Cambridge University Engineering Department
Engineering Tripos Part IIA
PROJECTS: Interim and Final Report Coversheet

IIA Projects

TO BE COMPLETED BY THE STUDENT(S)

Project:	GB3 – RISC-V Processor		
Title of report:	Individual Report		
Name(s): (capitals)		crsID(s):	College(s):
WILL HEWES		wh365	Pembroke

Declaration for: Interim Report 2

I confirm that, except where indicated, the work contained in this report is my own original work.

Instructions to markers of Part IIA project reports:

Grading scheme

Grade	A*	A	B	C	D	E
Standard	Excellent	Very Good	Good	Acceptable	Minimum acceptable for Honours	Below Honours

Grade the reports by ticking the appropriate guideline assessment box below, and provide feedback against as many of the criteria as are applicable (or add your own). Feedback is particularly important for work graded C-E. Students should be aware that different projects and reports will require different characteristics.

Penalties for lateness: Interim Reports: 3 marks per weekday; Final Reports: 0 marks awarded – late reports not accepted.

Guideline assessment (tick one box)

A*/A	**A/B**	**B/C**	**C/D**	**D/E**

Marker:		Date:	

Delete (1) or (2) as appropriate (for marking in hard copy – different arrangements apply for feedback on Moodle):

(1) Feedback from the marker is provided on the report itself.

(2) Feedback from the marker is provided on second page of cover sheet.

	Typical Criteria	Feedback comments
Project Skills, Initiative, Originality	Appreciation of problem, and development of ideas	
	Competence in planning and record-keeping	
	Practical skill, theoretical work, programming	
	Evidence of originality, innovation, wider reading (with full referencing), or additional research	
	Initiative, and level of supervision required	
Report	Overall planning and layout, within set page limit	
	Clarity of introductory overview and conclusions	
	Logical account of work, clarity in discussion of main issues	
	Technical understanding, competence and accuracy	
	Quality of language, readability, full referencing of papers and other sources	
	Clarity of figures, graphs and tables, with captions and full referencing in text	

ENGINEERING TRIPOS PART IIA

GB3 - Risc-V Processor

Second Interim Report

Group 4 - Resource Usage

Will Hewes - wh365
Pembroke College

Contents

1 Introduction

This project involves the collaborative design and implementation of a RISC-V processor on an iCE40 FPGA device. The overarching aim is to optimise the processor design with regard to its performance, power dissipation, and resource usage, while balancing trade-offs between these three aspects. Each team member is responsible for focusing on one of the three aspects. My reports will focus on the resource usage of the design.

This report outlines the initial work carried out to reduce resource usage in the RISC-V processor implementation. The goal has been to reduce the number of LUTs and flip-flops consumed by the processor while minimising excessive power dissipation or performance degradation. These modifications form part of a broader group effort targeting performance, power consumption, and resource usage. Coordination has been ongoing to ensure our changes are compatible and contribute towards a coherent final design.

The analysis has concerned the `bubblesort` program due to its increased complexity relative to `hardwareblink` and `softwareblink`, allowing the processor to be pushed further and offering more representative resource utilisation patterns. Preliminary synthesis results indicated that the original design made no use of available DSP blocks, instead relying heavily on general-purpose LUTs for arithmetic operations.

2 Preliminary Results

All tables provided will be in order of the changes that were made and displayed in the Appendix. The total reduction in resource usage is shown in Table 1.

2.1 Baseline

Having performed the baseline analysis in the previous interim report, the resource usage for the provided processor on the `bubblesort` algorithm is shown in Table 2 the Appendix.

In the baseline design, synthesis of the unmodified processor design showed that approximately 58% of the available logic cells on the iCE40 UP5K were in use, along with two thirds of the available Block RAMs. This figure indicates limited headroom for further feature integration or pipeline complexity, motivating efforts to streamline key datapaths and control logic. In particular, excessive use of general-purpose logic for ALU operations and CSR registers, along with unnecessarily wide control signal buses, were identified as contributors to high logic utilisation. Forwarding logic and arithmetic duplication remain areas for future optimisation. Additionally, the design made no use of the available DSPs, instead implementing its adder functions through LUTs. These observations guided the subsequent design changes detailed in this report.

2.2 Program Counter Clock Gating

To reduce unnecessary switching activity in the control path, support for gated updates to the program counter was introduced. This involved modifying the `program_counter.v` module to accept a `write_enable` signal, allowing updates to be conditionally applied only when pipeline progression is required. The cpu.v module was updated accordingly to assert `write_enable` whenever the stall signal is low, ensuring that the PC is held stable during pipeline stalls.

This change helps avoid logic toggling during idle or stalled cycles. This primarily has an impact on the power consumption, allowing the board to reduce unnecessary dynamic power dissipation caused by repeated switching in the program counter logic during pipeline stalls. The resource utilisation can be seen in Table 3 in the Appendix, helping free up 9 logic cells. Although the impact on LUT usage is minimal, this is part of a broader strategy to reduce switching by selectively enabling pipeline register updates.

2.3 Removal of CSR Logic

One of the most significant changes made was the complete removal of the Control and Status Register (CSR) logic. Control and Status Registers are optional RISC-V components used to support operating system-level features such as interrupt handling, timer access, and machine state tracking.

In the baseline design, CSR handling added extra complexity to the decode stage, required new control signals for accessing CSRs, and introduced a dedicated register file to store CSR values. This register file was implemented using block RAM, which increased resource usage. As CSR instructions are not required for correct operation at this level of implementation, this functionality could be removed for the purposes of resource optimisation. Since the benchmark programs execute purely in machine mode without requiring interrupts, exceptions, or privileged features, no CSR-related mechanisms are used, and their removal does not affect functional correctness.

The removal involved disabling CSR detection in `control_unit.v` and replacing related wires with fixed constants. Registers previously driven by CSR outputs were redirected to bypass logic or default values, ensuring that the pipeline remained functionally correct without CSR support.

This change resulted in a notable reduction in logic complexity and created a marked drop in block RAM usage. As seen in Table 4 in the Appendix, the logic cell usage dropped from 3064 to 2905, and the RAM usage dropped from 20 to 12, freeing up an additional 26% of the available block RAM. It also simplifies the decode and execution stages, reducing combinational logic depth. However, it does remove support for future system-level extensions that rely on CSR functionality, and would need to be revisited if such features are later required.

2.4 Trimming Control Signal Width

While reviewing how the processor decodes instructions and carries them out, a closer examination of the logic and signals involved in these stages was performed to find areas where resources could be saved. It was identified that the control signal wire `cont_mux_out` was defined as a 32-bit bus, despite only 11 bits being used downstream. To simplify the design and reduce unnecessary routing, the width was reduced to match actual usage, and the custom `mux2to1_11bit` module was added to represent this.

This change eliminates unused signal bits, improving signal clarity and slightly reducing synthesis complexity. As seen in Table 5 in the Appendix, the logic cell usage dropped from 2905 to 2891. This contributes to improved maintainability and more efficient use of the FPGA's routing resources. Removing unused logic also helps the synthesiser more aggressively optimise signal propagation and packing. This change is relatively minor in terms of immediate resource savings, but it illustrates a useful strategy for streamlining the design. By tightening signal widths and removing unused control bits, routing congestion can be reduced and synthesis complexity lowered. While the impact is small, applying this kind of low-level refinement across the processor can contribute meaningfully to overall optimisation. However, identifying such inefficiencies is time-consuming, as it often requires manual tracing of signals across modules and pipeline stages.

3 Potential Risks

While the changes implemented so far have improved logic utilisation without degrading performance significantly, several risks remain that could impact the stability or scalability of the processor design.

Firstly, future attempts to offload arithmetic operations to DSP blocks may reduce LUT usage, but will require careful mapping to avoid bottlenecks or timing degradation. Any overuse of these blocks could limit parallelism or necessitate costly logic workarounds. As of this stage, no DSP blocks are in use.

The most significant change to the implementation was the removal of CSR logic. Though this has a very positive impact on the LUT usage and in particular on the reduction of block RAMs required in the design, it has implications with regard to future extensibility and system-level compatibility. By removing CSR support, the processor is no longer capable of handling interrupts, errors, or access control, which limits its applicability in more complex applications such as systems that use timers, interrupts, or need to manage multiple tasks

Lastly, clock gating introduced in the pipeline registers assumes correct stalling logic. If the stall signal is incorrectly asserted or delayed, it could lead to data hazards, incorrect register values, or pipeline deadlock. This risk increases as the pipeline control logic becomes more intricate.

Continued validation through simulation and synthesis will be necessary to ensure that further optimisation efforts do not compromise correctness or long-term maintainability.

4 Future Work

Several areas remain that could be explored to further reduce resource usage:

- Simplifying large logic blocks using `generate` statements in Verilog to reduce duplicated code.
- Reducing repeated comparator logic in the forwarding unit by reusing shared comparison circuits.
- Using the FPGA's DSP blocks for arithmetic operations to free up logic cells and improve efficiency.
- Cleaning up and simplifying control signal wiring around the register file.

Each of these changes will be assessed for impact on logic usage and the other performance metrics.

5 Conclusion

This report outlines a series of logic and memory optimisations applied to the baseline RISC-V processor design. Through targeted changes - clock gating, CSR removal, and control signal width trimming - a cumulative reduction of 182 logic cells and 8 block RAMs was achieved without degrading functionality. These gains improve the design's flexibility for future performance or power-directed improvements.

The most immediate modification planned is to shift the adder operation to DSP blocks, which is expected to free up LUTs, reduce routing congestion, and improve overall timing performance.

A Resource Usage Data

Modification	LUTs	Block RAMs
Baseline → PC Gating	9	0
PC Gating → CSR Removal	159	8
CSR Removal → Signal Width Fix	14	0
Total Reduction	182	8

Table 1: Individual resource reductions following each modification

Resource type	Used	% of total
Logic cells (`ICESTORM_LC`)	3073 / 5280	58 %
Block RAMs (`ICESTORM_RAM`)	20 / 30	66 %
IO buffers (`SB_IO`)	8 / 96	8 %
Global buffers (`SB_GB`)	5 / 8	62 %
HF oscillators (`ICESTORM_HFOSC`)	1 / 1	100 %

Table 2: Baseline report

Resource type	Used	% of total
Logic cells (`ICESTORM_LC`)	3064 / 5280	57 %
Block RAMs (`ICESTORM_RAM`)	20 / 30	66 %
IO buffers (`SB_IO`)	8 / 96	8 %
Global buffers (`SB_GB`)	5 / 8	62 %
HF oscillators (`ICESTORM_HFOSC`)	1 / 1	100 %

Table 3: Report after adding clock gating to the Program Counter module

Resource type	Used	% of total
Logic cells (`ICESTORM_LC`)	2905 / 5280	57 %
Block RAMs (`ICESTORM_RAM`)	12 / 30	40 %
IO buffers (`SB_IO`)	8 / 96	8 %
Global buffers (`SB_GB`)	5 / 8	62 %
HF oscillators (`ICESTORM_HFOSC`)	1 / 1	100 %

Table 4: Report after removing Control and Status Register (CSR) logic

Resource type	Used	% of total
Logic cells (`ICESTORM_LC`)	2891 / 5280	57 %
Block RAMs (`ICESTORM_RAM`)	12 / 30	40 %
IO buffers (`SB_IO`)	8 / 96	8 %
Global buffers (`SB_GB`)	5 / 8	62 %
HF oscillators (`ICESTORM_HFOSC`)	1 / 1	100 %

Table 5: Report after removing unused control signal width

ENGINEERING TRIPOS PART IIA

GB3 - Risc-V Processor

First Interim Report

Group 4 - Resource Usage

Will Hewes - wh365
Pembroke College

Contents

1 Introduction

This project involves the collaborative design and implementation of a RISC-V processor on an iCE40 FPGA device. The overarching aim is to optimise the processor design with respect to its performance, power dissipation, and resource usage, while balancing trade-offs between these three aspects. Each team member is responsible for focusing on one of the three aspects. My reports will focus on the resource usage of the design.

This interim report characterises the baseline design. We are evaluating three provided example programs - `hardwareblink`, `softwareblink`, and `bubblesort` - to establish benchmarks for the logic usage, power dissipation and timing. These programs utilise the processor and surrounding hardware to varying degrees, and provide a baseline against which future design improvements can be compared.

2 Project Specification

Within the team, my specific task focuses on optimising the resource usage of the FPGA implementation. The goal of this optimisation is to reduce the number of Look-Up Tables (LUTs), flip-flops, and logic elements used in the design. This allows more headroom on the device for future expansions and increases its scalability.

LUTs and flip-flops are some of the fundamental building blocks of digital logic in an FPGA. A LUT is a configurable logic element that can implement arbitrary combinational logic functions by storing precomputed outputs for each possible input combination. Flip-flops are used to store single bits of state and are essential for implementing sequential logic, such as registers and control signals. The quantity and efficiency of these elements directly impact how much logic the FPGA can implement and how complex the system can be.

Optimising resource usage means reducing how many of these elements a design uses. By simplifying logic paths, reusing hardware blocks, or packing multiple functions into fewer LUTs and flip-flops, space can be freed up on the FPGA. This can come at the cost of slower computational speeds or increased power dissipation depending on how these changes are implemented. This represents the crux of the project, and future reports will aim to balance these constraints against each other.

For this interim report, the goal is not to change the programs or optimise the device yet. This report only characterises the resource usage of the three programs, providing a basis for analysing resource usage in future designs.

3 Summary of Preliminary Design Work

To assess the resource usage of each of the three programs, the designs were synthesised using `yosys` and place-and-route was performed using `nextpnr`.

The resulting resource usage reports of these processes are displayed in the Appendix.

3.1 Hardwareblink

The first program evaluated was `hardwareblink`. This example does not use the RISC-V processor core and instead relies purely on combinational and sequential logic defined in `Verilog`.

The program implements a simple hardware-only design that toggles *LED D14* at 1 Hz using a counter. In order to toggle the LED at 1 Hz, `hardwareblink` makes use of the 48 MHz *HFOSC*, and divides it by 24 000 000 in a counter. This output toggles a register, the value of which is mapped to pin *D3* (driving *LED D14*), giving rise to a 1 Hz blinking light. The summary of the resulting logic utilisation is shown in Table 1 in the Appendix.

The design uses 33 flip-flops, 61 carry-chain elements, 65 combinational logic cells, the high frequency clock, and no block RAMs. This corresponds to 160 cells in total.

Subsequently, `nextpnr` was used to place and route these logic cells, with the results shown in Table 2 in the Appendix. This report shows that only 103 logic cells were used in this implementation, corresponding to approximately 2% of the available 5280 logic cells on the device.

No block RAMs, memories, or processor logic were involved in this design, making it a minimal baseline for comparison. The low utilisation confirms the simplicity of the circuit and provides a reference point for understanding how resource usage increases with more complex processor-driven programs.

3.2 Softwareblink

The second program evaluated was `softwareblink`. In contrast to `hardwareblink`, this design makes use of the RISC-V processor core to execute a compiled C program that toggles *LED D14* via software. The program

operates by writing to a memory-mapped I/O register at regular intervals, with timing determined by a simple delay loop. This introduces a significant increase in logic complexity, as instruction decoding, arithmetic, control logic, and memory access logic are all active.

Table 3 in the Appendix shows the breakdown of primitive cell usage as reported by `yosys` after synthesis.

The design uses 673 flip-flops (and three not shown in the report), 61 carry-chain elements, 1965 combinational logic cells, the high frequency clock, and 20 block RAMs. This corresponds to 2878 cells in total.

Evidently this software design uses far more logic elements than the previous hardware design, requiring a significant portion of the total RAM capacity of the device.

Table 4 in the Appendix shows the corresponding logic usage on the FPGA as reported by `nextpnr`. This design takes up 50% of the device's logic cell resources and two-thirds of its available block RAMs, representing a substantial increase in resource usage relative to `hardwareblink`.

This evaluation establishes the baseline resource cost of running programs on the processor. The increased logic and memory usage, relative to `hardwareblink`, reflect the added complexity of instruction decoding, pipeline control, and memory access. These results provide a benchmark for future optimisations aimed at reducing the resource footprint of the processor.

3.3 Bubblesort

The third program evaluated was `bubblesort`, which tests the processor under a significantly heavier computational load compared to `softwareblink`. The program implements a nested loop to sort a small array in memory, exercising the arithmetic, comparison, branching, and memory access capabilities of the RISC-V core. This makes it a more demanding benchmark for evaluating resource utilisation.

Table 5 in the Appendix shows the resulting breakdown of primitive cells as reported by `yosys`, and the corresponding place-and-route results reported by `nextpnr` are shown in Table 6 in the Appendix.

This design required a larger number of LUTs than `softwareblink`, increasing from 1 965 to 2 344, as the processor executes a more instruction-heavy and branching-intensive routine. However, the number of flip-flops, RAM blocks, and carry-chain elements remained constant. This suggests that the core hardware logic is unchanged, and the increase in combinational logic reflects the execution of a more complex program rather than architectural modifications.

4 Future Work

Looking at the `yosys` and `nextpnr` reports for the three baseline programs, it is clear that the primary constraints placed on resource usage occur due to software complexity, particularly in terms of block RAM and logic cell utilisation.

In order to optimise resource usage, the focus will be on reducing logic complexity introduced by processor control and data path operations. This may include combining pipeline registers where possible, removing support for unused instructions, and relocating the general-purpose register file into a single block RAM to reduce flip-flop usage. Where feasible, simpler control schemes that mirror the low-overhead structure of `hardwareblink` will be adopted. These changes should reduce the number of LUTs required, primarily by reducing the amount of combinational logic needed for control flow, arithmetic, and memory interfacing.

5 Conclusion

A baseline characteristic for the resource usage has been determined. Looking at the outputs of the synthesis and place-and-route reports, the primary reason for greater resource usage comes down to the software complexity applied to the FPGA.

The `hardwareblink` program had the lowest resource usage, relying on no block RAM and only taking up roughly 2% of the available logic cells. It achieved this by avoiding the use of the RISC-V processor entirely, instead implementing the desired functionality directly in Verilog as a simple counter and output toggle.

In contrast, both `softwareblink` and `bubblesort` introduced significantly more complexity due to the demands of instruction execution, memory access, and control logic. These additional demands used over 50% of the available logic cells and took up two thirds of the available block RAM.

In order to minimise the resource usage associated with FPGA implementations, future work will focus on optimising the processor microarchitecture. This includes combining pipeline registers where feasible, and cutting down on unnecessary logic functions. Each change will be weighted against the performance and power dissipation constraints.

A Resource Usage Data

Primitive	**Count**
SB_LUT4 logic cells	65
SB_DFF flip-flops	0
SB_DFFE flip-flops with clock enable	1
SB_DFFSR flip-flops with set/reset	32
SB_DFFSS flip-flops with set/set	0
SB_CARRY carry-chain elements	61
SB_RAM40_4K block RAMs	0
SB_HFOSC internal oscillator	1

Table 1: Primitive cell usage reported by `yosys` for the `hardwareblink` design

Resource type	Used	% of total
Logic cells (ICESTORM_LC)	103 / 5 280	1.9 %
Block RAMs (ICESTORM_RAM)	0 / 30	0 %
IO buffers (SB_IO)	1 / 96	1.0 %
Global buffers (SB_GB)	2 / 8	25 %
HF oscillators (SB_HFOSC)	1 / 1	100 %

Table 2: Place-and-route logic utilisation reported by `nextpnr` for `hardwareblink`

Primitive	**Count**
SB_LUT4 logic cells	1 965
SB_DFF flip-flops	415
SB_DFFE flip-flops with clock enable	154
SB_DFFSR flip-flops with set/reset	100
SB_DFFSS flip-flops with set/set	4
SB_CARRY carry-chain elements	216
SB_RAM40_4K block RAMs	20
SB_HFOSC internal oscillator	1

Table 3: Primitive cell usage reported by `yosys` for the `softwareblink` design

Resource type	Used	% of total
Logic cells (ICESTORM_LC)	2662 / 5280	50 %
Block RAMs (ICESTORM_RAM)	20 / 30	66 %
IO buffers (SB_IO)	8 / 96	8 %
Global buffers (SB_GB)	5 / 8	62 %
HF oscillators (ICESTORM_HFOSC)	1 / 1	100 %

Table 4: Place-and-route logic utilisation reported by `nextpnr` for `softwareblink`

Primitive	**Count**
SB_LUT4 logic cells	2 344
SB_DFF flip-flops	415
SB_DFFE flip-flops with clock enable	154
SB_DFFSR flip-flops with set/reset	100
SB_DFFSS flip-flops with set/set	4
SB_CARRY carry-chain elements	216
SB_RAM40_4K block RAMs	20
SB_HFOSC internal oscillator	1

Table 5: Primitive cell usage reported by `yosys` for the `bubblesort` design

Resource type	Used	% of total
Logic cells (`ICESTORM_LC`)	3073 / 5280	58 %
Block RAMs (`ICESTORM_RAM`)	20 / 30	66 %
IO buffers (`SB_IO`)	8 / 96	8 %
Global buffers (`SB_GB`)	5 / 8	62 %
HF oscillators (`ICESTORM_HFOSC`)	1 / 1	100 %

Table 6: Place-and-route logic utilisation reported by `nextpnr` for `bubblesort`