there is some additional delay that must be accounted for known as the setup and hold time [7]. The setup time is the minimum amount of time that data must arrive at a register before the clock signal if it is to be successfully latched. Similarly, the hold time is the minimum amount of time that data must be held for after a clock signal has arrived. Thus, the minimum delay between data reaching a register and getting latched is the accumulation of the setup and hold time, known as τ_{sh}. If the setup or hold time is breached, then a situation known as metastability can occur where the latched value cannot be predicted.

Accounting for τ_{sh} of both registers, the total time to travel along the wire becomes $\tau + 2\tau_{sh}$. Hence, it is clear that pipelining has only served to increase the travel time. This demonstrates that the shortest time for a signal to travel along a wire occurs without pipelining.



Total time = τ

Figure 8: A simple wire



Total time = $\tau/2 + 2\tau/4 + 2\tau_{sh} = \tau + 2\tau_{sh}$

Figure 9: A pipelined wire

6. CONCLUSIONS

This paper has shown that it is not always desirable to work with IP that has been pipelined. In situations involving a feedback loop it has been demonstrated that pipelined designs produce slower data throughput, use more logic and consume more power than a non-pipelined design. It has also shown that a non-pipelined design offers the fastest throughput possible and that a pipelined design cannot match this because of the additional delay due to the setup and hold time of each register in the pipeline.

A scenario that would benefit from using pipelined IP in a feedback loop was presented. This involved channel interleaving where the same pipelined feedback loop was shared with several data channels all running at the same data-rate. It was shown that for a pipeline with *n* stages, up to *n* independent data channels could be interleaved to share the same hardware and produce *n* independent output channels. This architecture offers a low cost hardware solution for low data-rate applications.

In summary, the conclusion that can be drawn from this work is that engineers using DSP algorithm design tools must fully understand the implications of working with pipelined IP blocks when implementing algorithms that require feedback.

7. ACKNOWLEDGMENTS

The authors wish to thank the Engineering and Physical Sciences Research Council for their financial support during this work.

8. REFERENCES

[1] Haykin S, Adaptive Filter Theory (2nd Edition). Prentice Hall, Englewood Cliffs, NJ, 1990.
[2] Parhi K.K, VLSI Digital Signal Processing Systems: Design and Implementation. Wiley-Interscience, 1999.
[3] http://www.xilinx.com/xlnx/xebiz/designResources/ ip_product_details.jsp?key=dr_dt_system_generator
[4] http://www.altera.com/products/software/products/dsp/ dsp-builder.html
[5] http://www.latticesemi.com/
[6] http://www.elanix.com/
[7] Wakerly J.F, Digital Design: Principles & Practices (3rd Edition). Prentice Hall, Upper Saddle River, NJ, 2000.