**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 1<br><br>**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

## 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`