Simpler Adaptive Optics using a Single Device for Processing and Control

Anna Zovaro
The University of Sydney, Sydney, Australia
The Space Environment Research Centre, Canberra, Australia

Francis Bennet
The Australian National University, Canberra, Australia
The Space Environment Research Centre, Canberra, Australia

David Rye
The University of Sydney, Sydney, Australia
The Australian Centre for Field Robotics, Sydney, Australia

ABSTRACT

The Adaptive Optics Demonstrator (AOD) is an experimental LIDAR-based system used to track orbiting objects which uses adaptive optics (AO) to reduce the degradation of the laser beam as it propagates through the atmosphere. AO systems are by nature very complex and expensive due to computation and latency requirements; as a result it is common for an AO system for a 1–3 m telescope to involve multiple CPUs, digital signal processors (DSPs) and field-programmable gate arrays (FPGAs). In recent years, however, advances in FPGA technology have potentially enabled a single device to perform all data processing and control required for AO, presenting a far simpler, cheaper, and more efficient alternative to existing systems. In order to evaluate the feasibility of this concept, a simplified AO processing system implemented on a single low-cost FPGA is presented. In particular, this paper details the methods used to optimise the AO algorithms to maximise throughput by taking advantage of the FPGA architecture; these algorithms include wavefront sensor (WFS) thresholding, centroiding and wavefront reconstruction. It is found that the processing sequence required to generate deformable mirror (DM) commands from a raw WFS detector image can be executed at a rate of over 9 kHz using a single-precision floating-point implementation. This is significantly faster than the nominal bandwidth of 100 Hz for the AOD, indicating that today's FPGAs indeed have the potential to replace the existing AO processing infrastructure. The next step is to directly interface the FPGA to the WFS and DM, which is an important step in reducing the cost and complexity of an AO system; this will be addressed in a future investigation.

1. INTRODUCTION

Space debris poses a major problem to the future of vital services, such as communications and mapping, which are provided by multitudes of satellites. Due to high orbital speeds, debris as small as a centimetre can destroy a satellite, in turn creating more debris. A cascade of such events may lead to a Kessler syndrome, in which the concentration of debris is so high as to render the use of these orbits impossible [1].

The Research School of Astronomy and Astrophysics at the Australian National University in partnership with Electro-Optic Systems (EOS) Space Systems have been developing the Adaptive Optics Demonstrator (AOD) to range orbiting objects so that these events can be avoided [2]. The completed system, incorporating the 1.8 metre telescope at Mount Stromlo Observatory in Canberra, Australia, will employ a high-power infrared probe laser to range orbiting objects. The adaptive optics (AO) component provides atmospheric correction for imaging and laser ranging, allowing for the detection of smaller angular targets and drastically increasing the number of detectable objects. The AO system required is typical of those used on 1–3 m telescopes for space surveillance, and is computationally demanding, requiring a correction bandwidth on the order of 100 Hz. As a result, it is not unusual for AO systems to comprise multiple servers, digital signal processors (DSPs) and field-programmable gate arrays (FPGAs), with dedicated tasks such as wavefront sensor (WFS) data processing or wavefront reconstruction; in the AOD these requirements are met using a combination of a dedicated CPU and several PCIe cards. The resulting systems are complex and expensive, often costing millions of dollars and taking years to design. Despite their processing power, however, the need to transfer

data between the different components in the system introduces undesirable latencies. In particular, interconnections with non-deterministic transmission delays [3] as well as operating system jitter [4] can reduce the achievable bandwidth. While this multi-platform approach has been necessary in AO systems to date due to computation and latency requirements, this may no longer be the case for systems with less demanding processing needs.

In recent years, large advances have been made in FPGA and microprocessor technology, with today's devices having clock speeds in excess of 200 MHz whilst using a ≤ 5 V power supply. AO systems using a single such device for all processing and control may present a far simpler, cheaper, smaller and more efficient alternative to existing systems. For example, Basden et. al. in [4] found that a single CPU can successfully be used as a standalone controller in an AO system; however the performance of this system when scaled up to meet the more demanding processing requirements of more complex AO is limited by the fixed architecture of the CPU. In contrast, the extremely flexible architecture of FPGAs is suited to the acceleration of operations common in AO systems, as it enables programs to be restructured at the gate level. Moreover, developments over the past decade or so have effectively eliminated a number of barriers preventing their use as standalone AO controllers, with many of today's FPGAs having hardware floating-point support and hard-wired processors. The objective of this paper is to evaluate the feasibility of using a single FPGA as a standalone AO controller. In particular, the aim is to achieve satisfactory bandwidth and latency whilst meeting the processing requirements of a simple closed-loop AO system using the specifications of the AOD, through optimisation of the algorithms required to transform a raw Shack-Hartmann (SH) WFS detector image to the deformable mirror (DM) commands.

2. CURRENT FPGA TECHNOLOGY

FPGAs are highly configurable integrated circuits comprising programmable logic (PL)—or a 'fabric'—containing millions of basic logic elements and reconfigurable interconnects, from which programs are implemented at the circuit-level [5]. The FPGA architecture allows hundreds of operations to execute simultaneously, facilitating large-scale parallelisation. As a result they are often used in processing SH WFS data [6, 7, 4, 8] to minimise the latency of operations such as centroiding and filtering. Recently their increased clock speeds, cost and size have allowed FPGAs to replace large numbers of DSPs, for instance in the AO used in the 76 cm Advanced Technology Solar Telescope, in which two FPGAs handle a processing load which would otherwise require 96 DSPs [9].

Advances have been made towards implementing a complete AO system on a single FPGA [10, 8]; for example, Chang et. al. in [11] report a complete AO system on a Xilinx Virtex-5 FPGA. However, despite the advantages of such a system—including reduced cost, complexity and power usage—there are very few reported examples of such systems which handle both processing and control. This may be for a variety of reasons. FPGAs lack a fixed architecture, making interfacing with the WFS detector a complex process. Whilst intellectual property cores supporting industry standard interfaces such as Camera Link and GigE Vision exist, they are often expensive and inflexible. 'Soft-core' processors can be programmed into the PL to take advantage of manufacturer-provided drivers for operating systems such as Linux, but these are limited in power by low fabric clock speeds, which are on the order of 100 MHz [12]. Additionally, a historic shortcoming of FPGAs is that unlike DSPs [13] most did not have hardware support for floating-point operations. As recently as 2008 [14] FPGAs were practically limited to fixed-point or integer implementations, resulting in an undesirable closed-loop wavefront error due to the relatively poor precision of these data formats. Many of today's FPGAs contain optimised DSP slices in the PL [15], making it possible for them to perform all required operations in single- or double-precision floating-point with an acceptable throughput, and without sacrificing precision. Floating-point operations are still generally much slower than the equivalent fixed-point or integer operations, however.

In recent years, devices containing both hard-wired processing systems (PSs) and PL on the same chip have emerged. As the PS can be clocked at GHz frequencies, they have a distinct advantage over PL-only devices with soft-core processors. Meanwhile, compared to central processing unit-only systems, implementing AO data processing operations in the PL may result in a comparable bandwidth despite the relatively low PL clock speed. These devices often have a variety of I/O peripherals and support standard interfaces, such as Gigabit Ethernet, facilitating DM and WFS interfaces. By eliminating transmission delays arising from sending data to and from external devices such as frame grabbers, this alone could lead to a significant improvement in overall latency. The Xilinx Zynq-7000 FPGA, is an example of such a device, containing a dual-core ARM Cortex-A9 processor as well as PL with optimised DSP slices to accelerate floating-point and multiply-accumulate (MAC) operations [12]. In light of these features the Zynq was chosen as the platform for the presented system. The target used was the Zedboard, a development board for the Zynq-7000 7Z020, which is a low- to mid-range member of the device family.

3. AO SYSTEM DESIGN

The AOD is a closed-loop AO system, using a SH WFS for wavefront measurement and a DM for correction. The specifications of the AOD are shown in Table 1, and are the same as those used for the design presented here.

Table 1: AO specifications of the AOD [2, 16]

Parameter	AOD
Number of actuators	177
Number of subapertures (per side)	14
Number of subapertures (illuminated)	144
Pixels per subaperture (per side)	6
Number of slopes	288

Some of the operations required in an AO system are best suited to conventional processors whilst others can be best implemented in the PL. A diagram showing how a typical AO system might be implemented on the Zynq is shown in Fig. 1a. Top-level system control is handled by the AO System Control module inside the PS, which runs an embedded variant of the Linux kernel. This module oversees execution of the control loop. The WFS interface module, also implemented in the PS, communicates with the WFS camera and stores the detector image in memory in the PL. The operations required to go from the raw WFS detector image to DM commands are carried out in the PL to achieve maximum throughput via parallelisation and other optimisation methods unique to the FPGA architecture. This system is referred to as the Adaptive Optics Processing System (AOPS). The DM interface is also implemented in the PL to minimise the latency between computation of the DM commands and issuing them to the DM.

The AOPS, the focus on this paper, was divided into two modules so that each could be analysed and optimised individually before being combined into a single processing unit. The first module carries out thresholding and centroiding, and the second wavefront reconstruction. A data flow diagram of the AOPS is shown in Fig. 1b. As this is a proof-of-concept design the AOPS was kept as simple as possible. As a result a number of processing steps required in the AOD, including subtracting centroid offsets, removing tip/tilt/focus modes and more sophisticated thresholding methods were omitted. All operands are stored in block random-access memorys (BRAMs), memory cells located in the PL. A possible bottleneck in performance is caused by the storage requirements of the WFS detector image. Because the PS must be able to write to the BRAMs used to store it, only half of the total ports are available to the AOPS, effectively doubling access time. All other operands are only accessed from within the PL and so are not affected by this issue.

4. ALGORITHM OPTIMISATION

The configurable architecture of the PL in an FPGA allows algorithms to be optimised for high throughput. Pipelining and loop unrolling are two optimisation techniques which are relevant to those used in the AOPS. The nature of the operations required in the AOPS are best represented as nested loops. Loop unrolling is an optimisation technique which increases the throughput of loops by instantiating multiple copies of the loop body in the PL [17]. Consider the function `foo` shown in Fig. 2a. For the sake of simplicity, assume that each operation requires one clock cycle. When the loop is rolled, only one instance of the logic used to carry out a single loop iteration exists. As a result each iteration must be executed sequentially, giving a total latency of 12 cycles: this is similar to how the program would be executed in a conventional processor. Unrolling by a factor of $F = 2$ creates two such instances, enabling two iterations to execute concurrently. Completely unrolling the loop, corresponding to $F = 4$, creates four instances, enabling the entire loop to be executed in as much time as a single iteration. In the AOPS, loop iterations involve accessing sequential elements of arrays in BRAM, which at most have two read/write ports. As a result full unrolling of such a loop with n iterations is only effective if the array can be partitioned into $n/2$ BRAMs. For this example, if the loop is completely unrolled, two BRAMs are required. For large arrays such as the reconstructor matrix this is not possible as there are insufficient resources in the chip; in these instances, partial unrolling and the corresponding partitioning arrangement are used.

Another optimisation technique that is used in conventional processors and FPGAs alike is pipelining, in which a task is split into segments which use different hardware resources so that a new task can begin execution before the

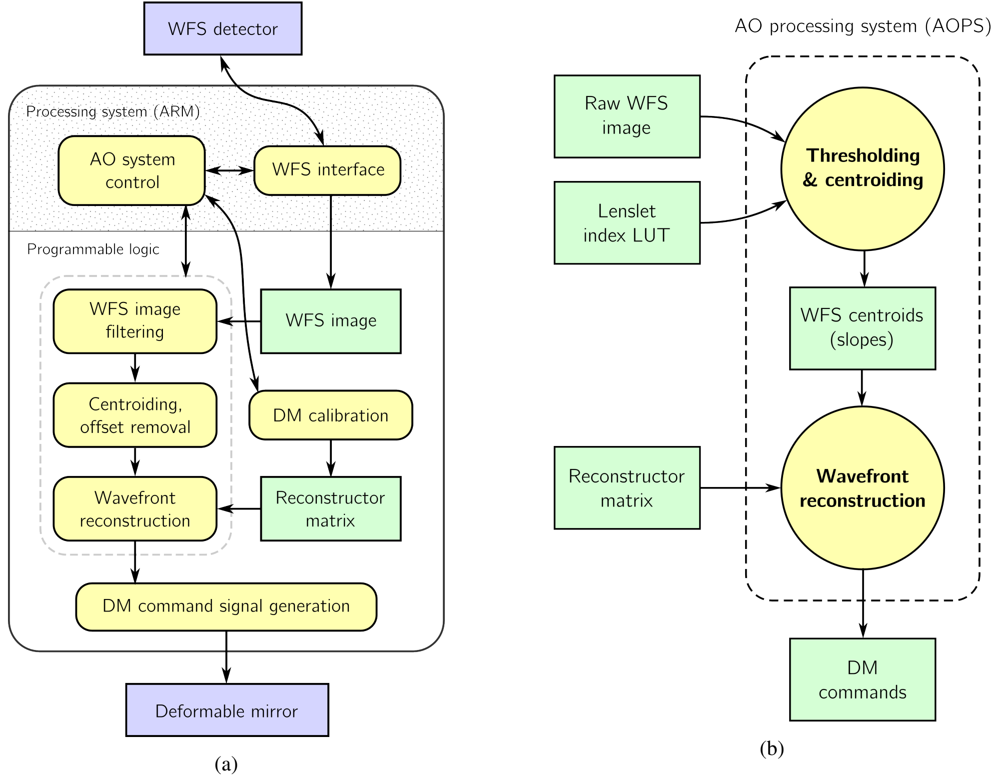


Fig. 1: (a) Diagram showing the top-level modules of the AO system including the AOPS (comprising the modules inside the dotted line). Green cells represent BRAMs, yellow cells processing modules and blue hardware; (b) A data flow diagram of the AOPS.

previous one finishes [17]. A diagram illustrating the concept applied to a function `bar` is shown in Fig. 2b. By enabling new calls of `bar` before previous calls have finished, the latency of successive function calls is significantly reduced. Loop bodies can also be pipelined to achieve a similar result.

5. MODULE OPTIMISATION

The Thresholding and Centroiding and Wavefront Reconstruction modules were designed and optimised individually before being combined to form the AOPS. The only variables with data types are constrained by the AOD requirements are the WFS detector image, with pixel values specified as 14-bit unsigned integers, and the DM commands, which are specified as 16-bit integers. The formats of the reconstructor matrix, WFS slopes and all other variables are design parameters which affect the achievable bandwidth and residual error of the system. Floating-point implementations were used, with fixed-point only considered only if latency requirements could not be met.

The software for each module was developed using Xilinx Vivado High-Level Synthesis (HLS), a utility which automatically translates C/C++ code into a register transfer level (RTL) implementation that is used to program the PL. HLS supports a number of relevant optimisation techniques including pipelining, loop unrolling and array partitioning which are effected during code transformation. The tool also provides latency and resource usage estimates, including the number of BRAMs required to store variables internal to the function, DSP slices, flip-flops (FFs) and look-up-tables (LUTs). Thes estimates were used in the analysis to determine the implementation with the lowest latency that could realistically be implemented on the Zynq. Some resources are required by the PS in order to communicate with the PL, which are not included in these estimates. It was found, however, that this amounts to fewer than 1% of the total resources available in the Zynq 7Z020, and so was ignored in the analysis.

The RTL implementation is dependent upon the fabric clock frequency f_{clk}, as HLS attempts to constrain the latency of the longest interconnect $T_{\min}$ in the design to below this clock period [17], with the design failing timing constraints if $T_{\min} > 1/f_{\text{clk}}$. The more resources used by the design, the less likely it is to meet timing requirements,

```
void foo(int a[4], int b[4], int c[4])
{
        int i;
        for (i = 0; i < 4; i++)
        {
            a[i] = 4 * i;
            b[i] = i / 2;
            c[i] = i + 5;
        }
}
```

```
int tmp;
void bar(int *x, int *y)
{
    tmp = *x;       // Read
    tmp = tmp + 5; // Add
    *y = tmp;       // Write
}
```

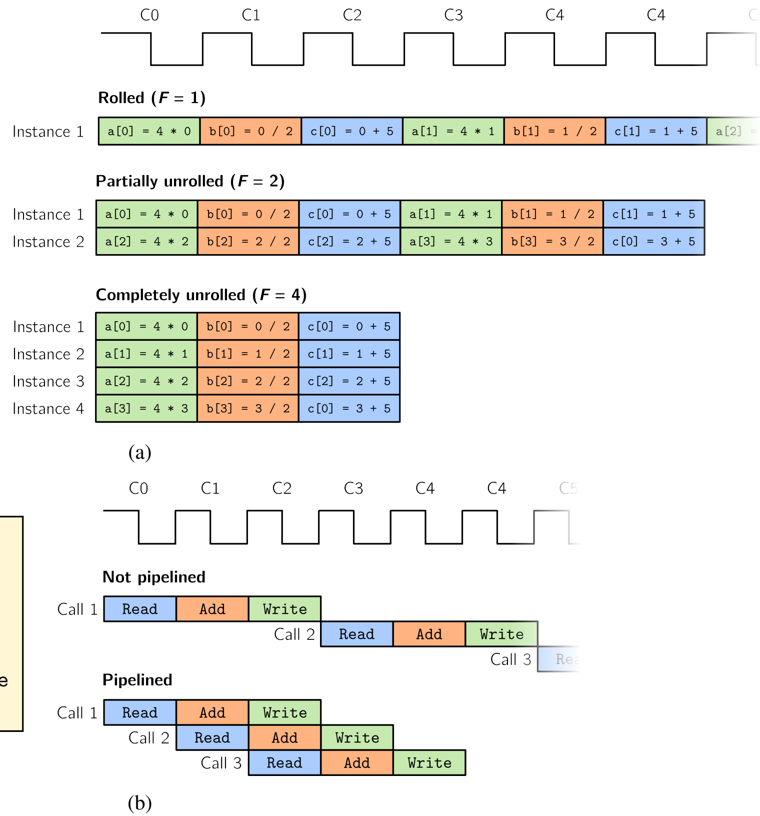


Fig. 2: Visualisations of unrolling (a) and pipelining (b) as applied to functions `foo` and `bar` respectively.

simply because clock signals must travel longer paths. A solution is to decrease f_{clk}; however this increases the overall latency. Hence, designs using less logic were deemed more favourable. Nonetheless, it should be stressed that the pass/fail criterion given by HLS is only an estimate, and whether or not a design will meet timing requirements when implemented on the FPGA is only determined during placing and routing. A fabric clock speed of $f_{\text{clk}} = 100$ MHz was used in the synthesis of all designs, and designs with $T_{\text{min}} > 10$ ns fail to meet timing constraints.

The design process of each module was as follows:

1. A top-level function representing the module was written in C++, with function arguments representing inputs and outputs.
2. After validation using testbench code, Vivado HLS was used to synthesise the function into the equivalent RTL implementation, and the estimated latency, T_{min} and resource usage recorded.
3. The RTL implementation was validated by performing a cosimulation between the implementation and the C++ testbench code containing the original function using a gate-level simulation tool.
4. Optimisation directives were applied, and steps 2 and 3 repeated.

5.1 Thresholding and Centroiding Module

This module applies a simple constant threshold filter to the WFS detector image before computing the centroids of each subaperture, or the x and y slopes $\bar{x}$ and $\bar{y}$, stored in a 288×1 vector $\mathbf{s}$. The two operations were combined as they both involve visiting each relevant pixel in the image. A simple centre-of-gravity centroiding technique was used, in which the centroids are calculated using

$$(\bar{x}, \bar{y}) = \left(\frac{M_{01}}{M_{00}}, \frac{M_{10}}{M_{00}} \right) \tag{1}$$

where the moments M_{mn} are calculated using the expressions

$$M_{mn} = \sum_{m} \sum_{n} x^m y^n I(x, y) \tag{2}$$

and I is the detector image, represented in memory as a 2D array in row-major order. The chosen method of calculating the WFS slopes, shown in Algorithm 1, involves nested loops. The outermost loop, `TC_loop_lenslets`, loops through the coordinates of the top-left-hand pixel of each illuminated subaperture k using 2×144 look-up table. The nested loops `TC_loop_rows` and `TC_loop_cols` then iterate over each pixel in the subaperture, performing the thresholding operation and updating the partial sums of the computed image moments M_{00_k}, M_{01_k} and M_{10_k} using a MAC operation. Two division operations then calculate the slopes $\bar{x}_k$ and $\bar{y}_k$.

Algorithm 1 The Thresholding and Centroiding module.

```
TC_loop_lenslets:
for each subaperture k do
TC_loop_rows:
    for each subaperture row y do
TC_loop_cols:
        for each subaperture column x do
            Threshold I(x, y)
            Update the partial sums of M_00k, M_01k and M_10k
        end for
    end for
    Compute & store x̄_k and ȳ_k
end for
```

The module was synthesised in HLS using a single-precision floating-point vector to represent $\mathbf{s}$. The resulting latencies and resource usage obtained by applying different optimisation strategies are shown in Table 3. The labels used to refer to each optimisation strategy are given in Table 2.

Table 2: Notation for the different optimisation configurations used in the thresholding and centroiding module. The BRAM is required to store the LUT.

Optimisation	**Label**
Pipelined `TC_loop_rows`	TC.P1
Pipelined `TC_loop_cols`	TC.P2
Unrolled `TC_loop_cols`, factor F	TC.UF
WFS detector image partitioned cyclically, factor F	TC.FF

The throughput can be increased by a factor of six if `TC_loop_cols` is completely unrolled (TC.U6) whilst pipelining `TC_loop_rows` (TC.P1). This was achieved by partitioning I cyclically such that every 6th column was stored in the same BRAM. It was found, however, that this arrangement failed timing requirements. As a result, TC.P2 and TC.P1 were chosen as the optimal configurations as they have similar latencies.

[1]The uncertainty in the latency is due to `if`-statements in the body of the `for`-loop, because HLS assumes that the result of the condition may change during run-time, even though execution of this function is in fact completely deterministic.

Table 3: Estimated latency of the RTL implementation of the Thresholding and Centroiding module (Algorithm 1) produced in Vivado HLS using different optimisation techniques in single-precision floating-point. Bold entries denote implementations that meet timing requirements with the best combination of latency and resource usage, and red entries implementations which fail timing requirements at f_{clk} =100 MHz.

			Resource usage			
Optimisation	**Latency** (cycles)	T_{min} (ns)	BRAM (18 Kbit)	DSP slices	FFs	LUTs
None	22897 – 136945[1]	8.70	1	3	3039	4060
TC.P1	**5213**	**9.09**	**1**	**4**	**1887**	**2532**
TC.P1, TC.U6, TC.F6	913	13.1	1	10	3979	5544
TC.P2	**5212**	**9.09**	**1**	**3**	**3011**	**4250**
		Available	280	220	106400	53200

5.2 Wavefront Reconstruction Module

The Wavefront Reconstruction module is used to generate the 177 $\times 1$ vector $\mathbf{c}$ of DM commands given the 288×1 vector $\mathbf{s}$ of WFS slopes output by the combined thresholding and centroiding module using the expression

$$\mathbf{c} = \mathbf{M}\mathbf{s} \tag{3}$$

where $\mathbf{M}$ is the 177×288 reconstructor matrix and the ith actuator command c_i is given by the dot product of the ith row of $\mathbf{M}$ and the slopes vector:

$$c_i = \sum_{j=1}^{288} M_{ij} s_j \tag{4}$$

where M_{ij} is the element of $\mathbf{M}$ at indices (i, j). To implement leaky-integrator control, the commands are updated at intervals Δt using

$$\mathbf{c}(t + \Delta t) = K\mathbf{c}(t) - \mathbf{M}\mathbf{s}(t) \tag{5}$$

where $0 \leq K \leq 1$ is the leak coefficient. Algorithm 2 was used to perform the matrix-vector multiplication. A temporary variable is used to store the partial sum of each DM command $\mathbf{c}_i$ in each iteration of `WFR_loop_cols` in order to avoid unnecessary BRAM accesses. After this loop exits, the actuator command is calculated using equation 5.

Algorithm 2 The wavefront reconstruction algorithm used.

```
WFR_loop_rows:
for Each actutaor i do
WFR_loop_cols:
    for Each subaperture j do
        Update the partial sum of c_i
    end for
    Compute & store actuator command c_i
end for
```

A single-precision floating-point implementation was applied to determine the most efficient method of optimising the nested loop structure. The optimisation strategies applied are shown in Table 4. As shown in Table 5, the lowest latency was achieved when `WFR_loop_rows` was pipelined (WFR.P1) and `WFR_loop_cols` was unrolled by a factor F =2 (WFR.U2, taking advantage of 2-port BRAM) but timing requirements are not met, as $T > 1/f_{\text{clk}}$.

A lower latency can be achieved by full unrolling of `WFR_loop_cols` (WFR.U288). This, however requires concurrent access to each column of M and each element of $\mathbf{s}$, in turn requiring at least 288 BRAMs, more than is available on the 7Z020, as well as 288 instances of the loop logic. As a compromise, `WFR_loop_cols` can be partially unrolled by a factor F, requiring M and $\mathbf{s}$ to be partitioned into F BRAMs. The latency and resource usage of this configuration for different F with `WFR_loop_rows` pipelined are shown in Fig. 3a and Fig. 3b respectively. As F

Table 4: Notation for the different optimisation configurations used in the Wavefront Reconstruction module.

Optimisation	**Label**
Pipelined `WFR_loop_rows`	WFR.P1
Pipelined `WFR_loop_cols`	WFR.P2
Unrolled `WFR_loop_cols`, factor F	WFR.UF
Reconstructor matrix & WFS slopes partitioned, factor F	WFR.FF

Table 5: Estimated latency of the RTL implementation of the WFR module (see Algorithm 2) using different optimisation techniques with a single-precision floating-point implementation. Bold entries denote implementations that meet timing requirements with the best combination of latency and resource usage, and red entries implementations which fail timing requirements at $f_{\text{clk}} = 100$ MHz.

			Resource usage			
Optimisation	**Latency** (cycles)	T_{min} (ns)	BRAM (18 Kbit)	DSP slices	FFs	LUTs
None	511354	8.41	0	5	557	909
WFR.P1, WFR.U2, WFR.F2	26796	10.12	0	12	11687	19962
WFR.P1, WFR.U6, WFR.F6	**5676**	**9.35**	**0**	**62**	**16228**	**27986**
WFR.P1, WFR.U12, WFR.F12	**3564**	**9.35**	**0**	**122**	**22692**	**44724**
WFR.P2	203918	8.63	0	605	1038	8.63
		Available	280	220	106400	53200

increases, diminishing returns in terms of latency are achieved, whilst the required resources increases approximately linearly. Although $F = 32$ and $F = 16$ result in the highest throughput, in practice they cannot be realised due to their high resource usage. $F = 6$ and $F = 12$ instead were chosen as they achieve a good balance between both metrics.

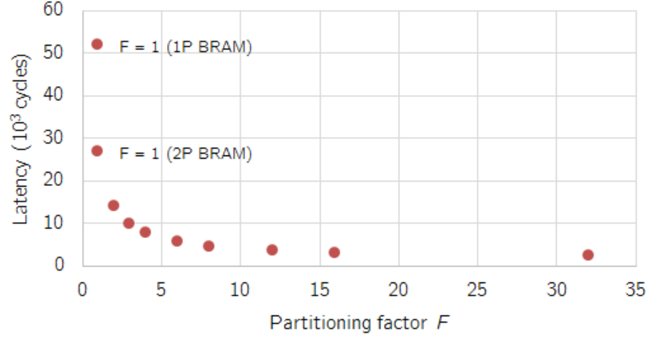
6. INTEGRATED ADAPTIVE OPTICS PROCESSING SYSTEM (AOPS)

The AOPS was formed by integrating the two processing modules. The resulting latency estimates are shown in Table 6 and the resource usage in Fig. 4a.

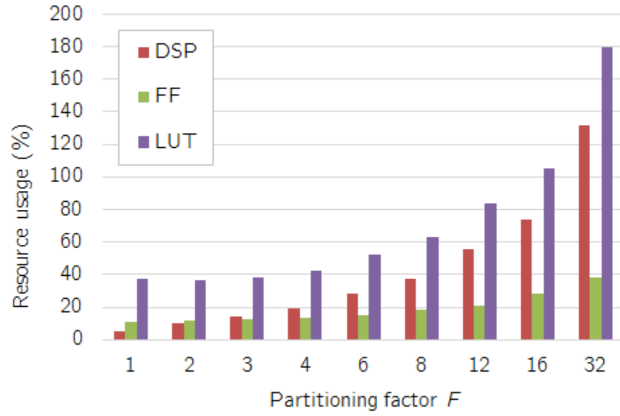
Table 6: Estimated latency of the RTL implementation of the integrated AOPS synthesised in Vivado HLS using different optimisation techniques with a single-precision floating-point implementation. Bold entries denote implementations that meet timing requirements with the best combination of latency and resource usage, and red entries implementations which fail timing requirements (at $f_{\text{clk}} = 100$ MHz unless otherwise specified).

Optimisation			
TC module	WFR module	**Latency** (cycles)	T_{min} (ns)
None	None	535115 – 649163	8.70
TC.P2	WFR.P1, WFR.F6	16905	9.35
TC.P1	**WFR.P1, WFR.F6**	**10913**	**9.35**
TC.P1	WFR.P1, WFR.F12	8789	9.35
TC.P1, TC.U6, TC.F6 ($f_{\text{clk}} = 100$ MHz)	WFR.P1, WFR.F6	7476	10.18
TC.P1, TC.U6, TC.F6 ($f_{\text{clk}} = 66.67$ MHz)	WFR.P1, WFR.F6	7180	13.33

Of the implementations that satisfy timing constraints, the best is a combination of TC.P1, WFR.P1 and WFR.F12, being theoretically able to be executed at a rate of 11.4 kHz. This design has a very high resource usage, as shown in Fig. 4a; as a result, the same design but with WFR.F6 is chosen instead, capable of executing at roughly 9.1 kHz, in order to leave space in the fabric for any other logic which may be required, such as the DM calibration and command signal generation modules. Ignoring timing constraints the fastest design comprises a combination of TC.P1, TC.F6, WFR.P1 and WFR.F6. Although it was found that this design did not meet timing constraints when



(a)



(b)

Fig. 3: Latency (a) and resource usage (b) of the RTL implementation of algorithm 2 with `WFR_loop_rows` pipelined (WFR.P1) and `WFR_loop_cols` unrolled (WFR.UF) with different partitioning/unrolling factors F for the reconstructor matrix $\mathbf{M}$ and WFS slope vector $\mathbf{s}$.

f_{clk} = 100 MHz, it is able to run with $f_{\text{clk}} = 66.67$ MHz at approximately 9.3 kHz. However compared to the design in which only `TC_loop_rows` is pipelined (TC.P1) and the WFS image is not partitioned, the performance gains are relatively small. The latter configuration is preferable simply because it does not require partitioning of the WFS detector image, in turn requiring fewer BRAMs and less complicated software for the PS.

These results indicate that, for the AOD at the very least, there is no need to use an integer or fixed-point implementation, as the AOPS can perform all required processes in single-precision floating-point at a rate allowing ample latency for both the generation of the DM command signals and the acquisition of the WFS image from the detector. In fact, it was found that even using a double-precision implementation, bandwidth requirements could be comfortably met, as shown in table 7. Although resource usage proved to be prohibitive for most implementations (see Fig. 4b) this could be avoided by using a higher-end member of the Zynq-7000 device family with a larger PL area.

Table 7: Estimated latency of the RTL implementation of the integrated AOPS synthesised in Vivado HLS using different optimisation techniques with a double-precision floating-point implementation. Bold entries denote implementations that meet timing requirements with the best combination of latency and resource usage, and red entries implementations which fail timing requirements (at $f_{\text{clk}} = 100$ MHz unless otherwise specified).

Optimisation			
TC module	WFR module	**Latency** (cycles)	$T_{\min}$ (ns)
None	None	693227 – 807275	8.70
TC.P2	WFR.P1, WFR.F6	16099	9.17
TC.P1	WFR.P1, WFR.F6	10917	9.17
TC.P1	WFR.P1, WFR.F12	8793	9.17
TC.P1, TC.U6, TC.F6 ($f_{\text{clk}} = 100$ MHz)	WFR.P1, WFR.F6	7480	10.18
TC.P1, TC.U6, TC.F6 ($f_{\text{clk}} = 66.67$ MHz)	WFR.P1, WFR.F6	7467	13.72

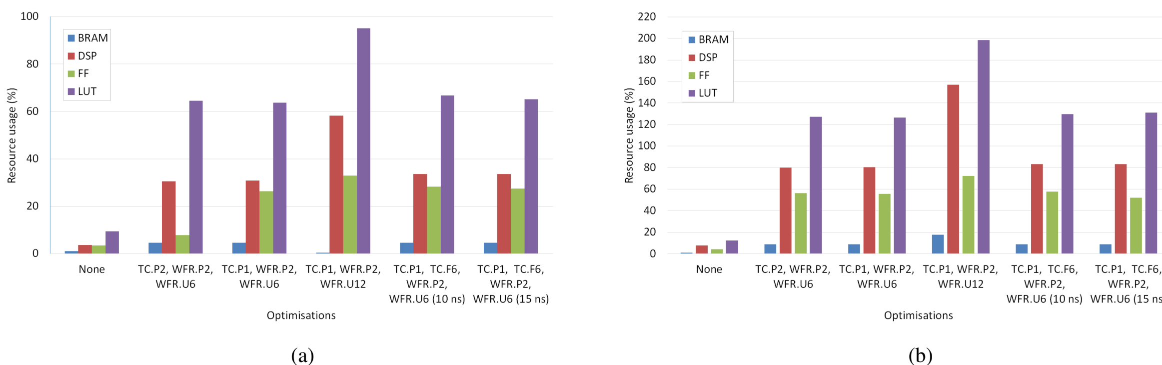


Fig. 4: Resource usage of different implementations of the AOPS with different applied optimisation techniques, using single-precision (a) and double-precision (b) floating-point format.

7. CONCLUSION

Recent advances in FPGA technology have paved the way for a complete AO system to be controlled by a single device, providing a cheaper, simpler and smaller alternative to existing systems. A simplified AO processing system handling the transformation between the WFS detector image and the DM commands, the Adaptive Optics Processing System (AOPS), was implemented on an FPGA, using the specifications of the Adaptive Optics Demonstrator (AOD). It was found that SH WFS thresholding and centroiding and wavefront reconstruction could be carried out in single-precision floating-point at over 9 kHz, leaving ample latency to accommodate the DM and WFS interfaces whilst meeting the $\sim$100 Hz bandwidth requirement of the AOD. This was achieved by taking advantage of parallelisation and other optimisation techniques unique to the FPGA architecture. The next stage is to build upon the presented findings by designing and validating a standalone closed-loop AO system based using the same technology, comprising a SH WFS and a DM, in order to determine the performance limits of FPGAs in this application. This will first involve analysis of the throughput that can be achieved between the WFS and the PS in the FPGA followed by development of the DM interface. The capabilities of the AOPS will also be extended to performing more sophisticated operations, such as centroid offsetting and removal of low-order aberrations such as tip, tilt and focus modes, to more accurately represent the processing needs of existing AO systems. It is hoped that in the future this design can be developed into a standalone AO system controller for use in general AO applications, such as in 1–3 m telescopes for space surveillance, or even for amateur astronomy.

8. ACKNOWLEDGEMENTS

The authors of this paper acknowledge funding for this research project from the Cooperative Research Centre for Space Environment Management (the Space Environment Research Centre (SERC) Limited). The authors would also

like to thank David Johnson for his kind assistance with Xilinx software and hardware.

REFERENCES

1. D. J. Kessler and B. G. Cour-Palais, “Collision Frequency and Artificial Satellites: The Creation of a Debris Belt,” *Journal of Geophysical Research*, vol. 83, pp. 2637–2646, June.
2. F. Bennet, C. D’Orgeville, Y. Gao, W. Gardhouse, N. Paulin, I. Price, F. Rigaut, I. T. Ritchie, C. H. Smith, K. Uhlendorf, and Y. Wang, “Adaptive optics for space debris tracking,” in *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, vol. 9148 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, p. 1, July 2014.
3. S. Mauch and J. Reger, “Real-time spot detection and ordering for a shack-hartmann wavefront sensor with a low-cost fpga,” *Instrumentation and Measurement, IEEE Transactions on*, vol. 63, pp. 2379–2386, Oct 2014.
4. A. Basden, D. Geng, R. Myers, and E. Younger, “Durham adaptive optics real-time controller.,” *Applied optics*, vol. 49, no. May, pp. 6354–6363, 2010.
5. A. Moore, *FPGAs for Dummies*. Wiley & Sons, 2014.
6. S. Mauch, J. Reger, C. Reinlein, M. Appelfelder, M. Goy, E. Beckert, and a. Tünnermann, “FPGA-accelerated adaptive optics wavefront control,” *Proc. SPIE*, vol. 8978, p. 897802, 2014.
7. G. Hovey, R. Conan, F. Gamache, G. Herriot, Z. Ljusic, D. Quinn, M. Smith, J. Veran, and H. Zhang, “An FPGA Based Computing Platform for Adaptive Optics Control,” *1st AO4ELT conference - Adaptive Optics for Extremely Large Telescopes*, vol. 07006, p. 07006, 2010.
8. K. Kepa, D. Coburn, J. Dainty, and F. C. Morgan, “High Speed Optical Wavefront Sensing with Low Cost FPGAs,” *Measurement Science Review*, vol. 8, no. 4, pp. 87–93, 2008.
9. K. Richards, “Adaptive optics real time processing design for the advanced technology solar telescope,” *Proceedings of SPIE*, vol. 8447, pp. 84472N–84472N–9, 2012.
10. M. Thier, R. Paris, T. Thurner, and G. Schitter, “Low-Latency Shack-Hartmann Wavefront Sensor Based on an Industrial Smart Camera,” *IEEE Transactions on Instrumentation and Measurement*, vol. 62, no. 5, pp. 1–9, 2012.
11. C. Y. Chang, B. T. Ke, H. W. Su, W. C. Yen, and S. J. Chen, “Easily implementable field programmable gate array-based adaptive optics system with state-space multichannel control,” *Review of Scientific Instruments*, vol. 84, no. 2013, 2013.
12. Xilinx, Inc., *Zynq-7000 All Programmable SoC Overview*, December 2013.
13. S. W. Smith, *The Scientist and Engineer’s Guide to Digital Signal Processing*. California Technical Publishing, 1998.
14. Y. Martín-Hernando, L.-F. Rodríguez-Ramos, and M. R. Garcia-Talavera, “Fixed-point vs. floating-point arithmetic comparison for adaptive optics real-time control computation,” *Proc. SPIE*, vol. 7015, pp. 70152Z–70152Z–12, 2008.
15. T. Vanevenhoven, “High-Level Implementation of Bit- and Cycle-Accurate Floating-Point DSP Algorithms with Xilinx FPGAs,” October 2011.
16. C. D’Orgeville, F. Bennet, M. Blundell, R. Brister, A. Chan, M. Dawson, Y. Gao, N. Paulin, I. Price, F. Rigaut, I. Ritchie, M. Sellars, C. Smith, K. Uhlendorf, and Y. Wang, “A sodium laser guide star facility for the ANU/EOS space debris tracking adaptive optics demonstrator,” *SPIE Astronomical Telescopes and Instrumentation*, p. 91483E, 2014.
17. Xilinx, Inc., *Vivado Design Suite User Guide*, May 2014.