# Highlight Report Document (Rev. 7.00)

PRCO304 - FPGA-based RISC microprocessor

**Ben Lancaster**
March 21, 2018

# Revision History

**Table 1:** Document revisions.

| Date | Highlight | Changes |
|---|---|---|
| 21/03/2018 | 7 | Highlight report 7. |
| 15/03/2018 | 6 | Highlight report 6. |
| 07/03/2018 | 5 | Highlight report 5.<br>Added Highlight Attachments section. |
| 28/02/2018 | 4 | Highlight report 4. |
| 20/02/2018 | 3 | Highlight report 3.<br>Added Risks & Challenges section.<br>Changed header (right) title. |
| 14/02/2018 | 2 | Highlight report 2. |
| 06/02/2018 | 1 | Highlight report 1. |

# Table of Contents

**1 Highlight Reports** **3**
1.1 Highlight Report 1 . . . . . 4
1.2 Highlight Report 2 . . . . . 5
1.3 Highlight Report 3 . . . . . 6
1.4 Highlight Report 4 . . . . . 7
1.5 Highlight Report 5 . . . . . 8
1.6 Highlight Report 6 . . . . . 9
1.7 Highlight Report 7 . . . . . 10

**2 Highlight Attachments** **11**

**3 Risks and Challenges** **13**
3.1 Project Management . . . . . 13
3.2 CPU Core . . . . . 13
3.3 Compiler . . . . . 13
3.4 Other . . . . . 13

# 1 Highlight Reports

## 1.1 Highlight Report 1

**PRCO304: Highlight Report 1**

**Name:** Ben Lancaster

**Date:** 06/02/2018

**Active project stage:** Stage 1.1: Research and Requirement Gathering

**Review of work undertaken:**
This week was assigned to work on stage 1.1: Research and requirement gathering.

**Research and requirement gathering:**
Research into existing soft-core processor designs has been started to identify their features, targets, and advantages and disadvantages. Key existing soft-core processors found are:
- Xilinx' MicroBlaze: a 32-bit Xilinx FPGA embeddable core capable of running operating systems, like Linux. Exposes a configurable GUI to customise the build of the processor to suit designers requirements (like number of GPIO, interrupts, timers, etc.).
- ARM Cortex-A9: a 32-bit Xilinx and Altera FPGA core. Features out-of-order execution, compatible with existing ARM Thumb2 C compilers, and multi-core processing.

I have used this research to aim my soft-core processor's requirements and architecture. To document and finalize my processors design and requirements, I have started a processor specification and reference document. This document outlines the processors features, architecture, compatibility, and instructions.

Additional progress:
- Version control set up for documentation, highlight reports, and code bases.

**Risks and Challenges:**
Urgent risks:
New risks:
Existing risks:
RC4: Schedule overrun. A gantt time chart has been created to better visualize task durations and requirements.

**Plan of work for the next week:**
Work will begin on Stage 1.2: Core high level design.

Finalised specifications and architecture of the soft-core processor will be put into a processor specification and reference document.
Architecture, control, pipelines, will be visualised in this document.

**Date(s) of supervisory meeting(s) since last Highlight:**
This is the 1st highlight report.
30/01/18 - An introductory meeting was held to discuss the project initiation document (PID) and gain feedback on the project.

**Notes from supervisory meeting(s) held since last Highlight:**
Ensure risks are carefully explored and project core deliverables are realistic and achievable.

## 1.2 Highlight Report 2

| **PRCO304: Highlight Report 2** |
|---|
| **Name:** Ben Lancaster |
| **Date:** 15/02/2018 |
| **Active project stage:** Stage 1.2: Core high level design |
| **Review of work undertaken:**<br>This week was assigned to work on stage 1.2: Core high level design. gathering.<br><br>**Core high level design:**<br>I have spent this week defining a processor specification and creating a processor specification/reference guide booklet (see attached). This booklet will contain both high-level and technical details regarding the design and implementation of the processor, including: register sets, control and pipelining strategies, the ISA and each instruction, and the compiler and how to use it.<br><br>This booklet will be developed over the life cycle of the project. Although the specification has been clearly defined, the booklet will be incrementally updated as processor features/requirements are added to the implementation (such as instructions, modules, and compiler features).<br><br>Currently the reference booklet contains: register set definitions, several primitive instructions, and a brief introduction to instruction cycle timing. |
| **Risks and Challenges:**<br>Urgent risks:<br>New risks:<br>Existing risks:<br>~~RC4: Schedule overrun. A gantt time chart has been created to better visualize task durations and requirements.~~<br>Resolved risks:<br>RC4: Schedule overrun. A gantt time chart has been created to better visualize task durations and requirements. (See attached time management chart indev.) |
| **Plan of work for the next week:**<br>Work will begin on Stage 2.0: Core dev. Register set implementation.<br><br>The register set module will be implemented in Verilog for the processor. Unit tests will be created to verify the timing/behaviour of the module.<br>The processor specification/reference booklet will be updated to describe how the register set has been implemented in the processor. |
| **Date(s) of supervisory meeting(s) since last Highlight:**<br>08/02/18 15:00 - 15:40 |
| **Notes from supervisory meeting(s) held since last Highlight:**<br>Discussion included comparing existing processor's (ARM, x86) features (privileged instructions, interrupts, IO, variable-length ISA) and designs (ISA and pipelining) to this processor. |

## 1.3 Highlight Report 3

**PRCO304: Highlight Report 3**

**Name:** Ben Lancaster

**Date:** 20/02/2018

**Active project stage:** Stage 2.0: Core Register-set Implementation.

**Review of work undertaken:**
This week was assigned to work on stage 2.0: Core Register-set Implementation.

**Core Register-set Implementation:**
Good progress has been made implementing the PRCO processor's register set in Verilog. The register set consists of 8 16-bit wide general purpose registers labelled rA through rH in duel-port read and single-port write.

Implementation progress is approximately 1 week ahead of schedule. Because of this, work has also been done on the decoder and ALU modules.

Consideration of the control/sequencing pipeline has been considered. The pipeline needs to work for time-varying functions (such as memory writes). The current plan is to give each module outputs to signal when it has finished so the following module can safely read in data and operate on it. A handshake between modules currently seems overkill due to the relatively simple structure but may be considered later in the project.

**Risks and Challenges:**
Urgent risks:
New risks:
RC5: Complex memory operations (PUSH, POP) may require multiple instructions. PUSH/POP might be split into: (1) Inc/dec stack pointer; (2) Read RAM[stack pointer]. The compiler will be able to resolve this issue.
Existing risks:
Resolved risks:

**Plan of work for the next week:**
Work will begin on Stage 2.1: Core dev. Decoder implementation.

Some progress has already made but the decoder is not finished.
The processor specification/reference booklet will continued to be updated with implementation specific details of the processor.

**Date(s) of supervisory meeting(s) since last Highlight:**
13/02/18 09:40

**Notes from supervisory meeting(s) held since last Highlight:**
This discussions was over email; it was decided that a physical meeting would not be beneficial as the current project stage was starting the *PRCO Processor Reference Guide* booklet. Progress on the booklet was shared and a brief overview of the Register-set and Decoder implementation.

## 1.4 Highlight Report 4

**PRCO304: Highlight Report 4**

**Name:** Ben Lancaster

**Date:** 28/02/2018

**Active project stage:**
Stage 2.1: Core: Register-set Implementation.
Stage 2.2: Core: ALU, RAM Implementation.

**Review of work undertaken:**
**Stage 2.1: Core: Decoder Implementation:**
Simple instructions, ADD, ADDI, MOV, MOVI, SUB, SUBI, LW, SW, instructions can now be decoded. The decoder has been integrated into the pipeline and it can choose and set up appropriate dependencies for the instruction.

**Stage 2.2: Core: ALU, RAM Implementation:**
ALU development has started. Some basic operations such as ADD, ADDI, SUB, SUBI, and pass-through ops such as MOV, MOVI, have been implemented. On-chip ram development will be starting this week.

**Core: Pipeline/control system**
A significant development breakthrough for the control/pipeline system has been achieved. I'm calling it a feed-forward pipeline as the flow of control only moves in the forward direction and when the previous module has completed.

**Compiler: Text parser development starting:**
Work into a simple text parser has begun including file opening, reading character by character, and a parser stack.

**Risks and Challenges:**
Urgent risks:
New risks:
Existing risks:
RC5: Complex memory operations (PUSH, POP) may require multiple instructions. PUSH/POP might be split into: (1) Inc/dec stack pointer; (2) Read RAM[stack pointer]. The compiler will be able to resolve this issue.
Resolved risks:

**Plan of work for the next week:**
Work will continue for 1 more week on stage 2.1 and 2.2 as per the time plan.
The processor specification/reference booklet will continued to be updated with implementation specific details of the processor.

**Date(s) of supervisory meeting(s) since last Highlight:**
21/02/18 13:00 - 13:4

**Notes from supervisory meeting(s) held since last Highlight:**
Discussion included improving time management gantt chart by showing task dependencies; and potential final demo ideas (store ASCII string on SDcard/external memory and have processor loop over and print each character out over RS232.

## 1.5 Highlight Report 5

| **PRCO304: Highlight Report 5** |
|---|
| **Name:** Ben Lancaster |
| **Date:** 07/03/2018 |
| **Active project stage:**<br>(ON-TIME) Stage 2.2: Core: ALU, RAM Implementation.<br>(EARLY) Stage 3.0: Compiler: Code-generation. |
| **Review of work undertaken:**<br>**(ON-TIME) Stage 2.2: Core: ALU, RAM Implementation:**<br>CMP and JMP instructions have been implemented. The CMP instruction is the only 3 register instruction (Type 3) and required a bit of reworking to implement. The CMP instruction subtracts Ra from Rb and sets appropriate status bits (SR_Z, SR_O, SR_E, SR_0) into the Rd register. The JMP instruction also required a bit of reworking as it affects the Program Counter. It is passed an 8-bit immediate containing jump conditions (JMP_EQ, JMP_GE, JMP_LT, etc.) and compares against the SR register specific in the CMP instruction.<br><br>**(EARLY) Stage 3.0: Compiler: Code-generation.**<br>Work has started ahead-of-schedule on code-generation for the compiler. I have begun implementing functions to encode instructions into the ISA's machine-code format. In addition, the compiler will also print out human-readable assembly in AT&T format.<br><br>**Real-hardware Implementation:**<br>I have also begun testing the implementation on the FPGA development board. Doing this early allows me to fix critical synthesis problems earlier, reducing risk for the project and demonstration. Figure 1 shows the FPGA core running on the FPGA development board. |
| **Risks and Challenges:**<br>Urgent risks:<br>New risks:<br>Existing risks:<br>RC5: Complex memory operations (PUSH, POP) may require multiple instructions. PUSH/POP might be split into: (1) Inc/dec stack pointer; (2) Read RAM[stack pointer]. The compiler will be able to resolve this issue.<br>Resolved risks: |
| **Plan of work for the next week:**<br>Work will begin into the integration of a UART (RS232) communication protocol, allowing us to better demonstrate functionality of the processor and connect to other peripherals.<br>Work will also begin on implementing an instruction single step cycle button, allowing better demonstration of the core. Currently the demonstration only lasts approximately 800ns.<br>The processor specification/reference booklet will continued to be updated with implementation specific details of the processor. |
| **Date(s) of supervisory meeting(s) since last Highlight:**<br>01/03/18 (bi-weekly highlight meeting) |
| **Notes from supervisory meeting(s) held since last Highlight:**<br>Biweekly meetings are held instead of weekly. |

## 1.6 Highlight Report 6

**PRCO304: Highlight Report 6**

**Name:** Ben Lancaster

**Date:** 15/03/2018

**Active project stage:**
(ON-TIME) Stage 2.3: Core: GPIO, Communication .
(EARLY) Stage 3.2: Compiler: Assembler.

**Review of work undertaken:**
Single-instruction stepping has been implementing allowing an external button to step and instruction (A key demo requirement!).

**(ON-TIME) Stage 2.3: Core: GPIO, Communication Implementation:**
A UART module library has been included in the core along with a FIFO buffer. The UART works well with single-instruction stepping, but free running the buffer immediately fills up and output is in random order.

**(EARLY) Stage 3.2: Compiler: Assembler.**
The assembler identifies instructions that require offsets and immediate to be calculated. The assembler can now modify instructions to fill in missing data.

**Risks and Challenges:**
Urgent risks:
New risks:
RC6: UART FIFO fills up too quickly, resulting in bad output.
Existing risks:
Resolved risks:
RC5: Complex memory operations (PUSH, POP) may require multiple instructions. Core will not support PUSH/POP as they are too complex. Compiler will output 2 instructions to emulate a PUSH/POP.

**Plan of work for the next week:**
Work will continue on parsing expressions in the compiler (if, for, while, etc.) and their codegen.

The processor specification/reference booklet will continued to be updated with implementation specific details of the processor.

The final report document content will be started (structure already laid out).

**Date(s) of supervisory meeting(s) since last Highlight:**
12/03/18 (bi-weekly highlight meeting)

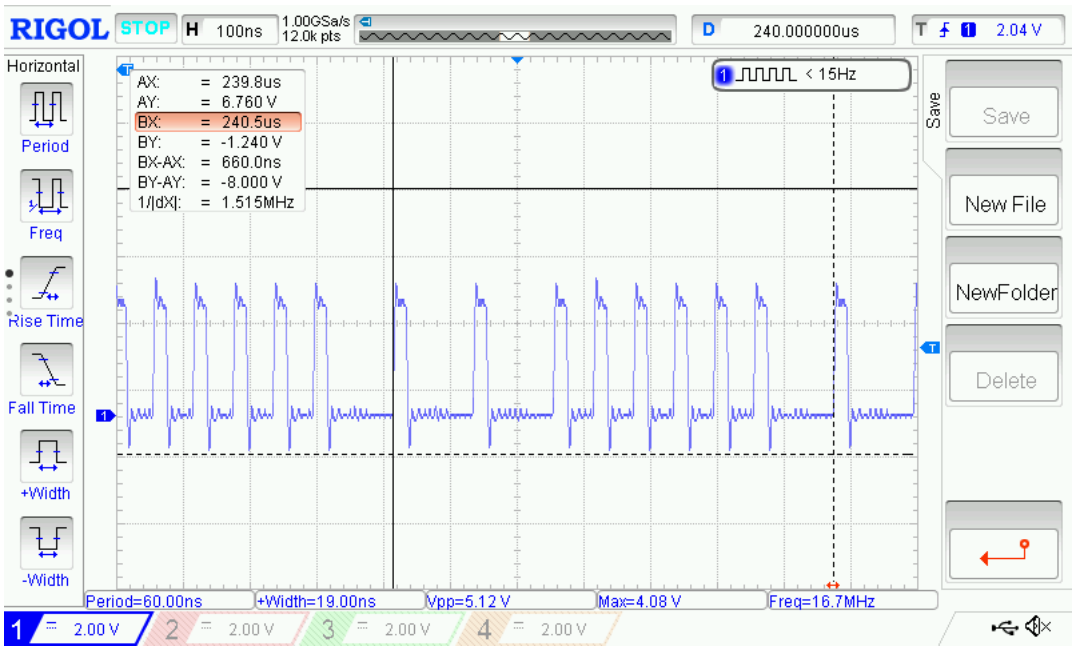
**Notes from supervisory meeting(s) held since last Highlight:**
RC6: Confirmation that PUSH/POP concepts will be split into 2 instructions due to limited complexity of the processor core.

## 1.7 Highlight Report 7

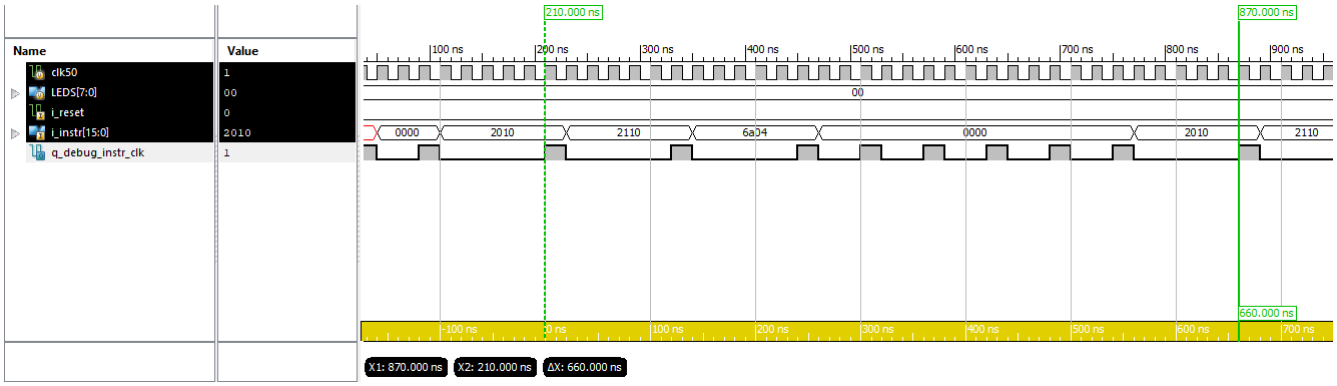
| **PRCO304: Highlight Report 7** |
|---|
| **Name:** Ben Lancaster |
| **Date:** 21/03/2018 |
| **Active project stage:**<br>(EXTENDED) Stage 2.3: Core: GPIO, Communication .<br>(ON-TIME) Stage 3.2: Compiler: Assembler.<br>(ON-TIME) Stage 3.3: Compiler: Verification. |
| **Review of work undertaken:**<br>HALT behaviour has been added.<br><br>**(ON-TIME) Stage 3.2: Compiler: Assembler.**<br>Compiler can now produce code generation for function x86 style stack frames, where the stack pointer and base pointer are pushed/popped to the stack when entering/exiting a function. This is the foundation for code generating passed and local parameters. An example is shown in section 2.<br><br>**(ON-TIME) Stage 3.3: Compiler: Verification.**<br>For the first time, the compiler output has been run on the processor. Two simple programs were run: one to test addition, and the other to test calling functions (without parameters). After fixing some bugs around the JMP instruction behaviour on the processor, both programs were able to run successfully. |
| **Risks and Challenges:**<br>Urgent risks:<br>New risks:<br>Existing risks:<br>RC6: UART FIFO fills up too quickly, resulting in bad output.<br>Resolved risks:<br>RC5: Complex memory operations (PUSH, POP) may require multiple instructions. Core will not support PUSH/POP as they are too complex. Compiler will output 2 instructions to emulate a PUSH/POP. |
| **Plan of work for the next week:**<br>Compiler language control statements such as IF and FOR need to be parsed and codegen'd. This is a requirement for the demo (iterating over contiguous memory and printing to UART?).<br><br>The processor specification/reference booklet will continued to be updated with implementation specific details of the processor.<br><br>The final report document will continued to be updated. |
| **Date(s) of supervisory meeting(s) since last Highlight:**<br>12/03/18 (bi-weekly highlight meeting) |
| **Notes from supervisory meeting(s) held since last Highlight:**<br>RC6: Confirmation that PUSH/POP concepts will be split into 2 instructions due to limited complexity of the processor core. |

# 2 Highlight Attachments

## Highlight 5




**(a)** Oscilloscope measurement of the *q_debug_instr_clk* signal running on the MiniSpartan6+ development board.




**(b)** Xilinx iSim simulation view of the *q_debug_instr_clk* signal.

**Figure 1:** Initial real-hardware implementation on the MiniSpartan6+ (XC6SLX9-3FTG256) development board showing timing of the *q_debug_instr_clk* signal. This signal is a 1 clock pulse indicating the start of an instruction cycle. In this example, instructions: MOVI $10, %Ra; MOVI $10, %Rb; and CMP %Rc, %Ra, %Rb followed by 6 NOP instructions, are used.

We can see that both implementations have a matching 660ns delay between instruction cycles for the same instructions, indicating that the real-hardware FPGA implementation is working correctly.

## Highlight 7

Compiler input file contents:

```
def foo() {
    10 + 1;
}

def main() {
    32;
    foo();
}
```

Compiler output machine code (pre-optimisation, post assembling):

```
0x00    ADDI    $-1,    Sp      4fff    Function/sf entry
0x01    SW      Bp,     +0(Sp)  16e0    (null)
0x02    MOV     Bp,     Sp      1ee0    main
0x03    MOVI    $20,    Ax      2020    NUMBER
0x04    MOVI    $9,     Cx      2209    Create return address
0x05    ADDI    $-1,    Sp      4fff    (null)
0x06    SW      Cx,     +0(Sp)  12e0    PUSH
0x07    MOVI    $d,     Cx      220d    call
0x08    JMP     Cx              6200    JMP
0x09    MOV     Sp,     Bp      1fc0    Function/sf exit
0x0A    LW      Bp,     +0(Sp)  0ee0    POP
0x0B    ADDI    $+1,    Sp      4f01    (null)
0x0C    HALT                    9000    MAIN HALT
-----------------------------------
0x0D    ADDI    $-1,    Sp      4fff    Function/sf entry
0x0E    SW      Bp,     +0(Sp)  16e0    (null)
0x0F    MOV     Bp,     Sp      1ee0    foo
0x10    MOVI    $a,     Ax      200a    NUMBER
0x11    ADDI    $-1,    Sp      4fff    (null)
0x12    SW      Ax,     +0(Sp)  10e0    PUSH
0x13    MOVI    $1,     Ax      2001    NUMBER
0x14    LW      Cx,     +0(Sp)  0ae0    POP
0x15    ADDI    $+1,    Sp      4f01    (null)
0x16    ADD     Ax,     Cx      4040    BIN ADD
0x17    MOV     Sp,     Bp      1fc0    Function/sf exit
0x18    LW      Bp,     +0(Sp)  0ee0    POP
0x19    ADDI    $+1,    Sp      4f01    (null)
0x1A    LW      Cx,     +0(Sp)  0ae0    POP
0x1B    ADDI    $+1,    Sp      4f01    (null)
0x1C    JMP     Cx              6200    FUNC RETURN to CALL
```

# 3 Risks and Challenges

Urgent risks
New risks
Existing risks
Resolved risks

## 3.1 Project Management

- RC4: Schedule overrun. A gantt time chart has been created to better visualize task durations, requirements, and dependencies.

## 3.2 CPU Core

- RC5: Complex memory operations (PUSH, POP) may require multiple instructions. PUSH/POP might be split into: (1) Inc/dec stack pointer; (2) Read RAM[stack pointer]. The compiler will be able to resolve this issue.
- RC6: UART FIFO fills up too quickly, resulting in bad output.

## 3.3 Compiler

There are currently no risks/challenges for the compiler.

## 3.4 Other

There are currently no additional risks/challenges.