Imperial College London



ELEC40006-Electronics Design Project 1 2019-2020 CPU Design – SEGFAULT Report for

Dr. Edward Stott and Mrs. Esther Perea

Alexander Pondaven, CID 01730117, Undergraduate Electrical and electronic engineering

Shuanghua Liu, CID 01729607, Undergraduate Electrical and electronic engineering

Yuliang Zhu, CID 01771679 Undergraduate Electrical and electronic engineering

Word count: 8305

Submitted June 2020

Contents

1 Abstract
2 Introduction
3 Project Planning and Management3
4 Design Criteria5
5 Outline of technical problem7
6 Design Process7
6.1 Overview of design7
6.2 Instruction Set Architecture (ISA) choice explanation12
6.3 ISA format14
6.4 Instruction Hardware Implementation18
6.5 Multiplication method research24
7 Benchmark tests
7.1 Benchmark: Fibonacci numbers using recursion32
7.1 Benchmark: Fibonacci numbers using recursion
 7.1 Benchmark: Fibonacci numbers using recursion
7.1 Benchmark: Fibonacci numbers using recursion327.2 Benchmark: Pseudo-random integers with LCG367.3 Benchmark: Traverse linked list to find an item398 Optimisation43
7.1 Benchmark: Fibonacci numbers using recursion327.2 Benchmark: Pseudo-random integers with LCG367.3 Benchmark: Traverse linked list to find an item398 Optimisation438.1 Basic CPU evaluation43
7.1 Benchmark: Fibonacci numbers using recursion327.2 Benchmark: Pseudo-random integers with LCG367.3 Benchmark: Traverse linked list to find an item398 Optimisation438.1 Basic CPU evaluation438.2 Choosing the multiplier45
7.1 Benchmark: Fibonacci numbers using recursion327.2 Benchmark: Pseudo-random integers with LCG367.3 Benchmark: Traverse linked list to find an item398 Optimisation438.1 Basic CPU evaluation438.2 Choosing the multiplier458.3 Pipelining46
7.1 Benchmark: Fibonacci numbers using recursion327.2 Benchmark: Pseudo-random integers with LCG367.3 Benchmark: Traverse linked list to find an item398 Optimisation438.1 Basic CPU evaluation438.2 Choosing the multiplier458.3 Pipelining469 Conclusion/Extension53
7.1 Benchmark: Fibonacci numbers using recursion327.2 Benchmark: Pseudo-random integers with LCG367.3 Benchmark: Traverse linked list to find an item398 Optimisation438.1 Basic CPU evaluation438.2 Choosing the multiplier458.3 Pipelining469 Conclusion/Extension5310 Link to Github54
7.1 Benchmark: Fibonacci numbers using recursion327.2 Benchmark: Pseudo-random integers with LCG367.3 Benchmark: Traverse linked list to find an item398 Optimisation438.1 Basic CPU evaluation438.2 Choosing the multiplier458.3 Pipelining469 Conclusion/Extension5310 Link to Github5411 References55

1 Abstract

This report presents in detail how a Central Processing Unit has been built to meet certain benchmarks and design goals. The CPU was based on an invented instruction set architecture and took advantage of the Harvard Architecture philosophy, while building off the work done with ARMish in term two. In this report, a clear explanation of the hardware implementation has been done including analysis of how individual instructions control hardware. The report also includes the optimisation process, where pipelining, latency/propagation delay and the trade-off between speed, power, and area have been discussed.

2 Introduction

The CPU is the core of a computer and plays a paramount role in electronic engineering. The primary objective was to build a CPU that could do the benchmarks (Calculating Fibonacci numbers, pseudo-random integers with a linear congruential generator and traverse a linked list), while remaining general (Turing complete). Hence research on different ISAs like AVR, MIPS, SPARC, and ARM, as well as the Harvard Architecture was conducted. Some instructions were specifically developed for the benchmarks like multiply, subroutines, and stacks. After deciding upon a 12 instruction ISA to meet the benchmark requirements, the separate hardware blocks were decided. The CPU's hardware took inspiration from the ARMish CPU built in term two, as it had a register file and ALU to carry out calculations. After implementing all instructions with hardware, the CPU was optimised to maximise power, speed, and area. This was done through pipelining, parallel computation, and analysis of different multipliers.

3 Project Planning and Management

Before starting the project, the Belbin Inventory had been completed to find out the different roles of team members. There was an implementer who is practical, reliable, efficient, hardworking, and methodical. It helped to have a shaper who is energetic, driven, and bold. The final person was a resource investigator who is outgoing and a great motivator. The task was split into three main tasks for each member, with all team members having a general understanding of the CPU. One person implemented the hardware and tested instructions. The second person had done research on the multiplier and hardware components. The last person helped write the test code for the benchmarks and implement the CPU. On 8th June, all the general instructions with associated hardware had been completed and tested. On 10th June, all three benchmarks had been completed and tested. On 12th June, the CPU had been optimised to achieve a better frequency. A regular meeting pattern had been used. All team members met every day since 24th May at 10:00 BST on Microsoft Teams to discuss progress and distribute tasks for the day. The TODO list website, Trello, and One Note were used to make share notes throughout the project. Some screen shots of the meeting plans and to do lists are in Appendix 3.

Snippet of team notes:

Date	Discussion	Alex	Peter	Jason
26/6	 Discussed Jason's assembly code for fib benchmark Went over stacks Discussed number of registers and instruction format Decided it is best to decide this once all benchmarks have been turned into assembly and we know what instructions will be needed 	Look at next benchmarks and see what instructions we may need and convert to assembly	Research multiplication methods	Make assembly more efficient and think of register to use
27/5	 Went over assembly code for LCG and linked list benchmarks Discussed multiplication methods Made a list of instructions using benchmarks 	Think of implementation of instructions (DECA oral as well)	Continue multiplication method research	Reduce number of instructions used in fib

4 Design Criteria

The CPU should be able to run the following three benchmarks.

Calculate Fibonacci numbers using recursion

```
int fib(const int n){
    int y;
    if (n <= 1) y = 1;
    else {
        y = fib(n-1)
        y = y + fib(n-2);
    return y;
}</pre>
```

Calculate pseudo-random integers with a linear congruential generator (LCG)

```
int lcong(
    const unsigned int a,
    const unsigned int b,
    const int n,
    const unsigned int s)
{
    unsigned int y = s;
    unsigned int sum = 0;
    for (int i = n ; i > 0; i--){
        y = y*a + b // calculate the new pseudo-random number
        sum = sum + y // add it to the total
    }
    return sum;
}
```

Traverse a linked list to find an item

```
typedef struct item{
    int value;
    struct item *next;
} item_t;
item_t* find(const int x, item_t* head){
    while (head->value != x){
        head = head->next;
        if (head == NULL) break;
        return head;
}
```

Software requirements specification:

Functional requirements:

- Benchmark tests:
 - Fibonacci test and recursion
 - Stack
 - Stack pointer
 - o LCG
 - Multiplication
 - o Traverse linked list and find item
 - Indirect addressing
- Correctness
 - Use benchmark algorithms to check using trial data
 - Compare with hand-calculated results
- Speed minimise geometric mean time (T1T2T3)^(1/3)
 - Found by counting number of CPU cycles required for each benchmark and how this changed with size of problem (e.g. size of list)
 - Find max clock speed of design and minimum execution time
- Power consumed minimise number of logic gates
 - Number of logic gates and clock speed

Non-functional requirements:

- Greatest number of applications for least number of transistors
- 16-bit instruction word
- At least 2k words of instructions and 2k words of data
- Built and simulated using Quartus

5 Outline of technical problem

Technical Problem: Build a general CPU that can complete the given benchmarks.

The CPU was not a task to create a hyper specialised instruction set that only implements the benchmarks in the most efficient way and does nothing else. It also required the CPU to be Turing complete, which involve implementing indirect or register addressing of memory, computed jump (jump with register value), and loading current value of PC into a register or memory. Each benchmark also required specific instructions and hardware to be implemented, including subroutines, multiplication, and the stack.

6 Design Process

6.1 Overview of design

Memory

The memory blocks took inspiration from the Harvard architecture philosophy1. This involved separated the memory into an instruction ROM, where the main program was stored, and a data RAM, where all the data was stored. The different memory types could then be addressed differently. This was useful for implementing indirect addressing and a pointer to the stack (a section in the data RAM), as the ALU can output the correct data address for each cycle depending on the instruction. The instructions and data could also be accessed simultaneously, which was very useful for pipelining, as the next instruction can be fetched at the same time as reading or writing from the data memory.

The instruction memory could be either a ROM or RAM as the CPU never needs to write to this memory block. The two types of memory both simulate pretty much identically in Quartus. The ROM was used as it looked cleaner without all the extra write outputs for RAM (see Appendix 9 for more detail on decision).

The size of both memory blocks were 4096 16-bit words. As the opcode was 4 bits, one instruction (LDI) could use the rest of the bits to load a 12-bit constant that could be used as an address for all 4096 locations in the memory blocks. This was useful for testing.

¹ Scott Thornton, "What's the difference between Von-Neumann and Harvard architectures" March 8th 2018 [online] MICROCONTROLLERTIPS available at:

<<u>https://www.microcontrollertips.com/difference-between-von-neumann-and-harvard-architectures/</u>>

Register file



The register file included 8 registers. This required a 3-bit port addresses to select which register to read from or write to. Using 16 registers would have required 4 bits, which would take much more space in the instruction words, and many registers would not be used for the benchmarks given. This register file took inspiration from the ARMish CPU register file built in the second term, although register files are common for most CPUs.

ALU with register file (file called ALU1)



Like ARMish, the ALU was able to read from two registers in the register file, addressed using bits in the instruction word. The ALU implemented the decoder for all the signals and could do calculations on the two registers that were read. It handled all the logic and was implemented in Verilog (full ALU in Appendix 1).

State machine (unpipelined)

The synchronous state machine controlled what state the CPU was in.



The unpipelined state machine just switched between the execute (EXEC1) cycle and the FETCH cycle as a new instruction is fetched every second cycle. This made it easier to pipeline the CPU as all logic was already carried out during EXEC1. The states were numbered as above, so the next state was just the inverse of the current state.



Decoder:

Originally, the decoder for control signals was separate from the ALU, although most logic signals were in the ALU, so many output and inputs needed to be created to leave different BSF files, which increased the time it took to test. This was simplified by moving the state machine into the regfile_ALU BDF and decoder logic was put into the ALU.

Original decoder:

```
module decode
input
                                       [3:0] op,
                        input FETCH,
input EXEC1,
output pc_sload,
                       output pc_cnt_en,
output ram_wren,
output pc_in_sel,
output ldi_sel,
                        output din_sel
               ):
               wire ldi, ldr
assign ldi =
assign ldr =
assign str =
assign str =
                                                                                                          sub, m
& !op
& !op
& op
& op
                                                                                                                       mul,
p [ 1
p [ 1
                                                                                              add,
                                                                                                                                     xsr
] &
] &
                                                                                                                                                 xs1,
!op [
op [ 0
op [ 0
                                                                                                                                                              b],
0];
0];
0];
                                                                                 jmp
!op
                                                       str
                                                                    mov
                                                                                                                                                                          stp;
                                                                           /, jm;
& !o;
& !o;
& !o;
& !o;
& op
& op
& op
& op
                                                  lop
                                                                                                                                                !op
                                                                                                2
                                                                      3
                                                                                                                                                            0
                                                 !op
                                                                                 !op
                                                                                                                                               op
                                                                                                 22
                                                                                 ! op
                                                                                                                                       &
                                                  !op
                                                                 333
                                                                                                      Ì
                                                                                                                                             !op
                                                                                                                        1
                                                                                                                                  Ę
                                                                                                                                                       io
E
                                                                                                                                                             (<sup>)</sup>;;
                                                  !op
                                                                                 !op
                                                                                                                                       &
&
                                                                                                                                             ор
                                                                                                                                                     Ε
                                                                                                        &
                                                                                                                                                            0
                                                                                                   assign
assign
                                  jmp
add
                                            =
                                                 !op
!op
                                                                                                                             1
                                                                                              2
2
2
2
                                                                                                              !op
                                                                                                                                             !op
                                 add =
sub =
mul =
                                                                                                       &
&
&
&
                                                                  3
                                                                                                              !op
                                                                                                                             1
                                                                                                                                       &
                                                                                                                                             ор
                                                            3
3
; ]
[
[
3
                               sub = !op 
mul = !op [
bl = op [ 3
ldmfd = op
stmfd = op
stp = op [
                assiğn
assign
                                                                                                             op
op
                                                                                                                               ]
                                                                                                                                     &
&
                                                                                                                                           !op
                                                                                                                                                          0
                                                                                                                      Ľ
                                                                                                                          \frac{1}{1}
                                                                                         t
                                                                                                                                                           );
];
                                                                                                                                          op
                                                                                            2
[
[
2
                                                                                                                                                  Ε
                                                                   3 ] & op
] & !op [
3 ] & !o
3 ] & !op
] & !op
                                                                                                                 p [ 1 ] &
!op [ 1 ]
op [ 1 ] &
p [ 1 ] & (
                                                                                   o [ 2
!op
!op
!op
pp [
                                                                                                  ] & !op
2 ] & !op
2 ] & !
2 ] & !
] & op
                assign
assign
assign
                                                                                                                                           !op
&
                                                                                                                                                          õ
                                                                                                                                                op
!op
                                                                                                                                                                    ];
                                                                                                                                          &
                                                                                                                                                               Ō
                assign stp = op
                                                                                                                                          op
                                                                                                                                                  Γ
                                                                                                                                                       0
                                                                                                                                                            1:
               // Need to think of all decode signals and then implement logic
assign pc_sload = exec1 & jmp;
assign pc_cnt_en = exec1 & ~(jmp | stp);
assign ram_wren = exec1 & (str | stmfd);
assign ldi_sel = ~ldi;
                endmodule
```

CPU block



This block made the connections between the regfile_alu block, the program counter and the memory. The program counter (PC) addressed the instruction memory, while the address for the data memory came from the ALU. The MUX at the data input of the PC switched between loading the PC with the value of the second register that was read for jump instructions and the least significant 12 bits of the instruction for branch instructions (discussed in the next section).

6.2 Instruction Set Architecture (ISA) choice explanation

The instructions were chosen to meet the requirements of the benchmarks and to create a general CPU. Given the word length of 16 bits, there was a limitation to how many bit fields could fit into the instruction. The register file contained 8 registers, which required a register address of 3 bits to select one of them. Therefore, two-operand instructions were used as it only took 6 bits of the instruction to select the two registers to read/write to. Three-operand instructions would be too large as they would require 9 bits and only 7 bits would be left for the opcode and other bit fields.

To decide upon the instructions required, the benchmark codes were first converted into a first draft of assembly code, so that the required instructions could be found.

LDI	Load a 12-bit constant into R0
LDR	Load value from memory into a register
STR	Store value from register into memory
MOV	Move shifted value of one register to another
JMP	Jump to an address depending on a condition
ADD	Add two register values
SUB	Subtract two register values
MUL	16-bit multiplication of two register values stored in separate output registers
BL	Branch and link
LDMFD	POP - Load value from stack and increase stack pointer
STMFD	PUSH - Store value into stack and decrease stack pointer
STP	Stops the program

ISA:

LDI: It was useful to load a 12-bit address into a register to be able to address all 4096 locations in memory. Due to the 16-bit word limit, LDI only worked for one instruction

LDR, STR: These instructions took inspiration from ARM, and were both used to implement register addressing. This is needed for the "find in list" benchmark.

MOV: Just like in ARM, the MOV instruction also carried out different shifts on the register value. While this was not specifically required for the benchmarks to work, it is a very useful function for general CPUs to have.

JMP: Jump was required for all benchmarks to implement loops in assembly code. The benchmarks also required conditional jumps, which were implemented using a condition field in the instruction that determines what condition needs to be met for the jump to work.

ADD, SUB: Based off ARMish.

MUL: Multiplication was required for the pseudo random number generator. This was implemented with a separate multiplier block, which both allowed the team to work in parallel, and for the ALU to carry out instructions at the same time as the multiplication is occurring.

BL: Branch and link² implemented subroutines, and were based off ARM. This involved having a link register (register 6) that stored the next address and then jumped to the address of the subroutine in the instruction word. When this subroutine (like a function in a separate part of the memory) was finished, it jumped to the value in the link register to return to where it left off in the main program. This instruction was useful when implementing the recursive Fibonacci benchmark test as it needed to call the Fibonacci function multiple times and needed to return to where it left off.

LDMFD/STMFD (POP/PUSH): These instructions implement the stack that is required for the Fibonacci benchmark. The stack was a section of the data memory addressed by a stack pointer. The stack pointer (register 7) decreased for POPs and increased for PUSHes. This took inspiration from the AVR ISA³. Each of these instructions needed to be called three times to pop/push the value of the link register, and the two local variables (M for multiple). The FD stands for full descending as the stack starts at the end of the data memory. This prevented the need to initialise the stack pointer at a particular address and run the risk of it exceeding 0xFFFF. Instead, the stack starts at 0xFFFF and descends when adding new items to the stack.

² Clarke, T. Class Lecture, Topic "LEC9.pdf", Department Of Electrical & Electronic Engineering, Imperial College London, UK,2020. Available at:

<https://intranet.ee.ic.ac.uk/t.clarke/arch/html16/lect16/lec9.pdf>

³ http://ww1.microchip.com/downloads/en/DeviceDoc/doc2503.pdf

6.3 ISA format

The decided ISA format took some inspiration from the ARM Thumb ISA4, due to the 16-bit word length. It helped make decisions on how many bits are needed for each field and how to reduce the instruction set.

ISA Table:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0		12Bit constant							LDI				
1	0	0	0	1	S		O	FS	ΕT			Rd			Rs		LDR
2	0	0	1	0	S		0	FS	ΕT			Rd			Rs		STR
3	0	0	1	1	CI	Ν	cwen		SH	IFT		Rd			Rs		MOV
4	0	1	0	0	Со	nd	Cor	npa	arat	tor		Rd			Rs		JMP
5	0	1	0	1	CI	Ν	cwen					Rd			Rs		ADD
6	0	1	1	0	CI	Ν	cwen					Rd			Rs		SUB
7	0	1	1	1								Rd			Rs		MUL
8	1	0	0	0		Address of the subroutine							BL				
9	1	0	0	1								Rd		1	1	1	LDMFD
10	1	0	1	0							1	1	1		Rd		STMFD
11	1	0	1	1													STP

Note: Rd= destination register, Rs = source register

Descriptions:

0 0 0 0 12Bit constant

LDI would load a constant to Register 0. It would read the operand (12 bits) to store the integer to the register. Being able to load a 12 bits constant would allow the CPU to load any address number to the register, and thus it could jump to any address in the memory. The opcode of LDI is 0000, so this helped address directly to R0 by letting port address in the register file read from the opcode.

⁴ ARM QRC 0006E, Thumb 16-bit Instruction Set Qucik Reference Card [online] available at: <<u>http://infocenter.arm.com/help/topic/com.arm.doc.qrc0006e/QRC0006_UAL16.pdf</u>>

0	0	0	1	S	OFFSET	Rd	Rs	LDR
---	---	---	---	---	--------	----	----	-----

LDR would load a value stored in the memory to one of the registers. Rs bits [2:0] was the address of the register which stored the address of the value to be loaded to another register. Rd bits [5:3] was the address of the register where the value should be moved to. OFFSET bits [10:6] would provide a positive or negative offset to the value in the Rs depending on the S bit. Sign bit (S) [11] would define the direction of the offset: 0 would be positive offset, 1 would be negative offset.

In order to do that, the value stored in the memory first need to be loaded into R0 using LDI.

The offset function helped the CPU implement register addressing for the "Finding in linked list" benchmark, since it could load the next location of the value where the address of the next item is stored.

The problem is that the LDR would require two cycles, one for the retrieving the data from RAM, one for writing back the value to the register. This was solved using the same state machine by allowing instructions to be completed in parallel (See Hardware descriptions in next section).

0	0	1	0	S	OFFSET	Rd	Rs	STR
---	---	---	---	---	--------	----	----	-----

STR was implemented in the same way as LDR except instead of loading a register with a memory address, STR stores the register value into the memory address given by mem[Rs ± Offset]. This was not required for the benchmark but helped generalise the CPU.

0 0 1 1 CIN cwe	SHIFT Rd	Rs Mov
-----------------	----------	--------

MOV would move the value from Rs to Rd while also doing optionally shifting the value. Rs bits [2:0] was the register which stored the value to be moved. Rd bits[5:3] was the register where the value should be moved to.

Shift type:

00: no shift

01: shift left

10: shift right

11: move multiply registers

CIN field:

00: cin=0

01: cin=1

10: cin = carrystatus (previous carry)

11: cin = CMSB (carry most significant bit of Rs)

The Move multiply registers option was added later with multiplication as the product is stored in two separate registers from the register file, so these instructions were necessary to move them to the register file. Only one MOV instruction was required after multiplying for the LCG as only the least significant 16 bits are required. Although, general multiplication with the full 32-bit product is possible.

0	1	0 0	Cond	Comparator	Rd	Rs	JMP
---	---	-----	------	------------	----	----	-----

JMP would change the address in the program counter under different condition. Rs bits [2:0] was the register which stored the address the program counter should jump to. Rd bits [5:3] was the register which stored the value to be compared with the comparator. Comparator bits [9:6] was the value to be compared with the value in Rd.

COND field [11:10]

00: JMP (always)

01: JEQ (jump if equal) Rd==comparator

- 10: JMI (jump if less than) Rd < comparator
- 11: JMB (jump if bigger than) Rd > comparator

0	1	0	1	CIN	cwen	Rd	Rs	ADD
0	1	1	0	CIN	cwen	Rd	Rs	SUB

ADD/SUB would add/subtract the value in Rd and Rs and store the result back to Rd. Rs bits [2:0] was the register which stored one of the number for addition. Rd bits [5:3] was the register which stored one of the number for addition and the register the result would be stored in. Bits [8:6] are reserved for future features. CWEN bit [9] would enable writing the result to the carry register. CIN is added to the result. Note that normal subtraction required CIN=1 as the conversion to 2's complement of Rs required an inversion and addition of 1.



MUL carried out 16-bit multiplication between Rd and Rs and stored the product in 2 output registers after the multiplier block.

|--|

LDMFD/POP would load the value stored in the stack area back to the register. Bits [2:0] was the location of the stack register (R7). Rd bits [5:3] was the register where the value should be restored to. Bits [11:6] were not needed.

The process of this instruction is

Rd = Mem[Stack Pointer]

Stack Pointer = Stack Pointer + 1



STMFD would store the value in the register to the stack area. Rd bits [2:0] was the register stored the value to be preserved. Bits [5:3] was the location of the stack register (R7). Bits [11:6] were not used. The location of 111 was flipped as it made the logic easier.

The process of this instruction is:

```
Stack Pointer = Stack Pointer - 1
```

Mem[Stack Pointer] = Rd

The reason why stack pointer would decrement by one first is that the stack register would initialise to be 0, and minus one would let it be 0xFFFF which pointed to the end of the memory.



STP function would stop the whole program (prevent the program counter from counting).

6.4 Instruction Hardware Implementation

LDI, ADD, SUB, and STP were implemented like ARMish

Input to register file



The series of MUXes selected the input of port1addr[2:0] (to address the register to write to) depending on the instruction. The possible inputs included instr[5:3] to address Rd, instr[2:0] for selecting register 7 for the pop instruction, 3'b000 for LDI, 3'b110 (register 6) for writing into the link register in BL, and write_next_status[2:0], which is used for LDR (described

below). The MUXes at the input of din (data in of register file) also switched between either data from the RAM (din[15:0]), the instruction word (instr[15:0]), or the output of the ALU (aluout[15:0]). See appendix 1 for ALU logic for control signals.

Output of ALU



The carry flip-flop bits carryout, carryen, and carrystatus were implemented using the ARMish design. The other register is used for LDR instructions as explained below.

LDR hardware implementation

LDR required two cycles: one for reading a value from the RAM, and one for writing data out of the RAM into the register file. This meant that data in (din) of the register file is only updated one cycle after the LDR instruction, so port1addr needs to be correct one cycle afterwards, as well as write enable (wen). A register called write_next was added at the ouput of the ALU that stores "write_next_flag" (telling the next instruction that the data is now at dout during the next cycle and can enable writing) and the address of Rd given by bits [5:3] of the LDR instruction. This register was enabled every execute cycle.

// Status FF bits:

// write_next tells next instruction that the data that is now at dout can be written into the Rs of the previous instruction (for load instructions)
// if write_next_flag is already 1, set to 0 (if it is not another ldr), otherwise set to 1 if it is an ldr or ldmfd instruction
assign write_next_flag = (write_next_status[3] & (~ldr | ldmfd)) ? 0 : (ldr | ldmfd);
// write_next_en enables writing to write_next register - needs to update for every instruction during exec1 so that it returns to 0
assign write_next_en = exec1;
// Output to write_next_register
assign write_next_out = {write_next_flag, instr[5:3]};
// write_next_sts proper form counting up so instruction can finish before next instruction
assign write_next_stp = write_next_status[3] & ~(ldr | str | jmp | ldmfd) | write_next_status[3] & ldmfd;
// carryoun enables writing to carry register - writes when cwen is enabled for add, sub, and mov
assign carryen = execl & (wret & (add | sub | mov);
// carryout equal to alucout if not a shift - note the special case of rsdata[0] for LSR or ASR (MOV instruction)
assign carryout = (mov & ((-field[1] & field[0]) | (field[1] & ~field[0]))) ? rsdata[0] : alucout;

This implementation allowed other instructions to be carried out in parallel with this second cycle for LDR, given that the following instruction does not write to the registers or read from the register being written to by LDR (as it has not updated yet). To prevent the CPU from bugging if the next instruction was a write instruction, the write_next_stp bit stops PC from counting during this second LDR instruction to allow it to finish before the next instruction executes. This acted like a delay to the system and allows the assembly to have any instruction after LDR as the ALU decides if the PC waits for LDR to finish or if the next instruction can be done in parallel. Note that LDR instructions could be executed after each other and be done in parallel as the first cycle of LDR just involved the data RAM. An XOR gate between the write enable out (wenout) of the ALU and the write_next_flag from the write_next register determined if the register is written or not (as these two bits should not be on at the same time).

LDR instructions after each other

Data memory: 0x0004 0x0002 0x00AB

Instruction mem:

LDI 0x1	0001
LDR 0 0 R1 R0	1008
LDR 0 0 R2 R0	1010
LDR 0 0 R3 R2	101A
STP	B000

Simulation:

				۲			
in —		CLOCK	В0		лл		лл
out		FETCH	B 1		11		ЛL
out		EXEC1	B 0				1
2	>	daddr	H 000			000	
	>	dout	H 0000	(i)X 0004 X 0002 X 00AB X		0004	
9	5	instr ad	H 000		(004	
<u>е</u>	,	inetr	H 0000			B000	+++++
•	ŕ	mou	11 0000			5000	
•	>	port1addr	B 000			000	
out		ldi_sel	B 1				
out		wenout	B 0				
out		wen	B 0				
out		carryen	B 0				
out		carrysta	B 0				
out		write n	BO				
-	,	write n	B 0000			0000	
aut		a1	H 0000		+++++	0001	+++++
-	Ľ	41	11 0000			0001	
•	>	q2	H 0000			0001	
S	>	R0out	H 0000	(0000)X 0001			
eut	>	R1out	H 0000	0000 X	0002		
	>	R2out	H 0000		000	2	
-	>	R3out	H 0000	0000		00AB	
Щ.	>	R4out	H 0000	0000			
	Ľ.						

In the simulation multiple LDR instructions could be executed consecutively after each other as the output of a register updated every EXEC1 clock edge.

Indirect addressing (LDR and STR)

// ALU output Calculations
assign alucout = alusum[16]; // carry bit
assign daddr = ldmfd? (alusum[11:0]-1) : alusum[11:0]; // offset address used for addressing data memory or stores R7 value for stack manipulation
assign aluout = alusum[15:0]; // 16 bit sum assign dmemin = stmfd ? rsdata : rddata; // input to data memory is q2 for stmfd, q1 for str

// determine alusum - need to change opcodes
Jalways @(*) begin
acse (op)
4'b0001, 4'b0010 : alusum = sign ? (rsdata - {11'b0,offset}) : (rsdata + {11'b0,offset}); // LDR and STR: calculate daddr

Indirect addressing was achieved by calculating the offset address into alusum and storing this as the data address (daddr) as seen above. The offset address was calculated using the parameters specified in the instruction word. This was used in both LDR and STR to specify the address for the RAM.

MOV

The move instruction was implemented similarly to ARMish together with the shift functionality inside the ALU using alusum.

4'b0011 : begin
case (instr[7:6])
2'b00 : alusum = {1'b0,rsdata} + cin; // MOV
2'b01 : alusum = {rsdata.cin}; // LSL
2'b10 : alusum = {rsdata[0], cin, rsdata[15:1]};// Right shift - cin determines LSR or ASR
2'b11 : alusum = instr[0] ? ({1'b0,MS_prod}+{1'b0,rddata}+cin) : ({1'b0,LS_prod}+{1'b0,rddata}+cin);
default : alusum = {1'b0,rsdata} + cin; // MOV
endcase;
end

The option of no shift also could add cin, so this was a quick way of moving a register and adding one simultaneously. When instr[7:6]=11, the mov instruction output either the least significant 16 bits (LS_prod) or the most significant 16 bits (MS_prod) of the multiplication result depending on whether Rs is 0 (moves LS_prod) or 1 (moves MS_prod). LCG only used the Rs=0 option, as only LS_prod is required. Alusum is directed back into the data in of the register file to be written into the register specified by port1addr.

JMP:

On the other hand, jump was implemented with a wire, jmp_cond, that determined if the PC should load the address of Rs depending on the conditions given in the opcode and by the comparisons made (eq and mi). Jmp_cond enables PC loading and stops it from counting.

```
// conditional operators - compares Rd and comparator (in cond) - used for if jump should occur
wire eq = (rddata == cond); // Rd == comparator
wire mi = (rddata < cond); // Rd < comparator
// change jmp cond to determine if jmp will occur depending on comparison and jump condition in instruction word
wire jmp_cond = jmp & ((-field[1] & -field[0]) | (-field[1] & field[0] & eq) | (field[1] & -field[0] & mi) | (field[1] & field[0] & ~eq & ~mi));
// PC and RAM control signals
assign pc_sload = exec1 & (jmp_cond | bl);
assign pc_cnt_en = exec1 & (jmp_cond | stp | bl | write_next_stp);
assign ram_wren = exec1 & (str | stmfd);
```

MUL:

Refer to multiplication method research for information about the Booth Radix 4 multiplier that was used.

Hardware in regfile_alu bdf:



The left-hand registers were used to store the values read from the two registers in the register file specified in the instruction word. This was necessary as these multiplicands needed to be constant throughout the entire 9 cycles that it takes this multiplier to carry out the multiplication. When the MUL instruction started, the mul_start control signal was asserted, getting stored in a DFF, so that in the next cycle, the multiplication can start in the multwithRadix4 block. When it was done, the mul_finish signal was activated, which enabled writing to the output registers, where the 32-bit product was stored. As discussed previously, the MOV instruction had an option to store this product into the register file. This could not be done with the MUL instruction as there was no space in the instruction word to include output registers.

2'b11 : alusum = instr[0] ? ({1'b0,MS_prod}+{1'b0,rddata}+cin) : ({1'b0,LS_prod}+{1'b0,rddata}+cin);

In the alusum calculation, add and multiply was incorporated into this design, so that the multiply product gets added onto what is already in the destination register. This allowed it to first move "b" into the register in the LCG benchmark and then move the product into the same register, where it automatically adds the "b". Therefore, an additional ADD instruction was not needed.

BL: Branch and Link

The link register was stored with "PC+1" and the port1addr selects 6 the MUX described at the beginning of the section.

4'b1000 : alusum = {5'b00000,pc_out} + 1; // BL

When the MUX at the data in of PC was asserted, it selects the least significant 12 bits of the instruction to be loaded into the PC. This carried out the jump from the instruction word rather than from a register for JMP instructions.

LDMFD(POP)/STMFD(PUSH)

LDMFD (pop) writes the new value into the stack pointer (subtracted by 1) while using the old value as the data address. STMFD (push) writes the new value into the stack pointer (added by 1) while using the new value as the data address. This logic was reflected in the daddr wire seen below. The stack pointer, R7 was selected at port1addr using a MUX (either from Rs or Rd – refer to ISA table).

4'b1001 : alusum = {1'b0,rddata} + 1; // POP (LDMFD) 4'b1010 : alusum = {1'b0,rddata} - 1; // PUSH (STMFD)

[assign daddr = ldmfd ? (alusum[11:0]-1) : alusum[11:0]; // offset addressed used for addressing data memory
All tests for instructions found in Appendix 13

6.5 Multiplication method research

Multipliers are important in CPU design, as they have long latency and consume relatively considerable power. A system's performance largely depends on multipliers because the multipliers are one of the slowest element within the system. In this section, a few ways of implementing the multipliers have been introduced and the strength and drawback of each implementation is discussed. In real life, the most important thing about multipliers is power, the power consumption of multipliers need to be controlled as low as possible, and proper cooling system need to be designed to remove the heat generate from the multiplier. However, in this project, the improvement in fundamental arithmetic functionality and speed of the multiplier can outweigh the increased power usage. Below is a table that shows multiplication in math perspective.

Multiplication table

		A ₃	A2	А,	A
	Х	Bz	32	B,	₿°
		AzBo	A2Bo	A, BO	AoBo
	+ A381	AzBI	A, B,	A.B.	
+ AzB	Az Bz	A, B2	A0B2	,	
+ A3B3 A2B3	A, B3	AoB3			

Multiplying A(N-bits) and B(M-bits) resulted in M*N bits

Multiplier Type I : shift adding multiplier(Combinational Multiplier)

A shift and add multiplier architecture is demonstrated in Figure 3. The inputs include a start signal, clock, and multiplicand. It used the shift and add method to output the result with a stop signal.





These multipliers are simple and take up little area. Higher radix multipliers are quicker, although there is a higher power usage due to the larger register use and more complicated

⁵ Deepak Bordiya, and Lalit Bandil, "*Comparative Analysis Of Multipliers*" International Journal of Engineering Research & Technology. Volume 2 Issue 9, September - 2013, pp. 1437– 1441. Accessed on: 8 June 2020. [Online]. Available at: <<u>https://www.ijert.org/research/comparative-analysis-of-</u> <u>multipliers-serial-and-parallel-with-radix-based-on-booth-algoritham-IJERTV2IS90625.pdf</u>>

logic. Implementing the example above using Verilog, a time analysis could be run to find out the propagation delay. If the multiplier with full-adder and logic gate was implemented, it would contain many adders and logic gate. The logic is the same with the multiplication table, each individual bit of the product came from multiple addition. The exact number of additions depends on the bit number.



Figuer 4 http://www.ellab.physics.upatras.gr/~bakalis/Eudoxus/CSAM.html

This image illustrates a hardware implementation for an 8 * 8-bit combinational multiplier, if N is the bits of the multiplicand, the delay should be 2N times the delay of the full adder.

The implementation could be done in Verilog, although this would be the function alreadyimplementedinFigure5.

wire [9:0] a,b; wire [19:0] result = a*b; // unsigned multiplication! If you want Verilog to treat your operands as signed two's complement numbers, add the keyword signed to your wire or reg declaration: wire signed [9:0] a,b; wire signed [19:0] result = a*b; // signed multiplication!

Figure 5⁶

Using the FPGA shown in Figure 6, the propagation delay would be around 10ns. Hardware multiplier block: two 18-bit twos complement (signed) operands



⁶ Gim P. Hom, Joe Steinmyer, Class Lecture, Topic: "*Arithmetic Circuits & Multipliers*" 6.111 Introductory Digital Systems Laboratory, Massachusetts Institute of Technology, MA, Fall, 2017.Avaiable at: <<u>http://web.mit.edu/6.111/www/f2017/handouts/L08.pdf</u>>

Multiplier Type II Sequential Multiplier



Figure 7

After the combinational multiplier, the sequential multiplier (Figure 7) was also commonly used in early hardware design. It only contains one full adder, four registers in total. A stored a M-bits multiplier, B store a N-bits multiplicand, P is N bit register and C is a single bit register. P and C had initial value of 0. To illustrate the multiple process, an example of 11*11 has been done in Figure 8.



Figure 8

It was doing implicit shifting and adding operation, where P/B are shifting registers, they always shifted right by one each cycle. The mathematical explanation is shown in Figure 9. The shifting operation is determined if shifting and adding both needed to be done or just shifting. The addition step is added at the MSB side, which means adding multiplicand*2^N, N is the number of bits of the multiplier. However, at the end of the multiplication, the value stored in P/B is shifted to right by N bits. The shifting cancels out the effect of adding to the MSB.



Figure 9

Multiplier Type III: Multiplier with Radix 4 and Radix 8.

The radix determines the addition choices in this case. In radix 2 multiplier, the only choice of addition was to add the multiplicand or not add the multiplicand. In radix 4, the choice for addition was not only 1 or 0 but 2*multiplicand and 3*multiplicand. In radix 8 the choices are even more. In radix 2 multiplier 16 additions need to be done for 16 bits multiplication, but with radix 4, only 8 additions need to be done and with Radix 8 it is 4 addition. Normally each addition takes one cycle, hence the number of cycles required could be reduced by increasing the Radix.

For Radix 4 multiplier, Figure 10 in Appendix, the shifting and adding method has also been used, the multiple of the multiplicand have been pre-computed before the additions. Instead

of shifting one bit to the right per cycle, 2 bits were shifted in Radix4 multiplier. The product of this 2 bits number and the multiplicand was added to with the product registers which is the left side to the register B, however it was shifted to the LSB at the end of the multiplication. So, the MSB partial product was not shifted because it is added at the end without any further shifting. Comparing to the normal Radix 2 multiplier, the Radix 4 take 9 additions, one of them was pre-computing the multiple of the multiplicand, it was more efficient than Radix2, the detailed tests regard to latency and power have been discussed in the optimisation section. In theory, higher radix multipliers have more power consumption and lower latency.

The waveform simulation has proved that the cycles required for multiplication have been reduced. Figure 11.



Radix 8 multiplier has the similar structure, Figure 11, but instead of shifting 2 bits, 4 bits were shifted. Therefore, the number of additions had been reduced further.

For the Radix 8 multiplier, more values needed to be pre-computed before the multiplication. However, the multiplication itself only took 4 additions. From the waveform simulation, Figure 12, the result would be available after 4 cycles, however the total number of additions of the Radix8 multiplier were 15 additions in total including 11 pre-computing additions. Clearly Radix 8 multipliers have larger propagation delay due to larger number of additions at the start compared to Radix 4 multipliers.



Figure 12

To conclude, higher radix multiplier meant less additions during the multiplication, but more pre-computed values were required. In this case, radix 8 multiplier need to do 6 more additions for one multiplication compared to radix 4 multiplier. For a 16bits or 32bits word, Radix 4 multiplier would be efficient enough. The radix of the multiplier should be chosen according to the bit length of operands. For the combinational multiplier, it used the most adders but could compute the result with less cycles, and the propagation delay problem could be solved by changing the intermediate adders to be carry save adders, so the carry would be saved instead of added. This would reduce the propagation delay of adding the carry. The final adding stage of the combinational multiplier would involve adding the carries. For the sequential multiplier, it could not be improved and had to take 16 cycles, but it only needed one full adder to do the multiplication.

7 Benchmark tests

7.1 Benchmark: Fibonacci numbers using recursion

Assembly:

R5 is the return register	R6 is the Link Register	R7 is the Stack register			
In Detail		Address	Opcode	Oprand	HEX value
R0 = 2	LDI 2	0	0000	00000000010	0002
R1 = n = 2	MOV R1 R0	1	0011	00000001000	3008
Call Fib(2)	BL Address of FIB	2	1000	004	8004
Stop	STP	3	1011		B000
R0 = 0	LDI 0	4	0000	00000000000	0000
R2 = 0 (y = 0)	MOV R2 R0	5	0011	00000010000	3010
	LDI ELSE's Address	6	0000	ELSE's Address	000C
if (R1>1) JUMP to ELSE	JMPB 1 R1 R0	7	0100	11 0001 001 000	4C48
R0 = 1	LDI 1	8	0000	00000000001	0001
R2 = y = 1	MOV 0 0 0 R2 R0	9	0011	00000010000	3010
Jump to END	LDI END's Address	10	0000	END's Address	0020
	JMP 0 0 R0	11	0100	00000000000	4000
ELSE	STMFD R6	12	1010	000000111110	A03E
Store all the current value	STMFD R1	13	1010	000000111001	A039
	STMFD R2	14	1010	000000111010	A03A
	LDI 1	15	0000	00000000001	0001
R1 = R1 - 1	SUB 0 0 0 R1 R0	16	0110	01000001000	6408
Call Fib(1)	BL Address of FIB	17	1000	FIB's Address	8004
Restore the values form the stack	LDMFD R2	18	1001	00000010111	9017
	LDMFD R1	19	1001	00000001111	900F
	LDMFD R6	20	1001	000000110111	9037
R2 = R5	MOV R2 R5	21	0011	00000010101	3015
	STMFD R6	22	1010	000000111110	A03E
	STMFD R1	23	1010	0 00000 111 001	A039
	STMFD R2	24	1010	0 00000 111 010	A03A
	LDI 2	25	0000	00000000010	0002
R1 = R1 - 2	SUB 0 0 0 R1 R0	26	0110	01000001000	6408
Call Fib(0)	BL Address of FIB	27	1000	FIB's Address	8004
	LDMFD R2	28	1001	0 00000 010 111	9017
	LDMFD R1	29	1001	0 00000 001 111	900F
	LDMFD R6	30	1001	000000110111	9037
R2 = R2 + R5	ADD 0 0 0 R2 R5	31	0101	00000010101	5015
END	MOV R5 R2	32	0011	00000101010	302A
PC = Link Register	JMP 0 0 R6	33	0100	00000000110	4006

Explanation:

Phase One: Calling

Loading the parameter n in to R1, and call the Fibonacci function.

Phase Two: Initializing variable y

Assigning local variable y to R2 by letting R2 = 0.

Phase Three: IF statement

Comparing whether n is bigger than one.

Phase Four: NOT Bigger

Assigning variable y to 1 by letting R2 = 1

Jump to the END phase

Phase Five: Bigger

Store all the values in this call to stack and call the Fibonacci (n-1)

Restore all the values and move the value in the return register(R5) to variable y (R2)

Store all the values in this call to stack again and call the Fibonacci (n-2)

Restore all the values and adding the values in the return register and variable y, and storing

the result back to variable y. Jump to END phase

Phase Six: END

Move the value of variable y (R2) to the return register (R5) Jump to the address stored in the link register

Testing:

Fib(2)

107 cycles

		Value at	0 ps 320.0 ns 640.0 ns 960.0 ns 1.28 us 1.6 us 1.92 us 2.24 us 2.56 us 2.88 us 3.2 us 3.52 us 3.84 us 4.16 us 4.48 us 4.8 us 5.12 us 5.44 us 5.76 us 0	6.08 us	6.4 us 6.	72
	Name	0 ps	0 ps			-
in	CLOCK	В 0		UTUTUUTUTU	UTUTUTUTUTUTUTUTUTUTUTUTUTUTUTUTUTUTUT	M
<u>out</u>	fetch	B 1		uninin	mininini	Ń
out	exec1	В 0		nunnun	www.w	ЛĹ
**	> daddr	H 000		000		+
**	> dout	H 0000		0000		+
**	> instr_a	Н 000	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	003		+
*	> instr	H 0000		B000		+
*	> port1a	dr U0	000000070000000000000000000000000000000	0		1
out.	ldi_sel	B 1				
out	pop_se	В 0				
out	wenou	В 0				_
out	wen	В 0				
out.	carrye	В 0				_
<u>out</u>	carrys	в В 0				
out	write_	ВО			່ານການນາກກ	Jſ
₩	> write_	B 0000	9/8/9/TW8/9/W/////9/TW9/8/8////////	0000		+
out	write_	ВО				
*	> rdin	н 0000	9#&^^^^^	0000		1
**	> dmemir	н 0000		0020		+
**	> q1	н 0000		0020		1
*	> q2	н 0000		0020		+
₩	> R0out	H 0000	\$			1
**	> R1out	н 0000	(%C 0002 X 0001 X 0002 X 0000 X 000	002		+
**	> R2out	H 0000	0000 X0001 X0000 0001 X	0002		1
*	> R3out	H 0000	0000			+
*	> R4out	H 0000	0000			+
*	> R5out	H 0000		0002		+
*	> R6out	H 0000	00X 0003 X 0012 X 0003 X 001C X 0	0003		+
*	> R7out	н 0000		000		+
out	pc_cnt	en B0				_
out						

The final value of R5 (the return register) was 2, which was correct for fib(2)=2.

First section of Fib(2)



The stack pointer R7 could clearly be seen decrementing as values are pushed into the stack in the screenshot above. R6 also updates as the link register.

FIB(5)=8

12.23 us (using 20 ns period clock) = 611 cycles



See Appendix 10 for Fib simulations from fib(2) to fib(7).



Relationship between execution time (using 20 ns period clock) and parameter

As the parameter was increased, the execution time increased exponentially. This was due to the use of a recursive implementation of the Fibonacci function, which required even more calls to itself as the parameter increases. This was an especially inefficient implementation of the function, and it would be interesting to see how a different implementation could be used to decrease execution time.

7.2 Benchmark: Pseudo-random integers with LCG

$x_{n+1} = (ax_n + b) \bmod 2^N$

Using linear congruential generator is one of the ways to generate pseudo- random integer, it takes three parameters a,b,N and one seed value Xn to generate the next value. The sequence generated by the generator is called a linear congruential sequence. The target of this research is to choose the suitable parameters that generate the longest sequence without repeating, so the sequence looks random. Figure LCG shows three examples with fixed sequence cycle.



Wikipedia diagram⁷

Firstly, the generator took a seed input which means the output is related with the input, hence the largest possible length of the sequence is 2^N. In this case, N was fixed at 16 as it is the word length. Ideally the maximum length of the sequence is 65536.

The first case is b= 0, a is a primitive element of 2^16, and assume 2^N is a prime number. The length of the sequence without repeating would be 2^16-1=65535. This special case is called "Lehmer random number generator". However, 2^N is not a prime number, so this is not a possible case. When b=0, 2^N is a power of 2, these parameters are commonly used because it is convenient for binary representation. This form has maximum sequence length of 2^N/4 when a=3 or a=5 and the initial seed input Xn is odd. The final case is when b does not equal zero.

⁷ Wikipedia, Linear Congruential generator.[online] available at: <<u>https://en.wikipedia.org/wiki/Linear_congruential_generator</u>>
According to Hull-Dobell Theorem8, if

b and 2^N are relative prime,
 a-1 is divisible by all prime factor of 2^N
 a-1 is divisible by 4 if 2^N is divisible by 4

The period of the sequence is equal to 2^N.

Therefore, for this implementation if b=1, $a=2^{15}+1=32769$, N=16, in theory the sequence would have a length of 65536, which is much too large for the execution time of the Quartus simulation

Assembly:

Pseudo Random Integer					
	General Instruction	OPCODE	OPERAND	Address	Hex
R1 = a = 25385	LDR 0 0 R1 R0	0001	00000001000	0	1008
R2 = b = 3	LDR 0 1 R2 R0	0001	000001010000	1	1050
R3 = n = 8	LDR 0 2 R3 R0	0001	000010011000	2	1098
R4 = s = 0 = y	LDR 0 3 R4 R0	0001	000011100000	3	10E0
R5 = sum = 0	LDR 0 3 R5 R0	0001	000011101000	4	10E8
LOOP	LDI END's address	0000	012	5	0012
	JEQ 0 R3 R0	0100	010000011000	6	4418
	MUL R4 R1	0111	000000100001	7	7021
Wait for MUL to finish	LDI 0	0000	00000000000	8	0000
	LDI 0	0000	00000000000	9	0000
	LDI 0	0000	00000000000	10	0000
	LDI 1	0000	00000000001	11	0001
	MOV 0 0 0 R4 R2	0011	000000100010	12	3022
Move LSB of product to	MOV 0 0 3 R4 0	0011	000011100000	13	30E0
R4	ADD R5 R4	0101	000000101100	14	502C
	SUB R3 R0	0110	010000011000	15	6418
	LDI Loop's address	0000	00000000005	16	0005
	JMP R0	0100	00000000000	17	4000
END	STP	1011	00000000000	18	B000
			Data	0	6329
				1	0003
				2	0008
				3	0000

⁸ Linear Congruential Generator I section two, Cornell Department of Mathematics. [online] available at:

<<u>http://pi.math.cornell.edu/~mec/Winter2009/Luo/Linear%20Congruential%20Generator/linear%2</u> <u>Ocongruential%20gen1.html</u>>

The value stored in R0 is initialised with 0. Firstly, load R1 with 25385 as the "a" value, load R2 with 3 as the "b" value, load R3 with 8 as the "n" value. R4 stores "y" value, R5 stores "sum" value. "Y" and "sum" are initialised with 0 as well. Then write the stop address to R0, compare the value 0 and the "n" value stores in R3, if they are equal, jump to the END 'address otherwise R4 multiplies with R1, the "y" and the "a" value. During the multiplication, write value 1 to R0. Once the multiplication is done, the least significant 16 bits have been stored in R4. Then the value stores in R5 is added with R4 which represent adding y value to the sum. After that, the n value is subtracted by 1 and jump back to the start of the loop where the comparison between n value and 0 has been done. The loop continues and at the end of each loop n value is subtracted by 1. Once the n value is one, the conditional jump will jump to the stop address and the test code for pseudo random integer has finished. The final random integer is stored at R5.

Testing:

Using "typical" parameters in overview A=25385=0x6329, b=3, n=8 213 cycles Execution time: 4.27 us Pattern: 0, 3, 10625, 34994, 47758, 51917, 49639, 25364, 39500 (does not seem to repeat)

	Name	Value at	0 ps 320.0 ns	640.0 ns	960.0 ns	1.28 us	1.6 us	1.92 us	2.24 us	2.56 us	2.88 us	3.2 us	3.52 us	3.84 us	4.16 us	4.48 us 4
out		0 ps					-		NEW							
*	> dout	H 0000	K32XHXHX00X 6329		6329 <u>X</u> 00	000 X 6329	<u>χ 0000 χ</u>	6329	1 0000 X	6329 X 000	6329		6329 X	000 X 6329		8.
-	> instr_ad.	Н 000	(1)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)	00000000000				5255555555								
-	> instr	H 0000	KUXUXUXUXUXU1XUXXX0000	X0000000X000		(30,X)(X)(X)(X)(X)(X)(X)(X)(X)(X)(X)(X)(X)(X	20000000	000000000000000000000000000000000000000	0000000000	X+X;X 0000 X+X;X+X;X	(x)(x)(x) x(x)(x)(x)	2000000000	X:XXX0000 X:XXX	000000000		XXXX
-	> port1add	r U0	X1X2X3X4X5X0X3X4X_0	_X 4 X5X3X_0_	X3X4X_0_X 4 X	(5X3X_0_X3X4X_0	X_X\$X\$X	0_X3X4X_0_	X 4 X5X3X 0	X3X\$XX 4 X5	(3X_0_X3X4X_0	<u>X 4 X5X3X 0</u>	X3X4X_0_X4	X5X3X_0_X3X4X	0 <u>X</u> 4 X5X3X	
eut	ldi_sel	B 1										ا لیبیا ل				UU
<u>ut</u>	pop_sel	B 0														
<u>eut</u>	wenout	B 0		ກກກກມ	լլորորո	MULIN	າກກາກກາ		տորուն		տուրու	տտու	Ատո	տուու	ກກກກການ	Առևտոտո
<u>94</u>	wen	B 0		mmu	LMM	mnLM	mmm	տեռու	տտու	Lmm	տուրու	տտու	LMM	տուու	MMMM	Առևտոտո
<u>eut</u>	carryen	B 0														
<u>eut</u>	carrysta.	B 0														
<u>eut</u>	write_n	. ВО	mmmmm	mmm	www.www	mmm	າກກາກກ	າທາກການ	າກກາກກາ	mmm	mmm	າກກາກກາ	mmm	www.	ກາກການ	mmmm
**	> write_n	. B 0000	000000000000000000000000000000000000000	00 10 00 00	()()() 0000 (10		00 X10XXX	000	0 (10)()()(00)			00 \10\()()(00		10/00/00/0/00	0000 \10\()(x00)X/0X
<u>out</u>	write_n	. B0	Π													
*	> rdin	H 0000	32(1)(1)(00)(51)(32)(0000		321,0000,000	((())))))))))))))))))))))))))))))))))))	0000000000	()()32(000)	000000000000000000000000000000000000000	32 0000 0000	0000	00000000	32,0000,000	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX		3600C
**	> dmemin	H 0000						00000		()(()(0000)()()()()		0)(()()()()()(00			0000	00:00:00
*	> q1	U 0	000000 0 00000 0)#X3XXXX(5)	(7)3)(0)(3)(XXXX 5 X8X8XX 0	XOSCOCC	5 (3)(3)(0	\$}}})()()(()(()	(4?)(•)()3()()	()((5)(3) ()(0)))))))))))))))))))))))))))))))))))))	x2x2X • XX	XXXXXX 5 XXXXXX	• XX3XXXXXX	x 5 X0X
-	> q2	U O	0 (0)00	0,		8(1)(8)(3)(0)(2)(3))(0)333(1)3(1)3	(\$(0)(3)() •)	3,3,9,9,9,9,6,6	0,	(1,9,3,0,2,3)(0	X3X3X1X2X1X8X5	30300 333	1,2,1),3,6,6,(),3,()	0 333(1)(1)(1)	3(9)(0(22)
-	> R0out	U O	0 22 0	χ 1 χ 5	22 0 1	X 5 X 22 X 0	X I X	5 22 0	<u>(1)</u> (5)	22 0 1	X 5 X 22 X 0	χ 1 χ 5	22 0 0	1 (5) 22)	D X 1	5 22
-	> R1out	U O	0													
-	> R2out	UO	σx													
	> R3out	U O	0 X 8	χ	7	χ 6	Υ	5	Υ.	4	χ 3	γ-	2	-γ	1 χ	
	> R4out	U O	0	- <u>v</u>	з У	10622	- Yay	24369	XX 1	2764 33	4159	- 131	63258	41261	/	
944	> R5out	110	0	γ	3	10625	ΗγH	34994		47758 Y	51917		49639	2536	4 V	
_	> R6out	0.0				A	^			^				^	·	
944	> DZout															
out	>	4 11 0		0		10510	HV-111	24266	V	761	4166	-v	2066 V	41059		
eut	> mutres_	2 110		0		10018	=\$===	4444	~ ~	420	4044	et en	A 1610	91200		
ent.	> mui_res_	2 00				,	<u> </u>							24002		
-	pc_cnt_e	n 00		10000000	00000000000		0000000	LUUUUUUU	UUUUUUUU			UUUUUUUU	000000000	0000000000	1000000000	
	pc_sload	B 0		الا		0					U	ļ				J (L) (L
eut	ram_wre	n B0														

The outputs were changed to the radix unsigned integer to read the values more easily.

Even when increasing the length of the loop, the integers never seemed to repeat due to the small execution time of the Quartus simulation.

Execution time scaled linearly as the length of the for loop increased, as the for loop just executed the corresponding number of times, where each loop required the same amount of time.

Detail	Instructions	Opcode	Oprand	Address	Hex
LDI value to be found	LDI 007	0000	7	0	0007
R1 = x	MOV R1 R0	0011	00000001000	1	3008
LDI address of LOOP	LDI 007	0000	7	2	0007
R3 = address of LOOP	MOV R3 R0	0011	00000011000	3	3018
LDI address of END	LDI 00D	0000	D	4	000D
R4 = address of END	MOV R4 R0	0011	00000100000	5	3020
R0 = head	LDI head	0000	1	6	0001
LOOP: R2 = value in the linked list	LDR R2 R0	0001	00000010000	7	1010
R2 = R2 - x	SUB R2 R1	0110	010000010001	8	6411
Checks if head->value == x	JEQ 0 R2 R4	0100	010000010100	9	4414
Load the next address of node	LDR R0 [R0+1]	0001	000001000000	10	1040
Jumps if end of list reached	JEQ 0 R0 R4	0100	01000000100	11	4404
	JMP R3	0100	00000000011	12	4003
END	MOV R5 R0	0011	00000101000	13	3028
	STP	1011		14	B000

7.3 Benchmark: Traverse linked list to find an item Assembly:

Note that to change the x value (value being searched for in the list), the first instruction needed to be changed to this x value. The above code looked for the number 7 in the list.

First the target value is loaded in R0 and stored in R1 in the next instruction, then load R0 with the beginning address of the loop and stores the address in R3 in the next instruction. After that load R0 with the end address of the loop and store the end address in R4 in the next instruction. After all the initialisation, R0 is loaded with the head pointer which contain the address of the first value, the value is compared with the target value by subtraction method. If it is the target value, jump to the end address and stored the pointer in R5, the test is finished. If it is not the target value, loading R0 with pointer pointing to the next value in the list and jump back to the beginning of the loop, this loop will stop when it iterates through the whole list or when the target value has been found. NULL was defined as the memory

address 0. The item was found if R5 (register 5) returns the address of the item with value x. If it remains at 0, the item was not found in the list.

Data memory:



This test data was used for the simulations (the list continues on the right side). One item in the linked list consisted of the value of the item and the value in the next memory address is the memory address pointing to the next item in the linked list (each item's value was highlighted in the image). The memory location storing a memory address acted like a sort of pointer. In this example, memory address 1 is the head item, which had a value of 1 and the next item in the list is at the memory address 3 (as the value in memory address 1+1 is 3). The length of this list was 20 items, although the actual length of the simulation depended on what value was being searched for in the list, as once it was found, the simulation stopped. The length was tested for 7, 10, 15, and 20 items (all simulation results for "find in list" benchmark found in Appendix 12).

Testing:

Length 10 (typical):

To test a length 10 linked list, the first instruction needed to be changed to 000A, as the linked list values are in numerical order, and thus it would stop at 10 items when 0xA or 10 is found as the value in the tenth item.

149 cycles

2.99 us



Relationship between execution time and linked list size



Execution time seemed to increase linearly with linked list size as each item took up the same amount of time. It would be interesting to investigate quicker search algorithms, possibly using binary search trees.

8 Optimisation

8.1 Basic CPU evaluation

The speed, power and area were measured using the full compilation on Quartus. This evaluation is

for the unpipelined CPU.



To find the geometric mean time, a parameter was fixed for each benchmark tests that obtained similar execution times between each test. This allowed comparison between geometric mean times.

FIB(3) = 3 191 cycles = 3.149 us

LCG typical parameters A=25385=0x6329, b=3, n=8 213 cycles = 3.512 us

Length 15 linked list: 219 cycles = 3.611 us

Geometric mean time = 3.42 us

Analysis of CPU block:

Timing analysis:

Table of Contents 📮 d	Slo	w 1200mV	85C Model Setup: 'clock'						
Flow Summary	~	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
Elow Settings	1	-0.487	instr_mem:inst alts~porta_address_reg0	data_mem:inst2 altporta_address_reg0	clock	clock	16.000	-0.060	16.475
Elow Non-Default Global Set	2	-0.470	instr_mem:inst alts~porta_address_reg0	data_mem:inst2 altporta_address_reg0	clock	clock	16.000	-0.058	16.460
Elaw Elanad Time	3	-0.215	instr_mem:inst alts~porta_address_reg0	data_mem:inst2 altporta_address_reg0	clock	clock	16.000	-0.056	16.207
Flow Elapsed Time	4	-0.212	instr_mem:inst alts~porta_address_reg0	data_mem:inst2 altporta_address_reg0	clock	clock	16.000	-0.057	16.203
How OS Summary	5	-0.209	instr_mem:inst alts~porta_address_reg0	data_mem:inst2 altporta_address_reg0	clock	clock	16.000	-0.060	16.197
Flow Log	6	-0.201	instr_mem:inst alts~porta_address_reg0	data_mem:inst2 altporta_address_reg0	clock	clock	16.000	-0.056	16.193
> Analysis & Synthesis	7	-0.198	instr_mem:inst alts~porta_address_reg0	data_mem:inst2 altporta_address_reg0	clock	clock	16.000	-0.054	16.192
> Fitter	8	-0.195	instr_mem:inst alts~porta_address_reg0	data_mem:inst2 altporta_address_reg0	clock	clock	16.000	-0.055	16.188
> Assembler	9	-0.192	instr_mem:inst alts~porta_address_reg0	data_mem:inst2 altporta_address_reg0	clock	clock	16.000	-0.058	16.182
TimeQuest Timing Analyzer	10	-0.184	instr_mem:inst alts~porta_address_reg0	data_mem:inst2 altporta_address_reg0	clock	clock	16.000	-0.054	16.178
E Summary	11	-0.163	instr_mem:inst alts~porta_address_reg0	data_mem:inst2 altporta_address_reg0	clock	clock	16.000	-0.059	16.152
Parallel Compilation	12	-0.146	instr_mem:inst alts~porta_address_reg0	data_mem:inst2 altporta_address_reg0	clock	clock	16.000	-0.057	16.137
	13	-0.135	instr_mem:inst alts~porta_address_reg0	data_mem:inst2 altporta_address_reg0	clock	clock	16.000	-0.066	16.117
SDC File List	14	-0.118	instr_mem:inst alts~porta_address_reg0	data_mem:inst2 altporta_address_reg0	clock	clock	16.000	-0.064	16.102
Clocks	15	0.143	instr_mem:inst alts~porta_address_reg0	data_mem:inst2 altporta_address_reg0	clock	clock	16.000	-0.058	15.847
Slow 1200mV 85C Mode	16	0.160	instr_mem:inst alts~porta_address_reg0	data_mem:inst2 altporta_address_reg0	clock	clock	16.000	-0.056	15.832
Fmax Summary	17	0.220	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_ale lpm_ff:R0 dffs[13]	clock	clock	16.000	-0.379	15.416
Timing Closure Recc	18	0.237	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_ale lpm_ff:R0 dffs[13]	clock	clock	16.000	-0.377	15.401
setup Summary	19	0.386	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_ale lpm_ff:R6 dffs[11]	clock	clock	16.000	-0.045	15.584
Hold Summary	20	0.390	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_ale lpm_ff:R2 dffs[11]	clock	clock	16.000	-0.013	15.612
Recovery Summary	21	0.403	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_ale lpm_ff:R6 dffs[11]	clock	clock	16.000	-0.043	15.569
E Removal Summary	22	0.407	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_ale lpm_ff:R2 dffs[11]	clock	clock	16.000	-0.011	15.597
Minimum Dulan Middle	23	0.414	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_ale lpm_ff:R5 dffs[11]	clock	clock	16.000	-0.017	15.584
minimum Puise Widtr	24	0.428	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_ale lpm_ff:R6 dffs[15]	clock	clock	16.000	-0.045	15.542
Worst-Case Timing I	25	0.431	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_ale lpm_ff:R5 dffs[11]	clock	clock	16.000	-0.015	15.569
Setup: 'clock'	26	0.445	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_ale lpm_ff:R6 dffs[15]	clock	clock	16.000	-0.043	15.527
Hold: 'clock'	27	0.456	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_ale lpm_ff:R5 dffs[15]	clock	clock	16.000	-0.017	15.542
Metastability Summa	28	0.467	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_aile lpm_ff:R0 dffs[7]	clock	clock	16.000	-0.016	15.532
> Slow 1200mV 0C Model	29	0.469	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_aile lpm_ff:R6 dffs[5]	clock	clock	16.000	-0.045	15.501
> Fast 1200mV 0C Model	30	0.473	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_ale lpm_ff:R5 dffs[15]	clock	clock	16.000	-0.015	15.527
Multicorner Timing Analy	31	0.484	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_aile lpm_ff:R0 dffs[7]	clock	clock	16.000	-0.014	15.517
	32	0.486	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_aile lpm_ff:R6 dffs[5]	clock	clock	16.000	-0.043	15.486
	33	0.499	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_aile lpm_ff:R5 dffs[5]	clock	clock	16.000	-0.017	15.499
	34	0.515	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_aile lpm_ff:R6 dffs[6]	clock	clock	16.000	-0.045	15.455
Report ICCS	35	0.516	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_aile lpm_ff:R5 dffs[5]	clock	clock	16.000	-0.015	15.484
Report RSKM	36	0.532	instr_mem:inst alts~porta_address_reg0	regfile_alu:regfile_aile lpm_ff:R6 dffs[6]	clock	clock	16.000	-0.043	15.440
IInconstrained Dathe									

The largest propagation delay in the CPU_block bdf was between the instruction and data memories. In the CPU, there is a large propagation delay due to the entire loop being completed in one cycle and being required for the next cycle. One way to decrease this propagation delay is to add pipelining stages. A register could be added at the output of the data RAM so that the output is ready at the next cycle. This would involve changes to the state machine so that all the stages have a value in each register, and then this could be pipelined so that each stage has a very log propagation delay, thus increasing the max clock frequency (discussed further in Extension).

Power:

Total Thermal Power Dissipation	235.22 mW
Core Dynamic Thermal Power Dissipation	19.63 mW
Core Static Thermal Power Dissipation	49.78 mW
VO Thermal Power Dissipation	165.81 mW

Table of Contents 🛛 📮 🗗	The	rmal Power Dissipation by Hierarchy				
Flow Summary		Compilation Hierarchy Node	Total Thermal Power by Hierarchy (1)	Block Thermal Dynamic Power (1)	Block Thermal Static Power (1)(2)	Routing Thermal Dynamic F
Flow Settings	1	 /CPU_block 	164.84 mW (148.10 mW)	38.09 mW (27.57 mW)	117.64 mW (117.64 mW)	9.11 mW (2.88 mW)
Elow Non-Default Global Settings	1	hard_block:auto_generated_inst	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)		0.00 mW (0.00 mW)
Elaw Elapsed Time	2	✓ busmux:bl_MUX	0.02 mW (0.00 mW)	0.01 mW (0.00 mW)		0.01 mW (0.00 mW)
	1	Ipm_mux:\$00000	0.02 mW (0.00 mW)	0.01 mW (0.00 mW)		0.01 mW (0.00 mW)
Flow US Summary	1	mux_psc:auto_generated	0.02 mW (0.02 mW)	0.01 mW (0.01 mW)		0.01 mW (0.01 mW)
Flow Log	3	✓ instr_mem:inst	4.40 mW (0.00 mW)	4.20 mW (0.00 mW)		0.21 mW (0.00 mW)
Analysis & Synthesis	1	 altsyncram:altsyncram_component 	4.40 mW (0.00 mW)	4.20 mW (0.00 mW)		0.21 mW (0.00 mW)
> 📙 Fitter	1	altsyncram_jda1:auto_generated	4.40 mW (4.40 mW)	4.20 mW (4.20 mW)	-	0.21 mW (0.21 mW)
> Assembler	4	✓ data_mem:inst2	4.59 mW (0.00 mW)	4.31 mW (0.00 mW)		0.28 mW (0.00 mW)
> ImeQuest Timing Analyzer	1	 altsyncram:altsyncram_component 	4.59 mW (0.00 mW)	4.31 mW (0.00 mW)		0.28 mW (0.00 mW)
> EDA Netlist Writer	1	altsyncram_eeh1:auto_generated	4.59 mW (4.59 mW)	4.31 mW (4.31 mW)		0.28 mW (0.28 mW)
PowerPlay Power Analyzer	5	✓ lpm_counter:PC	0.35 mW (0.00 mW)	0.04 mW (0.00 mW)		0.31 mW (0.00 mW)
Summany	1	[cntr_enh:auto_generated	0.35 mW (0.35 mW)	0.04 mW (0.04 mW)		0.31 mW (0.31 mW)
Collinea	6	✓ regfile_alu:regfile_alu	7.38 mW (0.04 mW)	1.97 mW (0.02 mW)		5.41 mW (0.02 mW)
Settings	1	alu1:alu1_block	1.91 mW (1.91 mW)	0.49 mW (0.49 mW)		1.42 mW (1.42 mW)
Operating Conditions Used	2	llpm_ff:carry_reg	0.02 mW (0.02 mW)	0.01 mW (0.01 mW)		0.01 mW (0.01 mW)
Thermal Power Dissipation by E	3	 busmux:dinMUX 	0.53 mW (0.00 mW)	0.09 mW (0.00 mW)		0.45 mW (0.00 mW)
Thermal Power Dissipation by E	1	Ipm_mux:\$00000	0.53 mW (0.00 mW)	(Win 00.0) Win e0.0		0.45 mW (0.00 mW)
Thermal Power Dissipation by H	1	[mux_tsc:auto_generated	0.53 mW (0.53 mW)	0.09 mW (0.09 mW)		0.45 mW (0.45 mW)
Core Dynamic Thermal Power D	4	✓ busmux:inst	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	-	0.00 mW (0.00 mW)
> Current Drawn from Voltage Su	1	[lpm_mux:\$00000	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)		0.00 mW (0.00 mW)
Confidence Metric Details	5	Ibusmux:LDI_MUX	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)		0.00 mW (0.00 mW)
Sinnal Activities	1	[lpm_mux:\$00000	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)		0.00 mW (0.00 mW)
	6	✓ busmux:link_MUX	0.27 mW (0.00 mW)	0.01 mW (0.00 mW)		0.26 mW (0.00 mW)
messages	1	Ipm_mux:\$00000	0.27 mW (0.00 mW)	0.01 mW (0.00 mW)		0.26 mW (0.00 mW)
Flow Messages	1	[mux_9rc:auto_generated	0.27 mW (0.27 mW)	0.01 mW (0.01 mW)		0.26 mW (0.26 mW)
 Flow Suppressed Messages 	7	lpm_ff:mul_A	0.14 mW (0.14 mW)	0.04 mW (0.04 mW)	-	0.10 mW (0.10 mW)
	8	llpm_ff:mul_B	0.08 mW (0.08 mW)	0.04 mW (0.04 mW)		0.04 mW (0.04 mW)
	9	[multwithRadix4:mul_radix4	0.85 mW (0.85 mW)	0.47 mW (0.47 mW)		0.38 mW (0.38 mW)
	10	lpm_ff:mul_res1	0.20 mW (0.20 mW)	0.02 mW (0.02 mW)	-	0.18 mW (0.18 mW)
	11	llpm_ff:mul_res2	0.17 mW (0.17 mW)	0.02 mW (0.02 mW)		0.15 mW (0.15 mW)
	12	 Ibusmux:pop_MUX 	0.00 mW (0.00 mW)	(Vilm 00.0) Wim 00.0	-	0.00 mW (0.00 mW)
	1	[lpm_mux:\$00000	0.00 mW (0.00 mW)	(Wm 00.0) Wm 00.0		0.00 mW (0.00 mW)
1	13	✓ Ireaisterfile:reafile	3 03 mW (0 00 mW)	0.68 mW (0.00 mW)		2.35 mW (0.00 mW)

Area:

Table of Contents 📮 🗗	Fitter	Resource Usage Summary	
Flow Summary		Resource	Usage
Flow Settings	1	✓ Total logic elements	808 / 15,408 (5 %)
Elow Non-Default Global Setting	1	Combinational with no register	560
Flow Flansed Time	2	Register only	23
	3	Combinational with a register	225
	2		
	3	✓ Logic element usage by number of LUT inputs	
Analysis & Synthesis	1	4 input functions	365
V / Fitter	2	3 input functions	280
Summary	3	<=2 input functions	140
E Settings	4	Register only	23
Parallel Compilation	4		
VO Assignment Warnings	5	 Logic elements by mode 	
> Incremental Compilation Sec	1	normal mode	534
Pin-Out File	2	arithmetic mode	251
Resource Section	6		
Resource Usage Summ	7	✓ Total registers*	248 / 17,056 (1 %)
	1	Dedicated logic registers	248 / 15,408 (2 %)
	2	VO registers	071,648(0%)
	0	Total LARa: partially or completely used	52 (062 (6 9/)
Output Pins	10	Virtual pipe	0
Dual Purpose and Dedic	10	Virtual pins	201/244/99.94 \
VO Bank Usage	1	Clock nine	1/7/14%)
All Package Pins	2	Dedicated input nins	0/9(0%)
Resource Utilization by	12	- Dedicated input pins	0,0(0,0)
Delay Chain Summary	13	Global signals	1
Pad To Core Delay Chai	14	M9Ks	16 / 56 (29 %)
Control Signals	15	Total block memory bits	131,072 / 516,096 (25 %)
Global & Other Fast Sig	16	Total block memory implementation bits	147,456 / 516,096 (29 %)
RAM Summary	17	Embedded Multiplier 9-bit elements	0/112(0%)
	18	PLLs	0/4(0%)
Logic and Pouting Section	19	Global clocks	1/20(5%)
	20	JTAGs	0/1(0%)
	21	CRC blocks	0/1(0%)
	22	ASMI blocks	0/1(0%)
Operating Settings and Con	23	Oscillator blocks	0/1(0%)
 Messages 			

Power and area were more difficult to change than speed, although they were taken into account when choosing the multiplier to carry out multiplication.

8.2 Choosing the multiplier

To maximise the clock frequency of the CPU, the propagation delay needed to be minimised. Therefore, the power and latency analysis were done with the Radix 2 multiplier (Figure analysis Radix 2 Appendix 8) and the Radix 4 multiplier (Figure analysis Radix 4 Appendix 7). After running the full compilation with model Cyclone IV E, the TimeQuest Timing Analyzer showed the time analysis. The set-up slack in time analysis means the difference between the data required time and the data arrival time. For the radix 4 multiplier, the worst-case slack is 0.395, for the radix 2 multiplier is 0.677. The radix 4 had half the cycles, 0.395*2>0.677. The result showed that the propagation delay for radix 4 multiplier was less compared to radix 2. The positive slack in each simulation meant there were still some margin for both multipliers to increase the clock frequency further.

The dynamic power is the power consumed when inputs are active, the static power is the power consumption the inputs are kept constant, it is usually caused by DFF. For the Radix 2 multiplier, total power dissipation was 97.79mV, core dynamic thermal power dissipation was 6.19mW, and core static thermal power dissipation was 42.86mW. For Radix 4 multiplier, total power dissipation was 102.71mW, core dynamic thermal power dissipation was 7.42mW, and core static thermal power dissipation was 42.88mW. The result makes perfect sense as the high dynamic power dissipation for radix 4 was because it shifted more bits in the same amount of time. The similar static power was because the same number of additions had been done in the same amount of time. Even the Radix 4 multiplier had a slightly larger total power dissipation, however it could do double the work that radix could do with the same amount of time. Hence, the radix 4 multiplier was the most efficient choice.

8.3 Pipelining

Pipelining⁹ involved creating a state machine that fetched the next instruction and executed the current instruction during the same cycle. This approximately halved the number of cycles required as fetch and exec1 happen at the same time.

Pipelined state machine:

EXEC

⁹ Torsten Grust, Class Lecture, Topic "*Pipelining*" Database Systems and Modern CPU Architecture, Eberhard Karls University of Tübingen, German, 2009. Available at: <<u>https://db.inf.uni-</u> <u>tuebingen.de/staticfiles/teaching/ss09/dbcpu/dbms-cpu-2.pdf</u>>

The new state machine executes FETCH only for the first cycle and then fetches the next instruction while executing the current instruction during EXEC1.



A reset pin was required for the DFF, so that FETCH starts at 1 at the beginning.

	Nama	Value at	0 ps	40.0 ns	80.0 ns	120.0 ns	160.0 ns	200.0 ns	240.0 ns
	Name	0 ps	0 ps						
in	CLK	В 0							
out	FETCH	B 1							
out	EXEC1	В 0							
in	rst	В 0							

Pipeline hardware (see Appendix 2):

To increase PC during the first cycle, an add_one block was used for all instructions (adds one to output of PC). A MUX chose between PC+1 and PC depending on add_sel, so it should always increase by 1 other than for stop instructions, where it should not increase again. Otherwise, the PC would count above the STP instruction and skip it.

Hardware:



Pipelined control signals logic

```
// PC and RAM control signals
assign pc_sload = exec1 & (jmp_cond | bl);
assign pc_cnt_en = exec1 & ~(jmp_cond | stp | bl | write_next_stp);
assign ram_wren = exec1 & (str | stmfd);
assign add_sel = exec1 & ~(stp | jmp_cond | bl | write_next_stp);
```

For pipelining, JMP and BL should not increase the PC, and they should not use PC+1, as this skips the address being jumped to. Write_next_stp (the delay for write instructions after an LDR) also had the same logic as it should stop the PC from counting during this cycle. BL also required an additional MUX at the input of the instruction memory so that the jump address updates during the same cycle as the BL instruction being executed, so that the next instruction already executes the instruction that was jumped to as it should be available at the memory output.

Improved Multiple LDR instructions Test:

Pipelined:

		Neme	Value at	0 ps 40.0 ns	80.0 ns 1	120.0 ns	160.0 ns	200.0 ns	240.0 ns	280.0 ns	320.0 ns	360.0 ns	400.0 ns	440.0 ns	480.0 ns	520.0 ns	560.0 ns	600.0 ns 6
		Name	0 ps	0 ps														
<u>in</u>	(clock	В 0															
eut		fetch	B 1															
out 🌥		exec1	В 0															
5	> (daddr	H 000	000 X	001 (000)												000	
*	> (dout	H 0000	0004	0002												0004	
₩	>	pc_out	H 000	000 X 0	01 🗙 002 🗙 003 🔪												004	
₩	> i	instr_ad	H 000	000 X 001 X 0	02 003 004												004	
*	> i	instr	H 0000	0001 (1	008×1010×101A												B000	
₩	>	port1addr	B 000	000 X	001 (010)	011											000	
<u>ut</u>		ldi_sel	B 1															
<u>°ut</u>		wenout	В 0															
out		wen	В 0															
<u>out</u>		carryen	В 0															
<u>eut</u>		carrysta	В 0															
out		write_n	В 0															
₩	>	write_n	B 0000	0000	(1001)(1010)	(1011)											0000	
<u>sut</u>	,	write_n	в 0															
₩	> 1	rdin	H 0000	000 0001 X0	004 0002												0004	
#	> (dmemin	н 0000	0000	0000												0001	
₩	> (q1	н 0000	0000	0000												0001	
*	> (q2	H 0000	0000 X	0001 (0000)												0001	
₩	>	R0out	H 0000	0000 X												0001		
#	> 1	R1out	H 0000	0000	X											0	1002	
#	> 1	R2out	H 0000	000	10												0002	
*	>	R3out	н 0000		0000												0004	
#	>	R4out	н 0000												00	00		
#	> 1	R5out	н 0000												00	00		
#	> 1	R6out	н 0000												00	00		
out																00		

Unpipelined:

			· ·	
in —		CLOCK	B 0	
out		FETCH	B 1	
out		EXEC1	В 0	
8	>	daddr	H 000	000 X 001 X 002 X 000
*	>	dout	H 0000	0004 X 0002 X 00AB X 0004
*	>	instr_ad	H 000	(000 X 1001 X 1002 X 1003 X 1004
*	>	instr	H 0000	0X 0001 X 1008 X 1010 X 101A X B000
*	>	port1addr	B 000	000 X 001 X 010 X 011 X 000
out		ldi_sel	B 1	
out		wenout	В 0	
out		wen	В 0	
out		carryen	В 0	
out		carrysta	В 0	
out		write_n	В 0	
#	>	write_n	B 0000	0000 X 1001 X 1010 X 1011 X 0000
#	>	q1	H 0000	(0000 X00 X 0000 X00 X00 X00 X00 X00 X00
#	>	q2	н 0000	<u>0000 X 0001 X 0002 X 0001 0001</u>
*	>	R0out	н 0000	(0000X 0001
*	>	R1out	H 0000	0000 X 0002
*	>	R2out	н 0000	0000 X 0002
*	>	R3out	H 0000	0000 X 00AB
*	>	R4out	H 0000	0000

Clearly the pipelined version was able to execute the LDR instructions in one cycle rather than two cycles for the unpipelined version. This test was already optimised due to the LDR implementation allowing for LDR instructions to be carried out in parallel, but the pipelining allowed for even better optimisation in terms of number of cycles.

Multiplication while pipelining

More instructions had to be added after MUL instruction so that 9 instructions or cycles pass to give the multiplier time to obtain the output. This could not be decreased using pipelining, which meant that the LCG benchmark (the one using multiplication) did not have as great an improvement in execution time compared to the other benchmarks, as the majority of the time was taken up by the multiplication. While the ability to carry out instructions in parallel was possible, this could not be fully taken advantage of for this specific benchmark as not many instructions could be executed in parallel that would actually lead to improved efficiency.

Pipelined CPU benchmark tests (using fixed parameters for comparison)



Pipelined: FIB(3) = 3

Pipelined: LCG with typical parameters:

A=25385=0x6329, b=3, n=8

163 cycles

	Name	Value at 0 ps	0 ps 0 ps	320.0 ns	640.0 ns	960.0 ns	1.28 us	1.6 us	1.92 us	2.24 us	2.56 us	2.88 us	3.2 us 3.52 u
*	> dout	H 0000		6329 🔾	0 6329	X00X 6329	X00X 6329	X00)X	6329 X	00) 6329	X0XX 6329	X00X 6329	
*	> instr_ad	H 000	0000000000	000000000000000000000000000000000000000	000000000000000000000000000000000000000	000000000000000000000000	000000000000000000000000000000000000000		X0000X00000(MARCONCORONNA		MANANA ROOMAN	200000000000000000000000000000000000000
*	> instr	н 0000		0000					0000				
#	> port1addr	U 0		0 (4)					<u>a</u>				
out	ldi_sel	B 1											
out	pop_sel	В 0											
out	wenout	В 0			10		tt utt	UU					
out	wen	В 0			10			UU					
out	carryen	B 0											
out	carrysta	B 0											
out	write_n	В 0											
#	> write_n	B 0000	1000000	0000 X	()()0000				0000				000 ()()(00)(01)
out	write_n	в 0											
#	> rdin	н 0000	32000000	0000	() () () () () () () () () () () () () (0000				
*	> dmemin	н 0000	0000	0000	0000	XXXX 0000 0000		00 101100	0000		X XX 000		00 *****
-	> q1	H 0000	0000	0000	0000	10000 X 100 100 100 100 100 100 100 100	10111 (JOIN) 00	00 1/111/001	0000	0000 XX 000			
-	> q2	H 0000	0000	0000	()/0000				0000	<u>)))()</u> 000))())(0000)			00)///////00/
	> R0out	н 0000	0000 \01	000 0000	01/20/21/ 0000	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	X001X00X11X 00	00 /0001/00/01	X 0000 X	001/00/01/ 0000	X0001X00X01X 0000	J (2001)(20)(21)(00	00 \0001\00
-	> R1out	U 0	0χ										
-	> R2out	U 0	οχ										
-	> R3out	UO		8	χ 7	χ 6	χ 5	γ	4	χ 3	X 2	χ 1	×
94	> R4out	UO		γ	3	10622	24369	12	2764	4159	63258	41261	
-	> R5out	UO		0	(3	X 10625	X 34994	Υ Υ	47758	51917	X 49639	25364	X
-	> R6out	UO								,			
-	> R7out	UO											
94	> mul res 1	UO		0		X 10619	X 24366	X 127	61 X	4156	X 63255	X 41258	-γ
94	> mul res 2	UO		0		χ 1	X 4114	 X 94:	39 X	4944	X 1610	24502	₩¥
out	pc cnt en	B 0							^				
out	pc sload	B 0											
out	ram_wren	BO											

Pipelined: Length 15 linked list search:

126 cycles



Each of the tests were clearly executed in less cycles. It was approximately halved from the unpipelined version, other than the LCG due to multiplication causing the same delay.

Pipelined compilation

Slow 1200mV 85C Model Fmax Summary											
	Fmax	Restricted Fmax	Clock Name	Note							
1	63.31 MHz	63.31 MHz	clock								

Fmax increased compared to the unpipelined CPU from 60 MHz to 63 MHz.

Geometric Mean Time (pipelined) = 2.047 microseconds

This was an increase from 3.42 microseconds. It is not exactly halved due to the long delay that remains due to the multiplier block of 9 cycles. This took up the majority of the execution time for the LCG benchmark.

Reg_Alu compilation :

By compiling this bdf, the paths that have the greatest propagation delay could be found.

Table of Contents	5100	v 1200111V	osc model setup: clock						
Flow Summary	1	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
Flow Settings	1	2.531	lpm_ff:write_next_reg dffs[3]	registerfile:regfile lpm_ff:R1 dffs[3]	clock	clock	16.000	-0.088	13.396
Elow Non-Default Global Setting	2	2.535	lpm_ff:write_next_reg(dffs[3]	registerfile:regfile[lpm_ff:R6 dffs[3]	clock	clock	16.000	-0.095	13.385
Elow Elensed Time	3	2.538	lpm_ff:write_next_reg(dffs[3]	registerfile:regfile[lpm_ff:R5[dffs[3]	clock	clock	16.000	-0.095	13.382
	4	2.607	lpm_ff:write_next_reg(dffs[3]	registerfile:regfile[lpm_ff:R7[dffs[3]	clock	clock	16.000	-0.088	13.320
Plow US Summary	5	2.610	lpm_ff:write_next_reg(dffs[3]	registerfile:regfile lpm_ff:R2 dffs[14]	clock	clock	16.000	-0.061	13.344
Flow Log	6	2.611	lpm_ff:write_next_reg[dffs[3]	registerfile:regfile lpm_ff:R1 dffs[14]	clock	clock	16.000	-0.061	13.343
> Analysis & Synthesis	7	2.626	lpm_ff:write_next_reg[dffs[3]	registerfile:regfile lpm_ff:R6 dffs[9]	clock	clock	16.000	-0.084	13.305
> Fitter	8	2.640	lpm_ff:write_next_reg[dffs[3]	registerfile:regfile lpm_ff:R5 dffs[9]	clock	clock	16.000	-0.095	13.280
> Assembler	9	2.653	lpm_ff:write_next_reg[dffs[3]	registerfile:regfile lpm_ff:R2 dffs[9]	clock	clock	16.000	-0.084	13.278
👻 📂 TimeQuest Timing Analyzer	10	2.654	lpm_ff:write_next_regidffs[3]	registerfile:regfile lpm_ff:R0 dffs[9]	clock	clock	16.000	-0.084	13.277
Summary	11	2.658	lpm_ff:write_next_regidffs[3]	registerfile:regfile lpm_ff:R6 dffs[10]	clock	clock	16.000	-0.095	13.262
Parallel Compilation	12	2.659	lpm_ff:write_next_regidffs[3]	registerfile:regfile lpm_ff:R4 dffs[10]	clock	clock	16.000	-0.095	13.261
SDC File List	13	2.659	lpm_ff:write_next_regidffs[3]	registerfile:regfile lpm_ff:R7 dffs[10]	clock	clock	16.000	-0.095	13.261
Clocks	14	2.659	lpm_ff:write_next_regidffs[3]	registerfile:regfile lpm_ff:R5 dffs[10]	clock	clock	16.000	-0.095	13.261
	15	2.666	lpm_ff:write_next_regidffs[3]	registerfile:regfile lpm_ff:R2 dffs[0]	clock	clock	16.000	-0.061	13.288
Slow 1200mV 85C Model	16	2.676	lpm_ff:write_next_regidffs[3]	registerfile:regfile lpm_ff:R0 dffs[14]	clock	clock	16.000	-0.084	13.255
Fmax Summary	17	2.679	lpm_ff:write_next_regidffs[3]	registerfile:regfile lpm_ff:R1 dffs[0]	clock	clock	16.000	-0.061	13.275
Timing Closure Recom	18	2.680	lpm_ff:write_next_regidffs[3]	registerfile:regfile lpm_ff:R6 dffs[0]	clock	clock	16.000	-0.095	13.240
Setup Summary	19	2.682	lpm_ff:write_next_regidffs[3]	registerfile:regfile lpm_ff:R5 dffs[0]	clock	clock	16.000	-0.095	13.238
Hold Summary	20	2.725	lpm_ff:write_next_regidffs[3]	registerfile:regfile lpm_ff:R6 dffs[14]	clock	clock	16.000	-0.095	13.195
Recovery Summary	21	2.726	lpm_ff:write_next_regidffs[3]	registerfile:regfile lpm_ff:R5 dffs[14]	clock	clock	16.000	-0.095	13.194
Removal Summary	22	2.729	lpm_ff:write_next_regidffs[3]	registerfile:regfile lpm_ff:R2 dffs[10]	clock	clock	16.000	-0.084	13.202
Minimum Pulse Width S	23	2.729	lpm_ff:write_next_regidffs[3]	registerfile:regfile lpm_ff:R0 dffs[10]	clock	clock	16.000	-0.084	13.202
Worst-Case Timing Pat	24	2.774	lpm_ff:write_next_regidffs[3]	registerfile:regfile ipm_ff:R4 dffs[3]	clock	clock	16.000	-0.084	13.157
Catar Jalanti	25	2.835	lpm_ff:write_next_regidffs[3]	registerfile:regfile lpm_ff:R2 dffs[3]	clock	clock	16.000	-0.061	13.119
Selup. Clock	26	2.921	lpm_ff:write_next_regidffs[3]	registerfile:regfile lpm_ff:R7 dffs[0]	clock	clock	16.000	-0.088	13.006
Hold: "clock"	27	2.922	lpm_ff:write_next_regidffs[3]	registerfile:regfile ipm_ff:R7 dffs[14]	clock	clock	16.000	-0.095	12.998
Metastability Summary	28	2.923	lpm_ff:write_next_reg(dffs[3]	registerfile:regfile lpm_ff:R4 dffs[14]	clock	clock	16.000	-0.095	12.997
> Slow 1200mV 0C Model	29	2.927	lpm_ff:write_next_regidffs[3]	registerfile:regfile lpm_ff:R4 dffs[9]	clock	clock	16.000	-0.095	12.993
> Fast 1200mV 0C Model	30	2.931	lpm_ff:write_next_reg(dffs[3]	registerfile:regfile[lpm_ff:R1 dffs[9]	clock	clock	16.000	-0.087	12.997
Multicorner Timing Analysis	31	2.939	lpm_ff:write_next_reg(dffs[0]	registerfile:regfile[lpm_ff:R1 dffs[3]	clock	clock	16.000	-0.088	12.988
> Advanced VO Timing	32	2.943	lpm_ff:write_next_reg(dffs[0]	registerfile:regfile[lpm_ff:R6 dffs[3]	clock	clock	16.000	-0.095	12.977
> Clock Transfers	33	2.946	lpm_ff:write_next_reg(dffs[0]	registerfile:regfile[lpm_ff:R5 dffs[3]	clock	clock	16.000	-0.095	12.974
Report TCCS	34	2.960	lpm_ff:write_next_regidffs[3]	registerfile:regfile(ipm_ff:R6)dffs[4]	clock	clock	16.000	-0.095	12.960
Report PSKM	35	3.015	lpm_ff:write_next_reg dffs[0]	registerfile:regfile[lpm_ff:R7 dffs[3]	clock	clock	16.000	-0.088	12.912
Linesestrained Bethe	36	3.022	lpm_ff:write_next_reg dffs[3]	registerfile:regfile[lpm_ff:R4 dffs[11]	clock	clock	16.000	-0.095	12.898
onconstrained Paths	37	3.023	lpm_ff:write_next_reg dffs[3]	registerfile:regfile[lpm_ff:R7 dffs[11]	clock	clock	16.000	-0.095	12.897
U Messages	38	3.028	lpm_ff:write_next_reg dffs[3]	registerfile:regfile[lpm_ff:R4 dffs[4]	clock	clock	16.000	-0.095	12.892
> EDA Netlist Writer	39	3.028	lpm_ff:write_next_regidffs[3]	registerfile:regfile[ipm_ff:R7 dffs[4]	clock	clock	16.000	-0.088	12.899
	40	3.029	lpm_ff:write_next_regidffs[3]	registerfile:regfile lpm_ff:R5 dffs[4]	clock	clock	16.000	-0.095	12.891

The write_next register seemed to cause significant delay to get to the register file, due to the large amount of combinational logic due to logic gates and MUXes. This was a bottleneck to the maximum clock frequency. Pipeline stages could have been used to store the value at intermediate stages in a register and change the state machine so that each stage or cycle only involves little propagation delay. This would increase the maximum clock frequency, but also increase the setup time is takes to fully take advantage of this pipeline. These pipeline stage registers could be added at the output of the data RAM as this seemed to also introduce significant slack, as well as between the register file and ALU.

9 Conclusion/Extension

Making the CPU was very interesting as it built off the work done during term 2 very well. One could see the CPU becoming increasingly advanced as it incorporated instructions from modern ISAs and computer architectures. The CPU met the benchmark tests that were written in C++, so the link between programming and computer architecture could be explored. Meeting the design goal of creating a general CPU was satisfying as there are many different applications that can be tested.

There are several ways for the CPU to improve and to investigate further if there was more time. Firstly, in terms of the memory blocks, a dual port RAM could be tested, as this would

probably be very similar to the current setup, while also being more compact. To make the CPU more general, a signed multiplier could be implemented with slightly more hardware to multiply signed numbers. More complex implementations of a multiplier could also be investigated to attempt to decrease the number of cycles it takes. Different implementations of the Fibonacci benchmark could also be tested, as the results showed that recursion is very inefficient execution time scales exponentially with the input parameter. An assembler for the invented ISA could also be made to make it easier to convert from assembly to machine code.

In terms of optimisation, pipeline stages would have been useful to limit the effect of propagation delay on the max clock frequency. It would be interesting to investigate how much the maximum clock frequency would improve due to the added intermediate stages lowering the propagation delay and how this would improve the execution times of the benchmarks.

10 Link to Github https://github.com/alexpondaven/CPU

Send login to ap2619@ic.ac.uk for permission to Github.

11 References

[1] Scott Thornton,"What's the difference between Von-Neumann and Harvard architectures" March 8th 2018 [online]MICROCONTROLLERTIPS available at:

<<u>https://www.microcontrollertips.com/difference-between-von-neumann-and-harvard-architectures/</u>>

[3] Clarke, T. Class Lecture, Topic "LEC9.pdf", Department Of Electrical & Electronic Engineering, Imperial College London, UK,2020. Available at: <<u>https://intranet.ee.ic.ac.uk/t.clarke/arch/html16/lect16/lec9.pdf</u>>

[4] ARM QRC 0006E, Thumb 16-bit Instruction Set Qucik Reference Card [online] available at: <<u>http://infocenter.arm.com/help/topic/com.arm.doc.qrc0006e/QRC0006_UAL16.pdf</u>>

[5] Deepak Bordiya, and Lalit Bandil, "*Comparative Analysis Of Multipliers*" International Journal of Engineering Research & Technology. Volume 2 Issue 9, September - 2013, pp. 1437– 1441. [Accessed 8 June 2020]. [Online]. Available at: <<u>https://www.ijert.org/research/comparative-analysis-of-multipliers-serial-and-parallel-with-radix-based-on-booth-algoritham-IJERTV2IS90625.pdf</u>>

[6]Gim P. Hom, Joe Steinmyer, Class Lecture, Topic: "Arithmetic Circuits & Multipliers" 6.111 Introductory Digital Systems Laboratory, Massachusetts Institute of Technology, MA, Fall, 2017.Avaiable at: <<u>http://web.mit.edu/6.111/www/f2017/handouts/L08.pdf</u>>

[7] Wikipedia, Linear Congruential generator.[online] available at: <<u>https://en.wikipedia.org/wiki/Linear_congruential_generator</u>>

[8]Linear Congruenrial Generator I section two,Cornell Department of Mathematics. [online] available at:

<<u>http://pi.math.cornell.edu/~mec/Winter2009/Luo/Linear%20Congruential%20Generator/linear%2</u> <u>Ocongruential%20gen1.html></u>

[9]Torsten Grust, Class Lecture, Topic "*Pipelining*" Database Systems and Modern CPU Architecture, Eberhard Karls University of Tübingen, German, 2009. Available at: <<u>https://db.inf.uni-</u> <u>tuebingen.de/staticfiles/teaching/ss09/dbcpu/dbms-cpu-2.pdf</u>>

[10] ATMEL, ww1.microchip.com. 2020. [online] Available at: <<u>http://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf</u>> [Accessed 1 June 2020].

[11] S3-eu-west-1.amazonaws.com. 2020. *MIPS32*® *Instruction Set Quick Reference*. [online] Available at: <<u>https://s3-eu-west-1.amazonaws.com/downloads-mips/documents/MD00565-2B-MIPS32-QRC-01.01.pdf</u>> [Accessed 28 May 2020].

Glossary, Grefer. 2020. *Risc (Reduced Instruction Set Computer)*. [online] Gartner. Available at: <<u>https://www.gartner.com/en/information-technology/glossary/risc-reduced-instruction-set-computer</u>> [Accessed 19 May 2020].

[13] Clarke, T. Class Lecture, Topic "LEC10.pdf", Department Of Electrical & Electronic Engineering, Imperial College London, UK,2020. Available at: <<u>https://intranet.ee.ic.ac.uk/t.clarke/arch/html16/lect16/lec10.pdf</u>>

Appendix 1: Unpipelined ALU1

1	
3 4 5 6 7 8 9 10 11 12 13	<pre>input [15:0] rddata, // destination register Rd data input [15:0] rsdata, // source register Rs data input [15:0] instr, // instruction word from dout input earrystatus, // source rogister for the dout input earrystatus, // carry from carry flipflop input [3:0] write_next_status, // write_next output from register input [11:0] pc_out, // value in PC - used for BL - storing PC+1 in link register R6 input [15:0] LS_prod, input [15:0] LS_prod,</pre>
14 15 16 17 18 20 21 22 23 24 25 26 27	<pre>output [15:0] dmemin, output [15:0] aluout, // output of ALU - written to Rd output carryout, output [3:0] write_next_out, // write_next_out[3] determines if address is written to register output carryen, // enables writing to carry register output write_next_en, // enables writing to write_next register output write_next_en, // enables writing Rd output [11:0] daddr, // address for data RAM (either Rs +- offset for load/store or R7 for stack instructions) output din_sel, // switches regile din between aluout and ldi/ldr input output write_next_stp, // when a write instruction is after LDR, the PC needs to wait so that LDR finishes writing to registers output bl_sel, output bl_sel, output wl_start,</pre>
28 29 30 31 32	output pc_sload, output pc_cnt_en, output ram_wren
33 34 35 36 37 38 39 40 41	<pre>L); //Intermediate values - need to determine instruction format wire [3:0] op = instr[15:12]; wire sign = instr[11:10]; wire [4:0] offset = instr[10:6]; wire [1:0] field = instr[11:10]; // Used to determine shift type, JMP type and for cin field wire [3:0] cond = instr[9:6]; // Determines number of shifts (can be used for other instructions) wire cwen = instr[9];</pre>
42 44 45 46 47 49 51 23 55 55 55 55 55	<pre>// opcodes wire ldi = !op [3] & !op [2] & !op [1] & !op [0]; wire idr = !op [3] & !op [2] & op [1] & !op [0]; wire str = !op [3] & !op [2] & op [1] & !op [0]; wire mov = !op [3] & !op [2] & op [1] & !op [0]; wire mov = !op [3] & op [2] & !op [1] & !op [0]; wire add = !op [3] & op [2] & !op [1] & !op [0]; wire add = !op [3] & op [2] & !op [1] & !op [0]; wire mul = !op [3] & op [2] & !op [1] & !op [0]; wire mul = !op [3] & !op [2] & !op [1] & !op [0]; wire stmfd = op [3] & !op [2] & !op [1] & !op [0]; wire stmfd = op [3] & !op [2] & !op [1] & !op [0]; wire stmf = op [3] & !op [2] & !op [1] & !op [0];</pre>
56	
57 58 59 60 61 62	<pre>// status += bits: // write_next tells next instruction that the data that is now at dout can be written into the Rs of the previous instruction (for load instructions) // if write_next_flag is already 1, set to 0 (if it is not another ldr), otherwise set to 1 if it is an ldr or ldmfd instruction assign write_next_flag = (write_next_status[3] & (~ldr ldmfd)) ? 0 : (ldr ldmfd);</pre>
63 64 65	// carryen enables writing to carry register - writes when cwen is enabled for add, sub, and mov assign carryen = exec1 & cwen & (add sub mov);
66 67 68	<pre>// write_next_en enables writing to write_next register - needs to update for every instruction during exec1 so that it returns to 0 assign write_next_en = exec1;</pre>
69 70 71	// carryout equal to alucout if not a shift - note the special case of rsdata[0] for LSR or ASR (MOV instruction) assign carryout = (mov & ((-field[1] & field[0]) (field[1] & ~field[0]))) ? rsdata[0] : alucout;
73 74 75	<pre>// Output to write_next register assign write_next_out = {write_next_flag, instr[5:3]};</pre>
76 77 78 79 80	<pre>// write_next_stp stop PC from counting up so instruction can finish before next instruction assign write_next_stp = write_next_status[3] & ~(ldr str jmp ldmfd stp) write_next_status[3] & ldmfd;</pre>
81 82 83 84 85 86	// Definition of wires and regs wire cin; // carry input from instruction wire shiftin; // shift in to MSB for XSR wire alucout; // carry from alusum calculation
8/ 88 89	reg [16:0] alusum; // ALU sum needs extra bit for carry
90 91 92 93 94	<pre>// ALU output Calculations assign alucout = alusum[16]; // carry bit assign adaddr = ldmfd? (alusum[11:0]-1) : alusum[11:0]; // offset address used for addressing data memory or stores R7 value for stack manipulation assign alucout = alusum[15:0]; // 16 bit sum</pre>
95 96 97	assign dmemin = stmfd ? rsdata : rddata; // input to data memory is q2 for stmfd, q1 for str
98 99 100 101	<pre>// wenout active for ldi, mov, add, sub, bl, ldmfd, stmfd (ldr not written during exec1, written in next cycle) assign wenout = exec1 & (ldi mov add sub bl ldmfd stmfd stp & write_next_status[3]);</pre>
102 103 104 105	//cin determined by cin or "type" field in instruction word assign cin = (~field[1] & field[0]) (field[1] & ~field[0] & carrystatus) (field[1] & field[0] & rsdata[15]);

// din_sel should be on if writing back to registers from alu (mov, add, sub, bl, ldmfd, stmfd) // If ldr is writing, din should not be changed yet assign din_sel = (mov | add | sub | bl | ldmfd | stmfd) & ~write_next_stp; // Tells when ldi is on, want it to turn off is ldr is still writing from previous instruction
assign ldi_sel = ldi & ~write_next_stp; //bl_sel tells link_MUX that it is a BL instruction and portladdr needs to be 6 to load PC+1
assign bl_sel = bl & ~write_next_stp; // pop_sel turns on for POP instruction when Rs needs to be written - Rs address at portladdr assign pop_sel = ldmfd & write_next_flag; // enables mul_start for the mult_instruction
assign mul_start = mul; // can be started even though write_next_stp is enabled (ldr is writing in this cycle) // conditional operators - compares Rd and comparator (in cond) - used for if jump should occur wire eg = (rddata == cond); // Rd == comparator wire mi = (rddata < cond); // Rd < comparator // change jmp cond to determine if jmp will occur depending on comparison and jump condition in instruction word wire jmp_cond = jmp & ((-field[] & -field[0]) | (-field[] & field[0] & enj) | (field[1] & field[0] & mi) | (field[1] & field[0] & ~eq & ~mi)); // PC and RAM control signals assign pc_sload = execl & (jmp_cond | bl); assign pc_ort.en = execl & (jmp_cond | stp | bl | write_next_stp); assign ram_wren = execl & (str | stmfd); // determine alusum - need to change opcodes ⊟ always @(*) begin □ case (op) 4'b0001, 4'b0010 : alusum = sign ? (rsdata - {11'b0,offset}) : (rsdata + {11'b0,offset}); // LDR and STR: calculate daddr 4'b0011 : begin case (instr[7:6])
2'b00 : alusum = {ib0,rsdata} + cin; // MOV
2'b01 : alusum = {rsdata,cin}; // LSL
2'b11 : alusum = {rsdata[0], cin, rsdata[15:1]};// Right shift - cin determines LSR or ASR
2'b11 : alusum = instr[0] ? (iib0,MS_prod)+{1 b0,rddata}+cin) : ({1'b0,LS_prod}+{1'b0,rddata}+cin);// Multiply product (leas
default : alusum = {1'b0,rsdata} + cin; // MOV endcase: 4'b0101 : alusum = {1'b0,rddata} + {1'b0,rsdata} + cin;// ADD 4'b0110 : alusum = {1'b0,rddata} + {1'b0,~rsdata} + cin;// SUB 4'b1000 : alusum = {5'b00000,pc_out} + 1; // BL 4'b1001 : alusum = {1'b0,rddata} + 1; // POP (LDMFD)
4'b1010 : alusum = {1'b0,rddata} - 1; // PUSH (STMFD) default : alusum = 0;
endcase:

Appendix 2: Pipelined ALU1 differences



Appendix 3: Meeting notes

Date	Discussion	Alex	Peter	Jason
25/5	 Discussed stack and shared research on AVR and SPARC Went through fibonacci benchmark 			

26/6	 Discussed Jason's assembly code for fib benchmark Went over stacks Discussed number of registers and instruction format Decided it is best to decide this once all benchmarks have been turned into assembly and we know what instructions will be needed 	Look at next benchmarks and see what instructions we may need and convert to assembly	Research multiplication methods	Make assembly more efficient and think of register to use
27/5	 Went over assembly code for LCG and linked list benchmarks Discussed multiplication methods Made a list of instructions using benchmarks 	Think of implementation of instructions (DECA oral as well)	Continue multiplication method research	Reduce number of instructions used in fib
29/5	 Went through Jason's in detail implementation of stack instructions - may need new instructions Made plan of CPU - went through each instruction to determine how it may be implemented - not sure about branch (BL) and LDMB, STMB instructions for fib 	Branch and link hardware implementation	Research multiply methods	LDMB (POP) implementation
31/5	 Discussed LDMFD instruction for loading and increasing stack pointer - this could be done with a bit in instruction word specifying if stack pointer should be changed Discussed instruction word format Got stuck on trying to make LDR one cycle - not sure if this is possible 	Look at different instruction formats (e.g. Thumb) and determine instruction formats	Force block and multiplication in ALU	LDR implementation in different ISAs
5/6	 Discussed multiplication - number of cycles Instruction or subroutine? 	Testing LDR	Parallel multiplier research	ISA table and Branch logic

6/6	 Discussed parallel multiplier ideas Shift/add Parallel - only helps with multiple consecutive multiplies How will we store the 2 registers afterwards 2 registers only for multiply result 			
7/6	 Github merging problems Peter made progress on multiply report Discussed how we could split up testing work later after github is fixed 	Fix github and finish testing all instructions	Implement synchronous multiplier	Report structure and revise hardware



Alex Pondaven added Think of how to implement LDI (12 bits in instruction word but also address the register needed) - LDI may only work for one register - extra cycles to TO DO 2 lup at 12.42 Alex Pondaven archived Delete test files in github Jun at 22:52 Alex Pondaven archived Jason: Stack Register 1 Jun at 22:52 Alex Pondaven added Delete test files in github to TO DO Jun at 19:22 YZ Yuliang Zhu added Jason: Stack Register to For next meeting 31 May at 10:24 Alex Pondaven archived Stacks/caches 30 May at 12:39 Alex Pondaven archived Make layout of CPU - rough sketch 30 May at 12:39 Alex Pondaven archived Multiply Methods, Harvard (Peter) 30 May at 12:39 Alex Pondaven archived How many registers to use? 30 May at 12:39 Alex Pondaven archived Instruction format) May at 12:39 Alex Pondaven added Remov eq and mi from status register - not needed to TO DO Jun at 15:31 Alex Pondaven added Make table to summarise instruction formats to TO DO Alex Pondaven added Update input to PC from g2 to TO DO Jun at 10:51 Alex Pondaven added Status register needs to store the comparison between Rd and Rs for JMP instructions to TO DO Alex Pondaven added Read through stack lecture to TO DO Jun at 19.14 Alex Pondaven archived ALU 3 Jun at 18:04 Alex Pondaven added <u>ALU</u> to TO DO Jun at 18:04 A Alex Pondaven moved Do we need to reference ARMish design as we took inspiration from there? from TO DO to Questions 3 Jun at 18:04 Alex Pondaven moved We basically have 2 decoders now in ALU and decoder outside of this block. Could SM be put inside of regfile alu and then make all control signals come from this block? Would be more clear

from TO DO to Questions

Jun at 18:04

MUX1? 3 Jun at 11:34 Alex Pondaven archived Make a decoder table to determine logic 3 Jun at 11:34

A lex Pondaven archived <u>Separate stack</u> pointer 3 Jun at 11:34

Alex Pondaven archived Do we need

Alex Pondaven archived <u>Think about SM</u> 3 Jun at 11:34

Alex Pondaven archived <u>Find decoder</u> signals 3 Jun at 11:34

A Alex Pondaven added Do we need MUX1? to TO DO 2 Jun at 13:09

Alex Pondaven added <u>Make a decoder</u> <u>table to determine logic</u> to TO DO 2 Jun at 13:04

Alex Pondaven added <u>Find decoder signals</u> to TO DO 2 Jun at 13:04

A Alex Pondaven added <u>Think about SM</u> to TO DO 2 Jun at 13:03

Alex Pondaven added <u>Separate stack</u> pointer to TO DO 2 Jun at 13:03

Alex Pondaven added <u>Change instr. mem</u> to RAM to TO DO

Alex Pondaven archived Update input to <u>PC from q2</u> 8 Jun at 15:42

Alex Pondaven added Forgot a bit in MOV code - need to update this to TO DO 8 Jun at 11:13

A lex Pondaven added Write tests for all instructions to TO DO 8 lun at 11:03

A Alex Pondaven archived <u>Do we need to</u> <u>reference ARMish design as we took</u> <u>inspiration from there?</u> 7 Jun at 17:01

A Alex Pondaven archived <u>Is there any way</u> to do LDI for all registers? 7 Jun at 17:01

A Alex Pondaven archived <u>Make table to</u> summarise instruction formats (include status register) 7. Jun at 17:01

Alex Pondaven archived <u>Remov eq and mi</u> from status register - not needed 7 Jun at 17:01

A Alex Pondaven archived <u>Status register</u> needs to store the comparison between Rd and Rs for JMP instructions 6 Jun at 10:32

Figure 1

implemented with MOV instruction from TO DO to Questions 3 Jun at 18:03 Alex Pondaven moved <u>Is there any way to</u> do LDI for all registers? from TO DO to Questions 3 Jun at 18:03 Alex Pondaven added Questions to this board 3 Jun at 18:03 Alex Pondaven added Do we need to reference ARMish design as we took inspiration from there? to TO DO 3 Jun at 18:02 Alex Pondaven added We basically have 2 decoders now in ALU and decoder outside of this block. Could SM be put inside of regfile alu and then make all control signals come from this block? Would be more clear to TO DO 3 Jun at 17:07 Alex Pondaven added What should the last cin field be? CMSB (as in ARMish) is not very useful now since shifts are being implemented with MOV instruction to TO DO 3 Jun at 16:17 Alex Pondaven added Is there any way to do LDI for all registers? to TO DO 3 Jun at 15:57 Alex Pondaven added Implement multiplication block , adding block, adding output register, to TO DO Alex Pondaven added Test fibonnacci to TO DO 10 Jun at 11:27 Alex Pondaven added Add LDMFD and STMFD to TO DO 10 Jun at 11:27 Alex Pondaven archived We basically have 2 decoders now in ALU and decoder outside of this block. Could SM be put inside of regfile alu and then make all control signals come from this block? Would be more clear 9 Jun at 20:45

Alex Pondaven moved What should the

very useful now since shifts are being

last cin field be? CMSB (as in ARMish) is not

A Alex Pondaven archived <u>What should the</u> last cin field be? CMSB (as in ARMish) is not very useful now since shifts are being <u>implemented with MOV instruction</u> 9 Jun at 20:45

A Alex Pondaven archived Forgot a bit in MOV code - need to update this 9 Jun at 20:44

Alex Pondaven archived <u>Write tests for all</u> instructions 9 Jun at 20:44

A Alex Pondaven added <u>Change STP opcode</u> to A rather than <u>B</u>, as stack was combined, so A is empty now to TO DO 9 Jun at 20:44



Figure 2

Figure 10

Appendix 5

```
module multwithRadix4(prod,finish,A,B,start,clk);
    input [15:0] A, B;
input start, clk;
output prod;
output finish;
                           count;
finish = !count;
    reg [4:0]
wire
    reg [32:0]
wire [31:0]
                        product;
prod = product[31:0];
    reg [17:0]
                           tmp;
     initial count = 0;
                           wire [17:0]
wire [17:0]
wire [17:0]
    always @( posedge clk )
        if( finish && start ) begin
            count = 8;
product = { 16'd0, B };
        end else if( count ) begin
            case ( {product[1:0] } )
2'b00: tmp = {2'b0, product[31:16] };
2'b01: tmp = {2'b0, product[31:16] }
2'b10: tmp = {2'b0, product[31:16] }
2'b11: tmp = {2'b0, product[31:16] }

             endcase
            product = { tmp, product[15:2] };
count = count - 1;
        end
endmodule
```

```
module multwithRadix8(prod,finish,A,B,start,clk);
                input [15:0]
                                                                   Α, Β;
                input
                                                                   start, clk;
                                                                   prod;
               output
              output
                                                                    finish;
                                                                   count;
              reg [2:0]
              wire
                                                                   finish = !count;
              reg [32:0]
                                                                   product;
              wire [31:0]
                                                                   prod = product[31:0];
              reg [19:0]
                                                                   tmp;
              initial count = 0;
             wire [19:0]
wire [19:0]
wire [19:0]
wire [19:0]
wire [19:0]
wire [19:0]
                                                                    A_X_1 = \{1'b0, A\};

A_X_2 = \{A, 1'b0\};

A_X_3 = A_X_2 + A
                                                                                                                              Á
                                                                                                                                      X 1:
                                                                         X_4 = \{A, 2'b0\};
X_5 = A_X_4 + A_4
                                                                          X_6
                                                                                       = A
                                                                                                                4
                                                                                                                       +
                                  19:0
19:0
               wire
                                                                          X_7 = A
                                                                                                                4
                                                                                                                       +
                                                                          X_8 = \{A, 3\}
              wire
                                                                                                                   b0};
                                                                                                                8
                                  19:0
19:0
               wire
                                                                          X_9 = A
                                                                                                                      +
                                                                          X_{10} = A_{X_8}
               wire
                                                                                                                           +
                                  19 0
19 0
19 0
19 0
               wire
                                                                          X_{11} = A_X_8 +
                                                                                                                                  A
                                                                                                                                                  3
              wire
                                                                    A_X_{12} = A_X_8 + A_X_12 = A_X_8 + A_X_12 = A
                                                                                                                                                  4
                                                                          X_{13} = A_X_8 + A_1
              wire
                                                                                                                                                  5
                                                                          X_{14} = A_{X_8}
              wire
                                                                                                                          +
                                                                                                                                  A
                                                                                                                                                  6
              wire [19:0]
                                                                    A_X_{15} = A_X_{8}
              always @( posedge clk )
Ξ
                      if( finish && start ) begin
                                 count
                                                                      = 4:
                                 product = { 16'd0, B };
                      end else if( count ) begin
                                 case ( {product[3:0]} )
                                           +
                                                                                                                                                                                                           +
                                                                                                                                                                                                                                      3
                                                                                                                                                                                                           +
                                                                                                                                                                                                           +
                                                                                                                                                                                                                                      5
                                                                                                                                                                                                           +
                                                                                                                                                                                                                                     6
                                                                                                                                                                                                           +
                                                                                                                                                                                                           +
                                                                                                                                                                                                                                     8
                                                                                                                                                                                                           +
                                                                                                                                                                                                                                     9
                                                                                                                                                                                                           +
                                                                                                                                                                                                                                     10
                                                                                                                                                                                                           +
                                                                                                                                                                                                                                     11:
                                                                                                                                                                                                           +
                                                                                                                                                                                                                    A
                                                                                                                                                                                                                                     12
                                                                                                                                                                                                                            X 13:
                                                                                                                                                                                                           +
                                                                                                                                                                                                                                    14;
                                                                                                                                                                                                            +
                                                                                                                                                                                                                                   15:
                                                                                                                                                                                                           +
                                    endcase
                                    product = { tmp, product[15:4] };
                                                                             = count -1;
                                    count
                        end
    endmodule
```

Figure 11

Flow Summary		Fmax	Restricted Fmax	Clock Name	Note
Flow Settings	1	277.39 MHz	250.0 MHz	clk	limit due to minimum period restriction (max VO toggle rate)
📰 Flow Non-Default Global Settir	g				
Flow Elapsed Time					
Flow OS Summary					
Flow Log					
Analysis & Synthesis					
> 📙 Fitter					
> Assembler					
🗙 📂 TimeQuest Timing Analyzer					
Summary					
Parallel Compilation					
SDC File List					
Clocks					
✓ <a>P> Slow 1200mV 85C Model					
📻 Fmax Summary					
Timing Closure Recon	m				
Setup Summary					

able of Contents 📮 🗗	Slow	1200mV 8	35C Model Setup: 'clk'						
Flow Summary		Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
Flow Settings	1	0.395	multwithRadix4:inst[product[17]	multwithRadix4:inst[product[22]	clk	clk	4.000	-0.064	3.556
Elow Non-Default Global Set	2	0.425	multwithRadix4:inst[product[18]	multwithRadix4:inst[product[22]	clk	clk	4.000	-0.064	3.526
Elow Elanged Time	3	0.448	multwithRadix4:inst[product[23]	multwithRadix4:inst[product[25]	clk	clk	4.000	-0.063	3.504
	4	0.456	multwithRadix4:inst[product[23]	multwithRadix4:inst[product[23]	clk	clk	4.000	-0.063	3.496
Flow OS Summary	5	0.483	multwithRadix4:inst[product[17]	multwithRadix4:inst[product[18]	clk	clk	4.000	-0.064	3.468
E Flow Log	6	0.498	multwithRadix4:inst[product[19]	multwithRadix4:inst[product[21]	clk	clk	4.000	-0.063	3.454
Analysis & Synthesis	7	0.513	multwithRadix4:inst[product[18]	multwithRadix4:inst[product[18]	clk	clk	4.000	-0.064	3.438
Fitter	8	0.514	multwithRadix4:inst[product[16]	multwithRadix4:inst product[21]	clk	clk	4.000	-0.064	3.437
Assembler	9	0.516	multwithRadix4:inst[product[17]	multwithRadix4:inst product[21]	clk	clk	4.000	-0.064	3.435
🖌 📂 TimeQuest Timing Analyzer	10	0.551	multwithRadix4:inst[product[18]	multwithRadix4:inst[product[25]	clk	clk	4.000	-0.067	3.397
Summary	11	0.601	multwithRadix4:inst[product[20]	multwithRadix4:inst[product[21]	clk	clk	4.000	-0.063	3.351
Parallel Compilation	12	0.612	multwithRadix4:inst[product[16]	multwithRadix4:inst[product[17]	clk	clk	4.000	-0.064	3.339
SDC File List	13	0.614	multwithRadix4:inst[product[17]	multwithRadix4:inst[product[17]	clk	clk	4.000	-0.064	3.337
	14	0.619	multwithRadix4:inst[product[21]	multwithRadix4:inst[product[21]	clk	clk	4.000	-0.063	3.333
CIOCKS	15	0.630	multwithRadix4:inst[product[18]	multwithRadix4:inst[product[21]	clk	clk	4.000	-0.064	3.321
V / Slow 1200mV 85C Mode	16	0.635	multwithRadix4:inst[product[17]	multwithRadix4:inst[product[25]	clk	clk	4.000	-0.067	3.313
Fmax Summary	17	0.640	multwithRadix4:inst[product[22]	multwithRadix4:inst[product[22]	clk	clk	4.000	-0.064	3.311
Timing Closure Recc	18	0.640	multwithRadix4:inst[product[26]	multwithRadix4:inst[product[25]	clk	clk	4.000	-0.429	2.946
Setup Summary	19	0.642	multwithRadix4:inst[product[17]	multwithRadix4:inst product[20]	clk	clk	4.000	-0.064	3.309
Hold Summary	20	0.647	multwithRadix4:inst[product[23]	multwithRadix4:inst[product[24]	clk	clk	4.000	-0.061	3.307
Recovery Summary	21	0.653	multwithRadix4:inst[product[17]	multwithRadix4:inst[product[26]	clk	clk	4.000	0.284	3.646
E Removal Summary	22	0.661	multwithRadix4:inst[product[19]	multwithRadix4:inst[product[25]	clk	clk	4.000	-0.066	3.288
Minimum Dulan Width	23	0.665	multwithRadix4:inst[product[16]	multwithRadix4:inst[product[25]	clk	clk	4.000	-0.067	3.283
	24	0.668	multwithRadix4:inst[product[19]	multwithRadix4:inst product[22]	clk	clk	4.000	-0.063	3.284
Vorst-Case Timing I	25	0.669	multwithRadix4:inst[product[19]	multwithRadix4:inst product[23]	clk	clk	4.000	-0.066	3.280
Setup: 'clk'	26	0.672	multwithRadix4:inst[product[18]	multwithRadix4:inst product[20]	clk	clk	4.000	-0.064	3.279
Hold: 'clk'	27	0.672	multwithRadix4:inst[product[17]	multwithRadix4:inst[product[24]	clk	clk	4.000	-0.065	3.278
Metastability Summa	28	0.684	multwithRadix4:inst[product[17]	multwithRadix4:inst product[23]	clk	clk	4.000	-0.067	3.264
> Slow 1200mV 0C Model	29	0.685	multwithRadix4:inst[product[16]	multwithRadix4:inst[product[23]	clk	clk	4.000	-0.067	3.263
> Fast 1200mV 0C Model	30	0.688	multwithRadix4:inst[product[17]	multwithRadix4:inst product[30]	clk	clk	4.000	0.284	3.611
Multicorner Timing Analy	31	0.688	multwithRadix4:inst[product[23]	multwithRadix4:inst product[22]	clk	clk	4.000	-0.060	3.267
Advanced I/O Timing	32	0.699	multwithRadix4:inst[product[18]	multwithRadix4:inst[product[26]	clk	clk	4.000	0.284	3.600
Clock Transform	33	0.705	multwithRadix4:inst[product[20]	multwithRadix4:inst[product[22]	clk	clk	4.000	-0.063	3.247
	34	0.706	multwithRadix4:inst[product[18]	multwithRadix4:inst[product[24]	clk	clk	4.000	-0.065	3.244
Report ICCS	35	0.718	multwithRadix4:inst[product[18]	multwithRadix4:inst[product[30]	clk	clk	4.000	0.284	3.581
Report RSKM	36	0.721	multwithRadix4:inst[product[23]	multwithRadix4:inst product[21]	clk	clk	4.000	-0.060	3.234
Unconstrained Paths	37	0.722	multwithRadix4:inst[product[18]	multwithRadix4:inst product[27]	clk	clk	4.000	0.284	3.577
 Messages 	38	0.728	multwithRadix4:inst[product[18]	multwithRadix4:inst[product[17]	clk	clk	4.000	-0.064	3.223
EDA Netlist Writer	39	0.752	multwithRadix4:inst[product[20]	multwithRadix4:inst[product[25]	clk	clk	4.000	-0.066	3.197
×	40	0.756	multwithRadix4:inst[product[19]	multwithRadix4:inst[product[18]	clk	clk	4.000	-0.063	3.196
· >	44	0 757	multurithDadic (risoflared upf) 171	multurithDadie/Lipplineduct/161	alle	all	4 000	0.064	2 404

Flow Sur	mmary ^		Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
Flow Set	ttings	1	0.357	multwithRadix4:inst[count[4]	multwithRadix4:inst[count[4]	clk	clk	0.000	0.063	0.577
Flow Nor	n-Default Global Set	2	0.498	multwithRadix4:inst[product[12]	multwithRadix4:inst[product[10]	clk	clk	0.000	0.063	0.718
Flow Flag	osed Time	3	0.499	multwithRadix4:inst[product[4]	multwithRadix4:inst[product[2]	clk	clk	0.000	0.063	0.719
Elow OS	Summary	4	0.525	multwithRadix4:inst[count[4]	multwithRadix4:inst[product[27]	clk	clk	0.000	0.429	1.111
Elevel es	Summary	5	0.525	multwithRadix4:inst[count[4]	multwithRadix4:inst[product[26]	clk	clk	0.000	0.429	1.111
		6	0.558	multwithRadix4:inst[count[3]	multwithRadix4:inst[count[3]	clk	clk	0.000	0.063	0.778
> Analysis	& Synthesis	7	0.572	multwithRadix4:inst[product[3]	multwithRadix4:inst product[1]	clk	clk	0.000	0.063	0.792
> Fitter		8	0.572	multwithRadix4:inst[product[11]	multwithRadix4:inst[product[9]	clk	clk	0.000	0.063	0.792
> 🧧 Assemble	er	9	0.573	multwithRadix4:inst[product[9]	multwithRadix4:inst[product[7]	clk	clk	0.000	0.063	0.793
🗸 📂 TimeQues	st Timing Analyzer	10	0.574	multwithRadix4:inst[product[7]	multwithRadix4:inst[product[5]	clk	clk	0.000	0.063	0.794
Sumr	mary	11	0.574	multwithRadix4:inst[product[2]	multwithRadix4:inst[product[0]	clk	clk	0.000	0.063	0.794
Para'	llel Compilation	12	0.575	multwithRadix4:inst[product[14]	multwithRadix4:inst[product[12]	clk	clk	0.000	0.063	0.795
SDC	File List	13	0.584	multwithRadix4:inst[product[0]	multwithRadix4:inst[product[14]	clk	clk	0.000	0.063	0.804
Clock	ke	14	0.586	multwithRadix4:inst[product[10]	multwithRadix4:inst[product[8]	clk	clk	0.000	0.401	1.144
	1000-1/050 H-1	15	0.625	multwithRadix4:inst[product[13]	multwithRadix4:inst[product[11]	clk	clk	0.000	0.063	0.845
V / Slow	7 1200mV 85C Mode	16	0.630	multwithRadix4:inst[count[4]	multwithRadix4:inst[count[0]	clk	clk	0.000	0.063	0.850
	Fmax Summary	17	0.631	multwithRadix4:inst[count[4]	multwithRadix4:inst[count[2]	clk	clk	0.000	0.063	0.851
E 1	Timing Closure Recc	18	0.632	multwithRadix4:inst[count[4]	multwithRadix4:inst[product[25]	clk	clk	0.000	0.063	0.852
 5	Setup Summary	19	0.632	multwithRadix4:inst[count[4]	multwithRadix4:inst[product[23]	clk	clk	0.000	0.063	0.852
	Hold Summary	20	0.649	multwithRadix4:inst[count[4]	multwithRadix4:inst[count[1]	clk	clk	0.000	0.063	0.869
E 6	Recovery Summary	21	0.691	multwithRadix4:inst[product[5]	multwithRadix4:inst[product[3]	clk	clk	0.000	0.063	0.911
E F	Removal Summary	22	0.700	multwithRadix4:inst[product[6]	multwithRadix4:inst[product[4]	clk	clk	0.000	0.063	0.920
	Minimum Pulse Widtt	23	0.806	multwithRadix4:inst[count[4]	multwithRadix4:inst[product[31]	clk	clk	0.000	0.429	1.392
	Manak Casa Timina I	24	0.807	multwithRadix4:inst[count[4]	multwithRadix4:inst[product[30]	clk	clk	0.000	0.429	1.393
· · · ·	worst-case mining i	25	0.816	multwithRadix4:inst[product[30]	multwithRadix4:inst[product[28]	clk	clk	0.000	0.077	1.050
	Setup: 'clk'	26	0.823	multwithRadix4:inst[product[15]	multwithRadix4:inst[product[13]	clk	clk	0.000	-0.262	0.718
	Hold: 'clk'	27	0.824	multwithRadix4:inst[product[8]	multwithRadix4:inst[product[6]	clk	clk	0.000	-0.262	0.719
E 1	Metastability Summa	28	0.841	multwithRadix4:inst[count[4]	multwithRadix4:inst product[15]	clk	clk	0.000	0.397	1.395
> 📙 Slow	/ 1200mV 0C Model	29	0.847	multwithRadix4:inst[count[4]	multwithRadix4:inst[product[24]	clk	clk	0.000	0.065	1.069
> 📕 Fast	1200mV 0C Model	30	0.848	multwithRadix4:inst[count[2]	multwithRadix4:inst[count[3]	clk	clk	0.000	0.063	1.068
📅 Multic	corner Timing Analy	31	0.849	multwithRadix4:inst[count[1]	multwithRadix4:inst[count[0]	clk	clk	0.000	0.063	1.069
> Adv	anced VO Timing	32	0.851	multwithRadix4:inst[count[1]	multwithRadix4:inst[count[2]	clk	clk	0.000	0.063	1.071
	k Transfere	33	0.854	multwithRadix4:inst[count[1]	multwithRadix4:inst[product[23]	clk	clk	0.000	0.063	1.074
	TOCO	34	0.856	multwithRadix4:inst[count[1]	multwithRadix4:inst product[25]	clk	clk	0.000	0.063	1.076
неро	ULI ICCS	35	0.870	multwithRadix4:inst[product[1]	multwithRadix4:inst[product[28]	clk	clk	0.000	0.432	1.459
Repo	OTT RSKM	36	0.915	multwithRadix4:inst[count[1]	multwithRadix4:inst[product[26]	clk	clk	0.000	0.429	1.501
> 🧧 Unco	onstrained Paths	37	0.916	multwithRadix4:inst[count[1]	multwithRadix4:inst[product[27]	clk	clk	0.000	0.429	1.502
🕕 Mess	sages	38	0.929	multwithRadix4:inst[count[1]	multwithRadix4:inst[count[4]	clk	clk	0.000	0.063	1.149
> 📙 EDA Netli	list Writer	39	0.941	multwithRadix4:inst[count[1]	multwithRadix4:inst[count[3]	clk	clk	0.000	0.063	1.161
· · · · · · · · ·	×	40	0.950	multwithRadix4:inst[count[1]	multwithRadix4:inst[count[1]	clk	clk	0.000	0.063	1.170

Flow Summary Flow Stings Flow Stings How Flow Power Analyzer Status Successful - Thu Jun 11 10.44.32 2020 Ouarus Prime Version 16.0.0 Build 211 04/27/2016 SJ. Lat Edition Flow Else Time CPU Flow Stepsed Time CPU Flow Stepsed Time Top-level Entity Name Flow Stepsed Time Top-level Entity Name Flow Stepsed Time Power Models Flow Stepsed Time Fow Logs Analysis & Synthesis For Enall Stepsed Time Stepsed Time Total Thermal Power Dissipation Assember Core Static Thermal Power Dissipation 2.2.4 mW Core Static Thermal Power Dissipation 4.2.8 mW W TimeQuest Timeg Analyzer Fow Filsy Power Analyzer Flow Istings Indeterminate Toggle Rates Core Static Thermal Power Dissipation Flow Thermal Power Dissipation Confidence Low: user provided insufficient toggle rate data Stepses Flow Resages Signal Actrides Flow Resages Flow Suppressed Messages Flow Suppressed Messages	Tab	le of	Contents 🛛 📮 🗗	PowerPlay Power Analyzer Summary	
 Fire W Settings Fire W Settings Fire W Non-Default Global Setting W Non-Default Global Setting Fire M Dever Dissipation <	-	Ħ	Flow Summary	PowerPlay Power Analyzer Status	Successful - Thu Jun 11 10:44:32 2020
Image: Section Sectin Sectin Section Section Section Section Section Section Section			Flow Settings	Quartus Prime Version	16.0.0 Build 211 04/27/2016 SJ Lite Edition
In own Non-Control Lobust Journal Journal Top-level Entity Name mutwithRadixATesting If prov Elapsed Time Family Cyclone N E Image: Plow Log Power Nodels Final Image: Plow Log Core Ophanic Thermal Power Dissipation 42.88 mW Image: Plow Plower Analyzer Power Flags Power Analyzer Power Flags Power Analyzer Image: Plow Power Analyzer Power Flags Power Analyzer Low Lober Tossipation Image: Plow Power Dissipation Image: Plower Dimage: Plower Dissipation Image: Plower Dissipation Image:			Flow Non Default Global Setting	Revision Name	CPU
First Vox Euglaeu Innie Family Cyclone M E First Vox Summary Device EP4CE6F17C6 Power Models Final Image: Constraint of the power Dissipation 102.71 mW Assembler Core Static Thermal Power Dissipation 42.88 mW Image: Constraint formal Power Dissipation 52.41 mW Core Static Thermal Power Dissipation 52.41 mW Power Kales VO Thermal Power Dissipation 52.41 mW Power Kales Power Estimation Confidence Low: user provided insufficient toggle rate data Image: Stating Stati			Flow Flopped Time	Top-level Entity Name	multwithRadix4Testing
 Pow OS Summary Device EPACEF17C6 Power Models Fila Anaysis & Synthesis Total Thermal Power Dissipation Core Dynamic Thermal Power Dissipation Summary Sommary Settings Thermal Power Dissipation Operating Conditions Used Thermal Power Dissipation Thermal Power Dissipation Thermal Power Dissipation Summary Settings Core Dynamic Thermal Power Dissipation Fow Kessages Fow Kessages Fow Suppressed Messages 			Flow clapsed fille	Family	Cyclone IV E
 Power Modeis Analysis & Synthesis Total Thermal Power Dissipation Total Thermal Power Dissipation Total Thermal Power Dissipation Assembler Assembler The Cuest Timing Analyzer EDA Netist Writer Power flag Power Analyzer Settings Settings Indeterminate Toggle Rates Ore Dynamic Thermal Power Dissipation Thermal Power Dissipation Thermal Power Dissipation Settings Settings Thermal Power Dissipation Thermal Power Dissipation Settings Settings Core Other Dissipation Settings Core Other Dissipation Core Other Dissipation Core Other Dissipation Signal Activities Messages Flow Suppressed Messages Flow Suppressed Messages 			Flow US Summary	Device	EP4CE6F17C6
> Analysis & Synthesis Total Thermal Power Dissipation 102.71 mW > Fiter Core Opamic Therma Power Dissipation 7.42 nW > Total Thermal Power Dissipation 42.88 mW W Thermal Power Dissipation 52.41 mW > EDA Netlist Wher Vor Thermal Power Dissipation 52.41 mW > PowerPlay Power Analyzer Fistimation Confidence Low: user provided insufficient toggle rate data Image: Signal Activities Fistimation Confidence Low: user provided insufficient toggle rate data Image: Confidence Method Dissipation Fistimation Confidence Low: user provided insufficient toggle rate data Image: Confidence Method Dissipation Fistimation Confidence Low: user provided insufficient toggle rate data Image: Confidence Method Dissipation Fistimation Confidence Low: user provided insufficient toggle rate data Image: Confidence Method Dissipation Fistimation Confidence Method Dissipation Fistimation Confidence Method Dissipation Image: Confidence Method Details Fistimation Confidence Method Details Fistimation Confidence Method Details Image: Confidence Method Details Fistimation Confidence Method Details Fistimation Confidence Method Details Image: Plow Suppressed Messages Flow Suppressed Messages Flow Sup			Flow Log	Power Models	Final
 Fitter Core Dynamic Thermal Power Dissipation 7.42 mW Assembler TomoQuest Timing Analyzer EDA Netlist Writer Power Play Power Analyzer Summary Settings Operating Conditions Used Thermal Power Dissipation 1 Settings Operating Conditions Used Core Dynamic Thermal Power Dissipation 1 Thermal Power Dissipation 2 Flow Suppr	>		Analysis & Synthesis	Total Thermal Power Dissipation	102.71 mW
 Assembler Che Static Infermal Power Dissipation EDA Netils Writer ChewerBay Power Analyzer Summary Sattings Indeterminate Toggle Rates Operating Conditions Used Thermal Power Dissipation Core Dynamic Thermal Power Signal Activities Messages Flow Suppressed Messages 	>		Fitter	Core Dynamic Thermal Power Dissipation	7.42 mW
 TrmcQuest Trming Analyzer EDA Netlist Wrter Power Pay Power Analyzer Summary Settings Indeterminate Toggle Rates Operating Conditions Used Thermal Power Dissipation 1 Thermal Power Dissipation 1 Core Dynamic Thermal Power Confidence Metric Details Signal Activities Messages Flow Messages Flow Suppressed Messages 	>		Assembler	Core Static Thermal Power Dissipation	42.88 mW
 EDA Netlist Writer PowerPlay Power Analyzer Summary Settings Indeterminate Toggle Rates Operating Conditions Used Thermal Power Dissipation Thermal Power Dissipation Thermal Power Dissipation Core Dynamic Thermal Power Current Drawn from Votag Signal Activities Messages Flow Messages Flow Suppressed Messages 	>		TimeQuest Timing Analyzer	VO Inermal Power Dissipation	52.41 mW
 PowerPlay Power Analyzer Summary Settings Indeterminate Toggle Rates Operating Conditions Used Thermal Power Dissipation 1 Thermal Power Dissipation 1 Core Dynamic Thermal Pow Current Drawn from Votage Signal Activities Messages Flow Messages Flow Suppressed Messages 	>		EDA Netlist Writer	Power Estimation Confidence	Low, user provided insufficient toggle rate data
 Summary Settings Indeterminate Toggle Rates Operating Conditions Used Thermal Power Dissipation 1 Thermal Power Dissipation 1 Thermal Power Dissipation 1 Core Dynamic Thermal Power Corret Drawn from Voltage Confidence Metric Details Signal Activities Messages Flow Messages Flow Suppressed Messages 	~	Þ	PowerPlay Power Analyzer		
 Settings Indeterminate Toggle Rates Operating Conditions Used Thermal Power Dissipation 1 Thermal Power Dissipation 1 Thermal Power Dissipation 1 Thermal Power Dissipation 1 Core Dynamic Thermal Power Confidence Metric Details Signal Activities Messages Flow Messages Flow Suppressed Messages 			Summary		
 Indeterminate Toggle Rates Operating Conditions Used Thermal Power Dissipation 1 Thermal Power Dissipation 1 Thermal Power Dissipation 1 Core Dynamic Thermal Pow Corred Drawn from Voltage Confidence Metric Details Signal Activities Messages Flow Messages Flow Suppressed Messages 			E Settings		
 Operating Conditions Used Thermal Power Dissipation 1 Thermal Power Dissipation 1 Thermal Power Dissipation 1 Core Dynamic Thermal Power Current Drawn from Voltage Confidence Metric Details Signal Activities Messages Flow Messages Flow Suppressed Messages 			Indeterminate Toggle Rates		
 Thermal Power Dissipation I Thermal Power Dissipation I Thermal Power Dissipation I Core Dynamic Thermal Pow Core Dynamic Thermal Pow Confidence Metric Details Signal Activities Messages Flow Messages Flow Suppressed Messages 			Operating Conditions Used		
 Thermal Power Dissipation I Thermal Power Dissipation I Core Dynamic Thermal Pow Current Drawn from Votage Confidence Metric Details Signal Activities Messages Flow Messages Flow Suppressed Messages 			Thermal Power Dissipation I		
 Thermal Power Dissipation 1 Core Dynamic Thermal Pow Current Drawn from Voltage Confidence Metric Details Signal Activities Messages Flow Messages Flow Suppressed Messages 			Thermal Power Dissipation I		
 Core Dynamic Thermal Pow Current Drawn from Voltage Confidence Metric Details Signal Activities Messages Flow Messages Flow Suppressed Messages 			Thermal Power Dissipation I		
 Current Drawn from Voltage Confidence Metric Details Signal Activities Messages Flow Messages Flow Suppressed Messages 			Core Dynamic Thermal Pow		
 Confidence Metric Details Signal Activities Messages Flow Messages Flow Suppressed Messages 		>	Current Drawn from Voltage		
 Signal Activities Messages Flow Messages Flow Suppressed Messages 			Confidence Metric Details		
 Messages Flow Messages Flow Suppressed Messages 			Signal Activities		
 Flow Messages Flow Suppressed Messages 			Messages		
Flow Suppressed Messages		6	Flow Messages		
		ŏ	Flow Suppressed Messages		
		-			

Figure analysis Radix 4

Table of Contents	8	Slow 1200	mV 85C Model Fmax Summ	ary						
	-1		Fmax Restricted Fmax	Clock Name		Note				
Flow Summary		1 200	03 MHz 250 0 MHz	alk	limit due to minimum period r	estriction (may I/O)	to cala rate)			
Flow Settings		1 300.	93 MHZ 250.0 MHZ	CIK	limit due to minimum period r	estriction (max VO	toggie rate)			
Flow Non-Default Global Se	tting									
Elow Flansed Time										
Flow OS Summary										
Flow Log										
Analysis & Synthesis										
> Fitter										
> Assembler										
V TimeQuest Timing Analyzer										
E Summany										
Summary										
Parallel Compilation										
SDC File List										
Clocks										
Slow 1200mV 85C Mod	el									
Fmax Summary										
Timing Closure Rec	omm									
setup Summary										
Hold Summary										
Recovery Summary	, -									
E Removal Summany										
E Renoval Summary										
Minimum Pulse Widt	h Su									
> Worst-Case Timing	Path									
E Metastability Summa	arv									
Slow 1200mV 0C Mode	'									
Fast 1200mV 0C Model										
Multicorner Timing Anal	/sis									
> Advanced VO Timing										
Clock Transfers										
Report TCCS										
Report PSKM										
Unconstrained Paths										
 Messages 										
> EDA Netlist Writer										
EDA Netlist Writer	ŀ	This nanel	reports FMAX for every clock	n the design regardle	see of the user-specified cloc	k narioda EMAX is	only computed	for nothe where t	he source and des	etii
 EDA Netlist Writer Flow Messages 		This panel i	reports FMAX for every clock	n the design, regardle	ess of the user-specified cloc	k periods. FMAX is FMAX is compute	s only computed in the rising	for paths where t	he source and des	stii
 EDA Netlist Writer Flow Messages Flow Suppressed Message 	s	This panel clocks, incl Altera reco	reports FMAX for every clock uding generated clocks, are ig mmends that you always use	n the design, regardle nored. For paths bety clock constraints and	ess of the user-specified cloc ween a clock and its inversion other slack reports for sign-t	k periods. FMAX is n, FMAX is compute off analysis.	s only computed as if the rising	for paths where t and falling edges	he source and des are scaled along	stii wi
DA Netlist Writer Flow Messages Flow Suppressed Message	s	This panel clocks, incl Altera reco	reports FMAX for every clock luding generated clocks, are ig mmends that you always use	n the design, regardle nored. For paths bety clock constraints and	ess of the user-specified cloo ween a clock and its inversion other slack reports for sign-	k periods. FMAX is n, FMAX is compute ff analysis.	s only computed as if the rising	for paths where t and falling edges	he source and des are scaled along	stii wi
EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow	s Slov	This panel clocks, incl Altera reco v 1200mV 8	reports FMAX for every clock uding generated clocks, are ig immends that you always use ISC Model Setup: 'clk'	n the design, regardle nored. For paths bety clock constraints and	ess of the user-specified cloo ween a clock and its inversion other slack reports for sign-	k periods. FMAX is n, FMAX is compute off analysis.	s only computed d as if the rising	for paths where t and falling edges	he source and des are scaled along	stii wi
EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Summary	s Slov	This panel clocks, incl Altera reco v 1200mV 8 Slack	reports FMAX for every clock uding generated clocks, are ig immends that you always use SC Model Setup: 'clk' From Node	n the design, regardle nored. For paths bety clock constraints and To Noc	ess of the user-specified cloc ween a clock and its inversion other slack reports for sign-r	k periods. FMAX is n, FMAX is compute off analysis.	s only computed ed as if the rising Relationship	for paths where t and falling edges Clock Skew	he source and des are scaled along Data Delay	stii wi
EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Summary Flow Summary Flow Summary	s Slov	This panel i clocks, incl Altera reco v 1200mV 8 Slack 0.677	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'clk' From Node mutwithckinstB copy(1)	n the design, regardle nored. For paths bety clock constraints and To Noc multwithclk:instoroc	ess of the user-specified cloc ween a clock and its inversion other slack reports for sign- te Launch Cloc duct[30] clk	k periods. FMAX is n, FMAX is compute ff analysis.	s only computed ad as if the rising Relationship 4.000	for paths where for paths where for and falling edges	he source and des are scaled along Data Delay 3.275	stii wi
EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Summary Flow Summary Flow Settings	s Slov	This panel i clocks, incl Altera reco v 1200mV 8 Slack 0.677 0.683	reports FMAX for every clock uding generated clocks, are ig immends that you always use ISC Model Setup: 'clk' From Node mutwithcik:inst[B_cooy[1] mutwithcik:inst[B_cooy[1]	n the design, regardle hored. For paths betw clock constraints and To Noc multwithclk.instiproc multwithclk.instiproc	ess of the user-specified cloc ween a clock and its inversion other slack reports for sign te Launch Clock duct[30] clk duct[31] ck	k periods. FMAX is h, FMAX is compute ff analysis. Latch Clock clk clk	Relationship	for paths where t and falling edges Clock Skew -0.063 -0.063	he source and des are scaled along to Data Delay 3.275 3.269	stii wi
EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Summary Flow Summary Flow Settings Flow Non-Default Global Set	s Slov	This panel i clocks, incl Altera reco v 1200mV 8 Slack 0.677 0.683 0.697	reports FMAX for every clock uding generated clocks, are ig immends that you always use ISC Model Setup: 'clk' From Node mutwithck:inst[B_copy[1] mutwithck:inst[B_copy[1] mutwithck:inst[B_copy[1]	n the design, regardle hored. For paths bety clock constraints and To Nor multwithclk-instgroo multwithclk-instgroo	ess of the user-specified cloc ween a clock and its inversion other slack reports for sign- te Launch Cloc duct[30] clk duct[31] clk duct[31] clk	k periods. FMAX is n, FMAX is compute off analysis. Latch Clock clk clk clk	Relationship	for paths where t and falling edges Clock Skew -0.063 -0.063 -0.062	Data Delay 3.275 3.269 3.256	stii wi
EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Summary Flow Settings Flow Non-Default Global Set Flow Elapsed Time	s Slov	This panel i clocks, incl Altera reco v 1200mV 8 Slack 0.677 0.683 0.697 0.744	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'cik' From Node mutwithckinst[B_copy[1] mutwithckinst[B_copy[0] mutwithckinst[B_copy[0] mutwithckinst[B_copy[0]	n the design, regardle nored. For paths bety clock constraints and To Noc multwithclk-instiproc multwithclk-instiproc multwithclk-instiproc	ess of the user-specified cloc ween a clock and its inversion other slack reports for sign- te Launch Cloc duct[30] clk duct[31] clk duct[31] clk	k periods. FMAX is h, FMAX is compute ff analysis. Latch Clock clk clk clk clk clk	Relationship 4.000 4.000 4.000	for paths where t and falling edges Clock Skew -0.063 -0.063 -0.062 -0.063	Data Delay 3.275 3.269 3.208	stii wi
EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Summary Flow Settings Flow Non-Default Global Set Flow Clapsed Time Flow OS Summary	s Slow	This panel i clocks, incl Altera reco v 1200mV 8 Slack 0.677 0.683 0.697 0.744 0.782	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'clk' From Node mutwithckinstB_copy[1] mutwithckinstB_copy[2] mutwithckinstB_copy[2]	n the design, regardle nored. For paths bety clock constraints and To Noc multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo	ess of the user-specified cloc ween a clock and its inversion other slack reports for sign-1 de Launch Cloc duct[30] clk duct[31] clk duct[31] clk duct[31] clk duct[31] clk	k periods. FMAX is n, FMAX is compute ff analysis. Latch Clock clk clk clk clk clk clk clk	Relationship 4.000 4.000 4.000 4.000	Clock Skew -0.063 -0.063 -0.063 -0.062 -0.063	Data Delay 3.275 3.269 3.256 3.208 3.211	stii wi
EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Summary Flow Settings Flow Non-Default Global Set Flow Cloped Time Flow Flow Cloped Time Flow Fl	s Slow	This panel i clocks, incl Altera reco v 1200mV 8 Slack 0.677 0.683 0.697 0.744 0.782	reports FMAX for every clock uding generated clocks, are ig immends that you always use ISC Model Setup; 'clk' From Node mutwithckinstB_copy[1] mutwithckinstB_copy[0] mutwithckinstB_copy[0] mutwithckinstB_copy[0] mutwithckinstB_copy[0]	n the design, regardle nored. For paths bety clock constraints and To Noc mutwithclk:instgroo mutwithclk:instgroo mutwithclk:instgroo mutwithclk:instgroo	ess of the user-specified cloc ween a clock and its inversion other slack reports for sign te Launch Cloc duct[30] clk duct[31] clk duct[31] clk duct[31] clk duct[31] clk duct[30] clk	k periods. FMAX is h, FMAX is compute off analysis. Latch Clock clk clk clk clk clk clk clk	Relationship 4.000 4.000 4.000 4.000 4.000	Clock Skew -0.063 -0.063 -0.062 -0.062 -0.062	Data Delay 3.275 3.269 3.256 3.208 3.171	stii wi
	s Slow	This panel clocks, incl Altera reco v 1200mV 8 Slack 0.677 0.683 0.697 0.744 0.782 0.788	reports FMAX for every clock uding generated clocks, are ig immends that you always use ISC Model Setup; 'clk' From Node mutwithclkinst[B_copy[1] mutwithclkinst[B_copy[0] mutwithclkinst[B_copy[0] mutwithclkinst[B_copy[0] mutwithclkinst[B_copy[0] mutwithclkinst[B_copy[0] mutwithclkinst[B_copy[0] mutwithclkinst[B_copy[0]	n the design, regardle nored. For paths bety clock constraints and To Noc multwithclkinstproc multwithclkinstproc multwithclkinstproc multwithclkinstproc multwithclkinstproc	ess of the user-specified cloc ween a clock and its inversion other slack reports for sign le Launch Cloc duct[30] clk duct[31] clk duct[31] clk duct[31] clk duct[30] clk duct[30] clk	k periods. FMAX is compute , FMAX is compute off analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000	for paths where I and failing edges Clock Skew -0.063 -0.062 -0.062 -0.063 -0.062 -0.063	Data Delay 3.275 3.269 3.256 3.208 3.171 3.164 9.67	stii wi
	s Slov	This panel in clocks, incl clocks, incl Altera reco v 1200mV 8 Slack 0.677 0.683 0.697 0.744 0.782 0.788 0.793 0.703	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'cik' From Node mutwithcik:instB_copy[1] mutwithcik:instB_copy[2] mutwithcik:instB_copy[2] mutwithcik:instB_copy[3] mutwithcik:instB_copy[3] mutwithcik:instB_copy[1] mutwithcik:instB_copy[1]	n the design, regardle nored. For paths bety clock constraints and To Noc mult with clk instgroo mult with clk instgroo	ess of the user-specified cloc ween a clock and its inversion other slack reports for sign- de Launch Cloc duct[30] clk duct[31] clk duct[31] clk duct[31] clk duct[31] clk duct[30] clk duct[30] clk duct[30] clk duct[30] clk	k periods. FMAX is compute ff analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000	Clock Skew -0.063 -0.063 -0.062 -0.063 -0.062 -0.063 -0.063 -0.063 -0.063 -0.063	Data Delay 3.275 3.269 3.269 3.269 3.171 3.164 3.159 3.459	stii wi
	s Slow 1 2 3 4 5 6 7 8	Slack 0.677 0.683 0.677 0.683 0.742 0.782 0.782 0.782 0.782 0.783 0.793 0.794 <th< td=""><td>reports FMAX for every clock uding generated clocks, are ig immends that you always use isc Model Setup: 'clk' From Node mutwithcikinstB_copy[1] mutwithcikinstB_copy[0] mutwithcikinstB_copy[0] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3]</td><td>n the design, regardle nored. For paths bety clock constraints and To Noc multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo</td><td>ess of the user-specified cloc ween a clock and its inversion other slack reports for sign fee Launch Cloc duct[30] clk duct[31] clk duct[31] clk duct[31] clk duct[30] clk duct[30] clk duct[30] clk duct[30] clk</td><td>k periods. FMAX is compute f, FMAX is compute off analysis. Latch Clock clk clk clk clk clk clk clk c</td><td>Relationship 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000</td><td>Tor paths where i and failing edges Clock Skew -0.063 -0.063 -0.062 -0.063 -0.062 -0.063 -0.063 -0.063 -0.063</td><td>Data Delay 3.275 3.266 3.208 3.171 3.169 3.158</td><td>stii wi</td></th<>	reports FMAX for every clock uding generated clocks, are ig immends that you always use isc Model Setup: 'clk' From Node mutwithcikinstB_copy[1] mutwithcikinstB_copy[0] mutwithcikinstB_copy[0] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3]	n the design, regardle nored. For paths bety clock constraints and To Noc multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo	ess of the user-specified cloc ween a clock and its inversion other slack reports for sign fee Launch Cloc duct[30] clk duct[31] clk duct[31] clk duct[31] clk duct[30] clk duct[30] clk duct[30] clk duct[30] clk	k periods. FMAX is compute f, FMAX is compute off analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000	Tor paths where i and failing edges Clock Skew -0.063 -0.063 -0.062 -0.063 -0.062 -0.063 -0.063 -0.063 -0.063	Data Delay 3.275 3.266 3.208 3.171 3.169 3.158	stii wi
	s Siov 1 2 3 4 5 6 7 8 9	Slack 0.677 0.683 0.697 0.782 0.782 0.782 0.783 0.793 0.794	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'clk' From Node mutwithckinst[B_copy[1] mutwithckinst[B_copy[0] mutwithckinst[B_copy[2] mutwithckinst[B_copy[2] mutwithckinst[B_copy[3] mutwithckinst[B_copy[3] mutwithckinst[B_copy[7]	n the design, regardle nored. For paths bety clock constraints and To Noc mutwithclkinstproc mutwithclkinstproc mutwithclkinstproc mutwithclkinstproc mutwithclkinstproc mutwithclkinstproc mutwithclkinstproc	ess of the user-specified cloc ween a clock and its inversion other slack reports for sign te Launch Cloc duct[30] clk duct[31] clk duct[31] clk duct[31] clk duct[30] clk duct[30] clk duct[30] clk duct[31] clk duct[31] clk duct[31] clk	k periods. FMAX is compute ff analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000	Clock Skew -0.063 -0.063 -0.063 -0.062 -0.062 -0.063 -0.062 -0.063 -0.062 -0.063 -0.063 -0.062 -0.063 -0.063 -0.063 -0.062 -0.063 -0.063 -0.063 -0.062 -0.063 -0.063 -0.063 -0.063 -0.062 -0.063 -0.063 -0.063 -0.063 -0.063 -0.063 -0.062 -0.063 -0.063 -0.062 -0.063 -0.063 -0.062 -0.063 -0.062 -0.063 -0.063 -0.062 -0.063 -0.062 -0.063 -0.062 -0.063 -0.062 -0.063 -0.063 -0.062 -0.063 -0.063 -0.062 -0.063 -0	Data Delay 3.275 3.266 3.208 3.171 3.164 3.158 3.156	stii wi
EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Suppressed Message Flow Summary Flow Settings Flow Non-Default Global Set Flow Clapsed Time Flow Cog Analysis & Synthesis Fiter Assembler For TimeQuest Timing Analyzer	s slow	This panel clocks, incl Altera reco v 1200mV 8 0.677 0.744 0.782 0.793 0.794 0.799	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'cik' From Node mutwithcik:instB_copy[1] mutwithcik:instB_copy[2] mutwithcik:instB_copy[2] mutwithcik:instB_copy[3] mutwithcik:instB_copy[3] mutwithcik:instB_copy[1] mutwithcik:instB_copy[1] mutwithcik:instB_copy[1] mutwithcik:instB_copy[1]	n the design, regardle nored. For paths bety clock constraints and To Noc multwithclkinstproc multwithclkinstproc multwithclkinstproc multwithclkinstproc multwithclkinstproc multwithclkinstproc multwithclkinstproc multwithclkinstproc	ess of the user-specified cloc ween a clock and its inversion other slack reports for sign- fe Launch Cloc duct[30] clk duct[31] clk duct[31] clk duct[31] clk duct[31] clk duct[30] clk duct[30] clk duct[30] clk duct[30] clk duct[30] clk duct[30] clk duct[30] clk duct[30] clk	k periods. FMAX is compute ff analysis.	Relationship 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000	For paths where i and falling edges Clock Skew -0.063 -0.063 -0.062 -0.063 -0.063 -0.063 -0.063 -0.063 -0.063	Data Delay 3.275 3.269 3.256 3.208 3.171 3.164 3.159 3.156 3.153	stii wi
EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Summary Flow Settings Flow Settings Flow Non-Default Global Set Flow Lapsed Time Flow Lapsed Time Flow Log Analysis & Synthesis Fiter Assembler FimeQuest Timing Analyzer El Summary	Slov 1 2 3 4 5 6 7 7 8 9 10 11	This panel clocks, incl Altera reco v 1200mV 8 Slack 0.677 0.788 0.793 0.796 0.799 0.802	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'clk' From Node mutwithcik:nst[8_copy[1] mutwithcik:nst[8_copy[2] mutwithcik:nst[8_copy[2] mutwithcik:nst[8_copy[2] mutwithcik:nst[8_copy[3] mutwithcik:nst[8_copy[3] mutwithcik:nst[8_copy[3] mutwithcik:nst[8_copy[7] mutwithcik:nst[8_copy[7] mutwithcik:nst[8_copy[7] mutwithcik:nst[8_copy[7]	n the design, regardle nored. For paths bety clock constraints and To Noc mutwithclk-instgroor mutwithclk-instgroor mutwithclk-instgroor mutwithclk-instgroor mutwithclk-instgroor mutwithclk-instgroor mutwithclk-instgroor mutwithclk-instgroor mutwithclk-instgroor mutwithclk-instgroor	ess of the user-specified cloc ween a clock and its inversion other slack reports for sign te Launch Cloc duct[30] te Launch Cloc duct[31] te Launch Cloc duct[31] duct[31] clk duct[31] clk duct[31] clk duct[31] clk duct[31] clk duct[30] clk duct[31] clk duct[31] clk duct[31] clk duct[29] clk duct[29] clk duct[29] clk duct[29] clk	k periods. FMAX is compute ff analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000	Clock Skew -0.063 -0	Data Delay 3.275 3.256 3.208 3.164 3.159 3.156 3.156 3.150	stii wi
	s s s s s s s s s s s s s s s s s s s	This panel I Clocks, incl Aftera reco v 1200mV 8 Slack 0.677 0.683 0.697 0.744 0.783 0.793 0.794 0.796 0.799 0.799 0.799 0.802 0.812	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup; 'clk' From Node mutwithcikinstB_copy[1] mutwithcikinstB_copy[0] mutwithcikinstB_copy[0] mutwithcikinstB_copy[0] mutwithcikinstB_copy[3] mutwithcikinstB_copy[1] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7]	n the design, regardle nored. For paths bety clock constraints and To Noc mutwithclk.instgroo mutwithclk.instgroo mutwithclk.instgroo mutwithclk.instgroo mutwithclk.instgroo mutwithclk.instgroo mutwithclk.instgroo mutwithclk.instgroo mutwithclk.instgroo mutwithclk.instgroo mutwithclk.instgroo	ess of the user-specified cloc ween a clock and its inversion other slack reports for sign tion te Launch Clock tud[30] clk tud[31] clk tud[31] clk tud[31] clk tud[31] clk tud[31] clk tud[30] clk tud[30] clk tud[28] clk tud[29] clk tud[29] clk tud[29] clk tud[31] clk tud[30] clk	k periods. FMAX is compute ff analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000	Clock Skew -0.063 -0.063 -0.063 -0.062 -0.063 -0	Data Delay 3.275 3.266 3.208 3.171 3.164 3.159 3.158 3.150 3.150 3.140	stii wi
EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Summary Flow Settings Flow Non-Default Global Set Flow Clapsed Time Flow Cog Analysis & Synthesis Fiter Assembler TimeQuest Timing Analyzer Summary Faralel Complation For Cire List	Slow 1 2 3 4 5 6 6 7 8 9 10 11 12 13	This panel I clocks, incl Altera reco v 1200mV 8 Slack 0.677 0.683 0.794 0.744 0.782 0.793 0.794 0.796 0.799 0.802 0.813	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'cik' From Node mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[1] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7]	n the design, regardle nored. For paths bety clock constraints and To Noc mult with clk instgroo mult with clk instgroo	ass of the user-specified cloc ween a clock and its inversion other slack reports for sign-1 de Launch Clock duct[30] clk duct[31] clk duct[30] clk duct[20] clk duct[20] clk duct[20] clk duct[20] clk	k periods. FMAX is compute ff analysis.	Relationship 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000	For paths where i and failing edges Clock Skew -0.063 -0.063 -0.062 -0.063 -0.062 -0.063 -0.062 -0.062 -0.063 -0.063 -0.063 -0.062 -0.062 -0.063 -0.063 -0.063 -0.062 -0.062 -0.062 -0.063 -0.062 -0.062 -0.062 -0.062 -0.063 -0.062 -0.0	Data Delay 3.275 3.269 3.256 3.269 3.256 3.208 3.171 3.164 3.159 3.158 3.155 3.155 3.150 3.140	stii wi
	s s s s s s s s s s s s s s s s s s s	Vispanel clocks, incl Altera reco Slack 0.677 0.683 0.697 0.744 0.782 0.788 0.793 0.796 0.799 0.802 0.812 0.814	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'clk' From Node mutwithcikinst[B_copy[1] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[1] mutwithcikinst[B_copy[7	n the design, regardle nored. For paths bety clock constraints and To Noc mult with clk.instgrood mult with clk.instgrood	ses of the user-specified cloc ween a clock and its inversion other slack reports for sign de Launch Clock Juct[30] clk Juct[31] clk Juct[31] clk Juct[31] clk Juct[31] clk Juct[31] clk Juct[31] clk Juct[30] clk Juct[31] clk Juct[32] clk Juct[32] clk	k periods. FMAX is compute ff analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000	Clock Skew -0.063 -0.063 -0.063 -0.063 -0.062 -0.063 -0	Data Delay 3.275 3.256 3.269 3.256 3.208 3.171 3.164 3.159 3.153 3.156 3.153 3.150 3.140 3.140 3.138	stii wi
EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Suppressed Message Flow Summary Flow Summary Flow Summary Flow Lags Analysis & Synthesis Assembler TimeQuest Timing Analyzer Summary Parallel Complation SDC File List Clocks Clocks Clocks Clocks Flow Suppressed Message Flow Suppressed M	s s 1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	This panel i clocks, incl Altera recover \$1200mV 8 Slack 0.687 0.683 0.687 0.744 0.782 0.793 0.794 0.796 0.799 0.802 0.812 0.813 0.818	reports FMAX for every clock uding generated clocks, are ig mmends that you always use isc Model Setup; 'clk' From Node mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[0] mutwithcikinstB_copy[0] mutwithcikinstB_copy[0] mutwithcikinstB_copy[0] mutwithcikinstB_copy[0]	n the design, regardle nored. For paths bety clock constraints and To Noc multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo multwithclk.instgroo	Launch Clock Res of the user-specified cloc ween a clock and its inversion the mersion other slack reports for sign fe Launch Clock duct[31] clk duct[31] clk duct[31] clk duct[31] clk duct[31] clk duct[30] clk duct[31] clk duct[30] clk duct[28] clk duct[29] clk duct[31] clk duct[32] clk duct[32] clk duct[31] clk duct[32] clk duct[31] clk duct[32] clk duct[31] clk duct[32] clk duct[31] clk duct[31] clk	k periods. FMAX is compute f, FMAX is compute off analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000 4.000	For paths where i and failing edges Clock Skew -0.063 -0.063 -0.062 -0.063 -0.063 -0.063 -0.063 -0.063 -0.063 -0.063 -0.063 -0.063 -0.063 -0.063 -0.063 -0.063 -0.063 -0.063 -0.063	Data Delay 3.275 3.269 3.255 3.208 3.171 3.164 3.159 3.158 3.150 3.151 3.153 3.150 3.140 3.134	stii wi
EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Summary Flow Settings Flow Non-Default Global Set Flow Clapsed Time Flow CS Summary Flow CS Summary Flow Log Analysis & Synthesis Assembler For TimeQuest Timing Analyzer Summary Parallel Complation SDC File List Clocks Flow Summary & Stormary Slow 1200mV 85C Mode	Sov 1 2 3 4 5 6 6 7 8 8 9 10 11 12 13 14 15 16	This panel 1 clocks, incl Attera reco v 1200m V 8 Slack 0.683 0.683 0.783 0.788 0.783 0.794 0.799 0.799 0.802 0.813 0.814 0.814 0.839	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'cik' From Node mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7] mutwithcikinstB_copy[7]	n the design, regardle nored. For paths bety clock constraints and To Noc mult with clk instgroo mult with clk instgroo	ass of the user-specified cloc ween a clock and its inversion other slack reports for sign-1 te Launch Clock duct[30] clk duct[31] clk duct[30] clk duct[20] clk duct[20] clk duct[20] clk duct[31] clk duct[31] clk duct[31] clk duct[31] clk	k periods. FMAX is compute ff analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000	For paths where i and falling edges Clock Skew -0.063 -0.063 -0.062 -0.063 -0.0	Data Delay 3.275 3.256 3.256 3.256 3.269 3.171 3.164 3.159 3.155 3.155 3.155 3.155 3.150 3.140 3.140 3.138 3.134 3.134	stii wi
	Sov 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17	This panel i clocks, incl Attera reco v 1200mV 8 Slack 0.677 0.744 0.782 0.793 0.794 0.794 0.799 0.802 0.812 0.812 0.814 0.818 0.830	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'clk' From Node mutwithcikinst[B_copy[1] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[7	n the design, regardle nored. For paths bety clock constraints and To Noc multwithclk.instgroo	ass of the user-specified cloc ween a clock and its inversion other slack reports for sign de Launch Clock duct[30] clk duct[31] clk	k periods. FMAX is compute ff analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.0000 4.0000 4.0000 4.000 4.000 4.000 4.000 4.000	For paths where is and failing edgess Clock Skew -0.063 -0.062 -0.063 -0.062 -0.063 -0.0	Data Delay 3.275 3.269 3.256 3.208 3.171 3.164 3.159 3.153 3.153 3.153 3.150 3.140 3.138 3.140 3.138 3.134 3.134 3.133	stii wi
EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Suppressed Message Flow Summary Flow Summary Flow Summary Flow Lagsed Time Flow Lag Analysis & Synthesis Assembler Fitter Assembler Fitter Assembler Fitter Summary Farallel Complation SDC File List Clocks Fixer Slow 1200mV 85C Mode Fixer Trimg Closure Recc	1 2 3 4 5 6 6 7 7 8 9 10 11 12 13 14 15 16 17 18	This panel i clocks, incl Altera recover \$1200mV 8 Slack 0.687 0.744 0.782 0.793 0.794 0.796 0.802 0.812 0.813 0.818 0.839 0.879	reports FMAX for every clock uding generated clocks, are ig mmends that you always use isc Model Setup; 'clk' From Node mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstBproduct[1] mut	n the design, regardle nored. For paths bety clock constraints and To Noc multwithclk.instgroo	ses of the user-specified cloc ween a clock and its inversion other slack reports for sign fe Launch Clock duct[31] clk duct[32] clk duct[33] clk duct[33] clk duct[33] clk duct[31] clk duct[32] clk duct[33] clk duct[31] clk	k periods. FMAX is compute ff analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000	Tor paths where i and failing edges Clock Skew -0.063 -0.0	Data Delay 3.275 3.269 3.256 3.208 3.171 3.164 3.159 3.150 3.140 3.134 3.134 3.130	stii wi
 EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Summary Flow Settings Flow Non-Default Global Set Flow Lapsed Time Flow Log Analysis & Synthesis Fiter Assembler TimeQuest Timing Analyzer Summary Flow Elist Cocks Slow 1200mV 85C Mode Fitming Closure Recc Setin Summary 	S S S 1 1 2 3 4 5 6 6 7 8 9 10 111 12 13 14 15 16 17 18 19 19	This panel I clocks, incl Attera reco v 1200m V 8 Slack 0.683 0.683 0.783 0.784 0.783 0.794 0.799 0.802 0.813 0.814 0.813 0.814 0.839 0.860 0.879	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'cik' From Node mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[7] mutwithcikinstB_copy[n the design, regardle nored. For paths bety clock constraints and To Noc mult with clk instgroo mult with clk instgroo	ass of the user-specified cloc ween a clock and its inversion other slack reports for sign-1 te Launch Clock duct[30] clk duct[31] clk duct[32] clk duct[30] clk duct[31] clk duct[32] clk duct[31] clk	k periods. FMAX is compute ff analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.0000 4.0000 4.0000 4.000 4.000 4.000 4.000 4.000	For paths where i and failing edges Clock Skew -0.063 -0.062 -0.063 -0.06	Data Delay 3.275 3.256 3.256 3.256 3.256 3.269 3.171 3.164 3.159 3.155 3.155 3.155 3.155 3.150 3.140 3.140 3.140 3.138 3.134 3.134 3.134 3.133 3.131 3.092 3.073 3.067	stii wi
	S S S S S S S S S S S S S S S S S S S	This panel i clocks, incl Attera reco v 1200mV 8 Slack 0.677 0.744 0.782 0.793 0.794 0.796 0.799 0.802 0.812 0.812 0.814 0.818 0.814 0.818 0.860 0.879 0.860 0.879 0.865 0.80	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'clk' From Node mutwithcikinst[B_copy[1] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[7	n the design, regardle nored. For paths bety clock constraints and To Noc multwithclkinstproc	Application Cite action Cite Cite	k periods. FMAX is compute ff analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.0000 4.0000 4.000 4.000 4.000 4.000 4.000 4.000	Clock Skew -0.063 -0.063 -0.062 -0.063 -0.062 -0.063 -0	Data Delay 3.275 3.256 3.269 3.256 3.208 3.171 3.164 3.153 3.153 3.140 3.138 3.140 3.138 3.130 3.131 3.092 3.073 3.067 3.057	stii wi
EDA Netlist Writer Flow Messages Flow Suppressed Message Flow Suppressed Message Flow Suppressed Message Flow Summary Flow Summary Flow Non-Default Global Set Flow Non-Default Global Set Flow Cag Analysis & Synthesis Fitter Assembler TimeQuest Timing Analyzer Summary Parallel Complation SDC File List Clocks Slow 1200mV 85C Mode Fmax Prace Setup Summary Fitting Closure Recc Setup Summary Hold Summary Hold Summary	Slow 1 2 3 4 5 6 6 7 8 9 10 11 12 13 14 5 6 6 7 8 9 10 11 12 13 14 5 6 7 7 8 9 9 10 11 12 3 4 5 6 7 7 10 11 12 3 14 5 5 6 6 7 7 10 10 10 10 10 10 10 10 10 10 10 10 10	This panel i clocks, incl Altera recover \$1200mV 8 Slack 0.687 0.744 0.782 0.793 0.794 0.795 0.812 0.812 0.814 0.818 0.839 0.860 0.879	reports FMAX for every clock uding generated clocks, are ig mmends that you always use SC Model Setup; 'clk' From Node mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[0] mutwithcikinstB_copy[0] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[5	n he design, regardle nored. For paths bety clock constraints and To Noc mult withclk instgroo mult withclk instgroo	ses of the user-specified cloc ween a clock and its inversion other slack reports for sign fe Launch Cloc duct[30] ck duct[31] ck duct[30] ck duct[31] ck duct[32] ck duct[33] ck duct[31] ck duct[32] ck duct[31] ck duct[31] ck duct[31] ck duct[31] ck duct[31] ck duct[31] ck duct[30] ck duct[31] ck	k periods. FMAX is compute ff analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000	Tor paths where i and failing edges Clock Skew -0.063 -0.0	Data Delay 3.275 3.269 3.269 3.266 3.268 3.171 3.164 3.159 3.158 3.156 3.153 3.150 3.140 3.140 3.134 3.134 3.134 3.134 3.134 3.134 3.134 3.134 3.134 3.134 3.134 3.134 3.135 3.156 3.157 3.057 3	stii wi
 EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Summary Flow Settings Flow Non-Default Global Set Flow Lapsed Time Flow Log Analysis & Synthesis Fiter Assembler Formacust Timing Analyzer Summary Flow Elist Clocks Soloc File List Clocks Steup Summary Timing Closure Recc Setup Summary Timing Closure Recc Setup Summary Hold Summary Recovery Summary 	s s s s s s s s s s s s s s s s s s s	This panel I clocks, incl Attera recover v 1200mV 8 Slack 0.683 0.683 0.683 0.783 0.783 0.784 0.783 0.794 0.799 0.802 0.813 0.814 0.813 0.814 0.839 0.860 0.879 0.860 0.879 0.885 0.898 0.895	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'cik' From Node mutwithcik:nstB_copy[1] mutwithcik:nstB_copy[2] mutwithcik:nstB_copy[2] mutwithcik:nstB_copy[3] mutwithcik:nstB_copy[3] mutwithcik:nstB_copy[3] mutwithcik:nstB_copy[7] mutwithcik:nstB_copy[n he design, regardle nored. For paths bety clock constraints and To Noc mult with clk instgroo mult with clk instgroo	ass of the user-specified cloc ween a clock and its inversion other slack reports for sign-1 de Launch Clock duct[30] clk duct[31] clk duct[30] clk duct[31] clk duct[31] clk duct[31] clk duct[31] clk duct[31] clk duct[31] clk duct[32] clk duct[31] clk duct[32] clk duct[31] clk duct[32] clk <td>k periods. FMAX is compute ff analysis.</td> <td>Relationship 4.000 4.</td> <td>For paths where i and falling edges Clock Skew -0.063 -0.062 -0.063 -0.062 -0.063 -0.062 -0.06</td> <td>Data Delay 3.275 3.269 3.256 3.209 3.171 3.164 3.155 3.156 3.156 3.153 3.150 3.140 3.138 3.140 3.138 3.131 3.092 3.057 3.055 3.057</td> <td>stii wi</td>	k periods. FMAX is compute ff analysis.	Relationship 4.000 4.	For paths where i and falling edges Clock Skew -0.063 -0.062 -0.063 -0.062 -0.063 -0.062 -0.06	Data Delay 3.275 3.269 3.256 3.209 3.171 3.164 3.155 3.156 3.156 3.153 3.150 3.140 3.138 3.140 3.138 3.131 3.092 3.057 3.055 3.057	stii wi
EDA Netlist Writer Flow Messages Flow Suppressed Messages Flow Suppressed Message Flow Summary Flow Summary Flow Capased Time Flow Lapsed Time Flow Lag Analysis & Synthesis Fiter Analysis & Synthesis Fiter Summary Farallel Compliation SUC File List Clocks Solv 1200mV 85C Mode Finang Clocks File Summary Finang Clocks File Summary Finang Clocks File Summary File Clock Summary File Mod Summary File Summary File Summary File Clocks File Summary File Clocks File Summary File Clock Summary	s slow 1 2 3 4 5 6 7 8 9 10 11 12 3 4 5 6 7 8 9 10 11 12 3 4 5 6 6 7 8 9 10 11 12 3 4 5 6 6 7 8 9 10 2 11 12 3 4 5 6 6 6 10 11 12 13 14 5 6 6 10 10 10 10 10 10 10 10 10 10	This panel I clocks, incl Altera recover × 1200mV 8 Slack 0.677 0.683 0.697 0.744 0.782 0.793 0.794 0.799 0.802 0.812 0.813 0.814 0.860 0.879 0.885 0.896 0.899	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'clk' From Node mutwithcikinst[B_copy[1] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[7	n the design, regardle nored. For paths bety clock constraints and To Noc multwithclkinstproc	ass of the user-specified cloc ween a clock and its inversion other slack reports for sign de Launch Clock Juct[30] clk Juct[31] clk Juct[30] clk Juct[30] clk Juct[31] clk Juct[31] clk Juct[30] clk Juct[30] clk Juct[31] clk </td <td>k periods. FMAX is compute ff analysis. Latch Clock clk clk clk clk clk clk clk c</td> <td>Relationship 4.000 4.</td> <td>Tor paths where is and failing edgess closes of the second second</td> <td>Data Delay 3.275 3.269 3.256 3.208 3.171 3.164 3.153 3.153 3.150 3.140 3.138 3.140 3.138 3.140 3.130 3.140 3.130 3.092 3.073 3.055 3.050</td> <td>stii wi</td>	k periods. FMAX is compute ff analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000 4.	Tor paths where is and failing edgess closes of the second	Data Delay 3.275 3.269 3.256 3.208 3.171 3.164 3.153 3.153 3.150 3.140 3.138 3.140 3.138 3.140 3.130 3.140 3.130 3.092 3.073 3.055 3.050	stii wi
EDA Netlist Writer Flow Messages Flow Suppressed Message Flow Suppressed Message Flow Suppressed Message Flow Suppressed Message Flow Summary Flow Summary Flow Summary Flow Lapsed Time Flow Lapsed Time Flow CS Summary Fitter Analysis & Synthesis Fitter Assembler TimeQuest Timing Analyzer Summary Summary Source Statemary Source Statemary Source Statemary Finance Concept Summary Source Statemary Fitter Assembler Source Statemary Fitter Fallel Compliation Source Concept Source Statemary Fitter Fitter Assembler Fitter Assembler Fitter Assembler Fitter House Statemary Fitter Fitter Balance Statemary Fitter Fitter Recovery Summary	s s s s s s s s s s s s s s s s s s s	This panel I clocks, incl Altera recover \$1200mV 8 Slack 0.687 0.744 0.762 0.793 0.794 0.796 0.802 0.812 0.814 0.818 0.839 0.865 0.895 0.896	reports FMAX for every clock uding generated clocks, are ig mmends that you always use SC Model Setup: 'clk' From Node mutwithckinstB_copy[1] mutwithckinstB_copy[0] mutwithckinstB_copy[0] mutwithckinstB_copy[0] mutwithckinstB_copy[0] mutwithckinstB_copy[1] mutwithckinstB_copy[1] mutwithckinstB_copy[1] mutwithckinstB_copy[1] mutwithckinstB_copy[1] mutwithckinstB_copy[1] mutwithckinstB_copy[1] mutwithckinstB_copy[1] mutwithckinstB_copy[1] mutwithckinstB_copy[1] mutwithckinstB_copy[1] mutwithckinstB_copy[2] mutwithckinstB_copy[2] mutwithckinstB_copy[2] mutwithckinstB_copy[5] mutwithckinstB_copy[5] mutwithckinstB_copy[6] mu	n he design, regardle nored. For paths bety clock constraints and To Noo mult withclk instgroo mult withclk instgroo	ses of the user-specified cloc ween a clock and its inversion other slack reports for sign fe Launch Cloc duct[30] clk duct[31] clk duct[30] clk duct[31] clk duct[32] clk duct[30] clk duct[31] clk duct[30] clk duct[30] clk duct[30] clk duct[31] clk duct[32] clk	k periods. FMAX is compute ff analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.0000 4.0000 4.0000 4.0000 4.0000 4.0000 4.0000 4.00000 4.00000 4.00000 4.00000000	For paths where is and failing edgess Clock Skew -0.063 -0	Data Delay 3.275 3.269 3.256 3.269 3.269 3.171 3.164 3.158 3.156 3.150 3.140 3.143 3.133 3.134 3.133 3.073 3.067 3.055 3.050 3.048	stii wi
	s Slov 1 2 3 4 5 6 7 8 9 10 11 2 3 4 5 6 7 8 9 10 11 2 3 4 5 6 7 8 9 9 10 11 12 3 4 5 6 7 8 9 9 10 11 12 13 14 15 16 10 10 10 10 10 10 10 10 10 10	This panel I clocks, incl Attera recover v 1200mV 8 Slack 0.673 0.683 0.683 0.783 0.784 0.783 0.794 0.786 0.799 0.802 0.813 0.814 0.813 0.814 0.813 0.814 0.839 0.860 0.879 0.865 0.895 0.896	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'cik' From Node mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[7] mutwithcikinstB_copy[n he design, regardle nored. For paths bety clock constraints and To Noc mult with clk instgroo mult with clk instgroo	ass of the user-specified cloc ween a clock and its inversion other slack reports for sign-t fued(30) de Launch Clock duct[31] de Launch Clock duct[31] duct[31] clk duct[32] clk duct[30] clk duct[30] clk duct[30] clk duct[30] clk duct[30] clk duct[30] clk duct[31] clk duct[32] clk duct[31] clk duct[32] clk duct[31] clk duct[32]	k periods. FMAX is compute ff analysis.	Relationship 4.000	For paths where i and failing edges and failing edges clock Skew -0.063 -0.063 -0.062 -0.063 -0.062 -0.063 -0.062	Data Delay 3.275 3.269 3.256 3.269 3.256 3.269 3.151 3.155 3.155 3.156 3.153 3.150 3.140 3.138 3.140 3.138 3.140 3.138 3.140 3.138 3.140 3.138 3.140 3.138 3.140 3.138 3.057 3.055 3.055 3.050 3.048 3.047	stii wi
	s Slow 1 1 2 3 4 4 5 6 6 7 8 9 10 11 12 13 4 5 6 6 7 8 9 9 10 11 12 13 14 15 16 17 12 23 24 22 23 24 25	This panel I clocks, incl Altera recover × 1200mV 8 Slack 0.677 0.683 0.697 0.744 0.782 0.793 0.794 0.799 0.802 0.813 0.814 0.818 0.839 0.869 0.869 0.879 0.886 0.898 0.902 0.904 0.906	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'clk' From Node mutwithcikinst[B_copy[1] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[7	n the design, regardle nored. For paths bety clock constraints and To Noc mult with clk.instgroo mult with clk.instgroo	ses of the user-specified cloc ween a clock and its inversion other slack reports for sign de Launch Clock duct[30] clk duct[31] clk duct[31] clk duct[31] clk duct[31] clk duct[30] clk duct[31] clk duct[30] clk duct[31] clk </td <td>k periods. FMAX is , FMAX is compute off analysis. Latch Clock clk clk clk clk clk clk clk c</td> <td>Relationship 4.000 4.</td> <td>For paths where is and failing edgess class of the second second</td> <td>Data Delay 3.275 3.256 3.269 3.256 3.209 3.171 3.164 3.159 3.153 3.150 3.140 3.138 3.140 3.133 3.140 3.133 3.092 3.073 3.055 3.050 3.047</td> <td>stii wi</td>	k periods. FMAX is , FMAX is compute off analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000 4.	For paths where is and failing edgess class of the second	Data Delay 3.275 3.256 3.269 3.256 3.209 3.171 3.164 3.159 3.153 3.150 3.140 3.138 3.140 3.133 3.140 3.133 3.092 3.073 3.055 3.050 3.047	stii wi
 EDA Netlist Writer Flow Messages Flow Suppressed Message Flow Suppressed Message Flow Suppressed Message Flow Suppressed Message Flow Summary Flow Settings Flow Analysis & Synthesis Fitter Analysis & Synthesis Fitter Assembler Fitter Assembler Summary Parallel Complation SDC File List Clocks Fing Clocks Fing Clocks Fing Summary Recovery Summary Recovery Summary Recovery Summary Removal Summary Minimum Pulse Widt Worst-Case Timing 1 Setup: Citk 	s Slow 1 1 2 3 4 4 5 6 6 7 8 9 9 10 11 12 13 14 5 6 7 8 9 9 10 11 12 13 14 5 6 7 8 9 9 10 11 12 3 4 4 5 6 7 8 9 9 10 11 2 3 4 4 5 6 6 7 7 8 9 9 10 10 11 12 3 3 4 4 5 6 6 7 7 8 9 9 10 11 12 13 10 11 10 10 10 10 10 10 10 10 10 10 10	This panel i clocks, incl Altera recover \$1200mV & Slack 0.687 0.744 0.762 0.793 0.794 0.799 0.802 0.812 0.814 0.818 0.879 0.865 0.896 0.902 0.906 0.906	reports FMAX for every clock uding generated clocks, are ig mmends that you always use isc Model Setup: 'clk' From Node mutwithckinstB_copy[1] mutwithckinstB_copy[2] mutwithckinstB_copy[2] mutwithckinstB_copy[3] mutwithckinstB_copy[3] mutwithckinstB_copy[3] mutwithckinstB_copy[7] m	n he design, regardle nored. For paths bety clock constraints and To Noc multwithclkinstpro-	ass of the user-specified cloc ween a clock and its inversion other slack reports for sign te Launch Cloc duc[30] te Launch Cloc duc[31] te Launch Cloc duc[30] duct[31] clk duct[31] clk duct[31] clk duct[31] clk duct[31] clk duct[30] clk duct[31] clk duct[31] clk duct[30] clk duct[30] clk duct[31] clk duct[32] clk duct[31]	k periods. FMAX is compute ff analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000	For paths where is and failing edgess Clock Skew -0.063 -0.062 -	Data Delay 3.275 3.269 3.256 3.269 3.269 3.261 3.171 3.164 3.158 3.156 3.153 3.150 3.140 3.143 3.133 3.134 3.133 3.073 3.067 3.057 3.055 3.048 3.047 3.047	stii wi
 EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Summary Flow Settings Flow Non-Default Global Set Flow Lapsed Time Flow CS Summary Flow CS Summary Flow Log Analysis & Synthesis Fiter Assembler Summary Flow Log Kasembler Sour Flat Compilation SDC File List Clocks Solv 200mV 85C Mode Fmax Summary Timing Closure Recc Setup Summary Hold Summary Minimum Pube Widt Worst-Case Timing Hold: 'cit' 	S Slove 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 17 22 23 24 25 22 27	This panel I clocks, incl Altera recover 200mV 8 Slack 0.683 0.683 0.783 0.783 0.793 0.794 0.799 0.802 0.813 0.814 0.839 0.860 0.879 0.885 0.898 0.904 0.906 0.906	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'cik' From Node mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[7] mutwithcikinstB_copy[n he design, regardle nored. For paths bety clock constraints and To Noc mult with clk instgroo mult with clk instgroo	ass of the user-specified cloc ween a clock and its inversion other slack reports for sign- de Launch Clock duct[30] clk duct[31] clk duct[32] clk duct[30] clk duct[31] clk duct[32] clk duct[31] clk duct[32] clk duct[31] clk duct[31] clk	k periods. FMAX is compute ff analysis. Calc Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000	For paths where i and failing edgess 	Data Delay 3.275 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.256 3.171 3.164 3.155 3.156 3.153 3.150 3.140 3.140 3.133 3.140 3.133 3.092 3.057 3.055 3.048 3.047 3.047 3.047 3.047	stii wi
 EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Summary Flow Settings Flow Settings Flow Capable State State Flow Capable State State Flow Capable State Flow Summary Flow Capable State Flow Summary Flow Capable State Flow Summary Flow Capable State Summary Parallel Compilation SDC File List Clocks Sow 1200mV 85C Mode Franz Summary Hold Summary Recovery Summary Monimum Pulse Widtl Worst-Case Timing I Statep: Clock Worst-Case Timing I Statep: Clock 	Sove 1 1 2 3 4 4 5 6 6 7 8 9 10 11 11 12 13 14 15 16 17 18 9 20 21 22 22 24 25 26 26 27 28	This panel I clocks, incl Altera recover × 1200mV 8 Slack 0.697 0.744 0.788 0.793 0.794 0.799 0.802 0.813 0.814 0.818 0.8260 0.879 0.825 0.898 0.906 0.906	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'cik' From Node mutwithcikinst[B_copy[1] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[1] mutwithcikinst[B_copy[1] mutwithcikinst[B_copy[1] mutwithcikinst[B_copy[1] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[4] mutwithcikinst[B_copy[4] mutwithcikinst[B_copy[4] mutwithcikinst[B_copy[4] mutwithcikinst[B_copy[3	n he design, regardle nored. For paths bety clock constraints and To Noc mult with clk.instproc mult with clk.instproc	ass of the user-specified cloc ween a clock and its inversion other slack reports for sign- de Launch Clock duct[30] clk duct[31] clk duct[30] clk duct[30] clk duct[30] clk duct[30] clk duct[30] clk duct[31] clk duct[30] clk duct[31] clk duct[31] clk	k periods. FMAX is compute ff analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000 4.	for paths where is and failing edgess class of the second	Data Delay 3.275 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.151 3.152 3.153 3.150 3.140 3.138 3.140 3.138 3.140 3.133 3.092 3.073 3.057 3.055 3.047 3.047 3.047 3.047 3.047	stii
> EDA Netlist Writer • Flow Messages • Flow Suppressed Message • Flow Settings • Flow Non-Default Global Set • Flow Lapsed Time • Flow Log • Analysis & Synthesis • Fiter • Assembler • Fiter • Assembler • Slow 1200m/V 85C Mode • Finang Closure Recc • Slow 1200m/V 85C Mode • Finang Closure Recc • Setup Summary • Minimum Pulse Widt • Worst-Case Timing I • Stow 1200m/V 0C Model	Slov 1 2 3 4 5 6 7 8 9 10 11 2 3 4 5 6 9 10 11 12 13 14 15 10 11 12 23 4 25 22 23 24 22 23 24 22 23 24 22 23 24 22 23 24 22 22 23 24 22 22 22 22 22 22 22 22 22	This panel I clocks, incl Attera reco v 1200m V 8 58ack 0.697 0.744 0.783 0.793 0.793 0.799 0.799 0.799 0.812 0.812 0.813 0.814 0.818 0.839 0.860 0.813 0.814 0.839 0.860 0.875 0.885 0.895 0.885 0.895 0.895 0.895 0.895 0.906	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup; 'cik' From Node mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[4] mutwithcikinstB_copy[3] mutwithcikinstB_copy[n he design, regardle nored. For paths bety clock constraints and To Noc mult withclk instgroo mult withclk instgroo	sso of the user-specified cloc ween a clock and its inversion other slack reports for sign le Launch Cloc duct[30] le Launch Cloc duct[31] le Launch Cloc duct[31] duct[31] clk duct[31] clk duct[31] clk duct[31] clk duct[31] clk duct[30] clk duct[31] clk duct[31] clk duct[30] clk duct[30] clk duct[30] clk duct[31] clk duct[32] clk duct[31] clk duct[31]	k periods. FMAX is compute ff analysis.	Relationship 4.000 4.	For paths where is and failing edgess Clock Skew -0.063 -0.062 -	Data Delay 3.275 3.269 3.256 3.269 3.269 3.171 3.164 3.153 3.156 3.153 3.150 3.140 3.143 3.134 3.134 3.134 3.057 3.057 3.055 3.048 3.047 3.047 3.047 3.047 3.047 3.047 3.047	stii wi
 EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Suppressed Message Flow Summary Flow Settings Flow Settings Flow Lapsed Time Flow Log Analysis & Synthesis Fitter Assembler Flow Compliation SUC File List Clocks Solv 1200mV 85C Mode Timing Closure Recc Setup Summary Recovery Summary Mod Summary Mod Summary Recovery Summary Minimum Pube Widt Worst-Case Timing 1 Sow 1200mV 0C Model Fast 1200mV 0C Model 	Slow 1 1 2 3 4 4 5 6 7 8 9 9 10 11 12 13 4 5 6 7 8 9 9 10 11 12 13 4 15 16 17 8 19 20 21 22 23 24 25 26 27 28 29 20 23 30	This panel I clocks, incl Altera recover 200mV 8 Slack 0.683 0.697 0.744 0.783 0.793 0.794 0.799 0.802 0.813 0.814 0.839 0.860 0.879 0.885 0.898 0.904 0.906 0.906	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'cik' From Node mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[7] mutwithcikinstB_copy[n he design, regardle nored. For paths bety clock constraints and To Noc mult with clk instgroo mult with clk instgroo	ass of the user-specified cloc ween a clock and its inversion other slack reports for sign- de Launch Clock duct[30] clk duct[31] clk duct[30] clk duct[31] clk duct[32] clk duct[31] clk duct[31] clk duct[32] clk	k periods. FMAX is compute ff analysis.	Relationship 4.000 4.	For paths where i and failing edgess and failing edgess clock Skew -0.063 -0.062 -0.063 -0.062 -0.063 -0.062 -0.063 -0.062 -0.063 -0.062 -0.063 -0.062 -0.0062 -0.0	Data Delay 3.275 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.256 3.171 3.164 3.155 3.156 3.153 3.150 3.140 3.140 3.143 3.133 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047	stii wi
 EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Suppressed Message Flow Summary Flow Settings Flow Content Global Set Flow Non-Default Global Set Flow Lagaed Time Flow Lagaed Time Flow OS Summary Flow Lagaed Time Maysis & Synthesis Fiter Analysis & Synthesis Fiter Analysis & Synthesis Fiter Masembler TomoQuest Timing Analyzer Sourmary Parallel Compilation SDC Fite List Clocks Stow 1200mV 85C Mode Freat Summary Removal Summary Minimum Pulse Widtt Worst-Case Timing I Stow 1200mV 0C Model Test 200mV 0C Model Test 1200mV 0C Model 	Sov 1 2 3 4 5 6 7 8 9 9 10 11 12 3 4 5 6 7 8 9 9 10 11 12 3 4 5 6 7 8 9 9 10 11 12 3 4 5 6 7 8 9 9 10 11 12 3 4 5 6 7 8 9 9 10 11 12 13 14 15 16 10 10 10 10 10 10 10 10 10 10	This panel I clocks, incl Altera recover × 1200mV 8 Slack 0.697 0.783 0.782 0.788 0.793 0.794 0.799 0.802 0.812 0.813 0.814 0.855 0.898 0.902 0.906 0.906 0.906 0.906	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'cik' From Node mutwithcikinst[B_copy[1] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[7	n he design, regardle nored. For paths bety clock constraints and To Noc mult with clk.instproc mult with clk.instproc	ses of the user-specified cloc ween a clock and its inversion other slack reports for sign- le Launch Clock duct[30] ck duct[31] ck duct[30] ck duct[31] ck duct[32] ck duct[31] ck duct[32] ck duct[31] ck duct[31] ck duct[32] ck duct[3	k periods. FMAX is , FMAX is compute off analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000 4.	for paths where i and falling edges Clock Skew -0.063 -0.063 -0.062 -0.063 -0.062 -0.	Data Delay 3.275 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.151 3.152 3.153 3.150 3.140 3.138 3.140 3.138 3.140 3.133 3.092 3.073 3.057 3.055 3.050 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047	sti
> EDA Netlist Writer • Flow Messages • Flow Suppressed Message • Flow Suppressed Message • Flow Suppressed Message • Flow Summary • Flow Settings • Flow Non-Default Global Set • Flow Non-Default Global Set • Flow Lapsed Time • Flow Log > Analysis & Synthesis > Fiter > Assembler • Fiter > Assembler • Slow 1200m/V 85C Mode • Finang Closure Recc • Slow 1200m/V 85C Mode • Finang Closure Recc • Slow 1200m/V 85C Mode • Finang Closure Recc • Slow 1200m/V 85C Mode • Minimum Fulse Wdtt • Worst-Case Timing I • Slow 1200m/V 0C Model > Slow 1200m/V 0C Model > Fast 1200m/V 0C Model > Fast 1200m/V 0C Model	Sov 1 1 2 3 4 5 6 7 8 9 9 10 11 12 3 4 5 6 7 8 9 9 10 11 12 3 4 5 6 7 8 9 9 10 11 12 3 4 5 6 7 8 9 9 10 11 12 3 4 4 5 6 6 7 8 9 9 10 11 12 13 14 15 16 16 17 16 17 16 17 16 17 17 17 17 17 17 17 17 17 17	This panel I clocks, incl Attera reco v 1200m V 8 Slack 0.697 0.744 0.783 0.793 0.793 0.793 0.795 0.799 0.802 0.812 0.813 0.814 0.818 0.818 0.839 0.860 0.813 0.814 0.839 0.860 0.875 0.885 0.895 0.885 0.895 0.885 0.895 0.895 0.895 0.895 0.906 0.906 0.906 0.906 0.906	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup; 'cik' From Node mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[4] mutwithcikinstB_copy[4] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[4] mutwithcikinstB_copy[3] mutwithcikinstB_copy[n he design, regardle nored. For paths bety clock constraints and To Noc mult withclk instgroo mult withclk instgroo	ass of the user-specified cloc ween a clock and its inversion other slack reports for sign- le Launch Clock duct[30] ck duct[31] ck duct[30] ck duct[31] ck duct[30] ck duct[31] ck duct[30] ck duct[31] ck duct[30] ck duct[31] ck duct[32] ck duct[31] ck duct[31] ck duct[1	k periods. FMAX is compute ff analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000 4.	For paths where i and failing edges and failing edges 0.063 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062	Data Delay 3.275 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.256 3.171 3.164 3.155 3.156 3.150 3.140 3.140 3.143 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047	sti
 EDA Netlist Writer Flow Messages Flow Suppressed Message Flow Suppressed Message Flow Suppressed Message Flow Summary Flow Settings Flow Settings Flow Settings Flow Settings Flow Source Flow Suppressed Message Flow Settings Flow Settings Flow Settings Flow Settings Flow Summary Flow Suppression Analysis & Synthesis Fitter Assembler Summary Parallel Compilation SDC File List Clocks Solu 1200mV 85C Mode Fmax Summary Recovery Summary Method Summary Method Summary Mol Summary Setup Summary Metastability Summary Slow 1200mV 0C Model Fast 1200mV 0C Model Multicorner Timing Analy Advanced V0 Timing 	S S S S S S S S S S S S S S S S S S S	This panel I clocks, incl Altera recover 200mV 8 Slack 0.683 0.683 0.683 0.783 0.783 0.783 0.793 0.794 0.799 0.802 0.813 0.814 0.839 0.860 0.895 0.896 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'cik' From Node mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[7] mutwithcikinstB_copy[n he design, regardle nored. For paths bety clock constraints and To Noc mult with clk instgroo mult with clk instgroo	ass of the user-specified cloc ween a clock and its inversion other slack reports for sign- de Launch Clock duct[30] clk duct[31] clk duct[32] clk duct[30] clk duct[30] clk duct[30] clk duct[30] clk duct[30] clk duct[30] clk duct[31] clk	k periods. FMAX is compute ff analysis.	Relationship 4.000 4.	For paths where i and failing edgess and failing edgess -0.063 -0.062 -0.063 -0.062 -0	Data Delay 3.275 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.256 3.171 3.164 3.159 3.156 3.153 3.150 3.140 3.140 3.138 3.140 3.133 3.092 3.057 3.055 3.0567 3.055 3.0467 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047	sti
 EDA Netlist Writer Flow Messages Flow Suppressed Message Flow Suppressed Message Flow Suppressed Message Flow Summary Flow Settings Flow Settings Flow Capable Source Flow Construction Flow Summary Flow Capable Source Flow Summary Flow Construction Flow Summary Flow Construction Source Summary Parallel Compilation SOC File List Clocks Sour 1200mV 85C Mode Franz Summary Hold Summary Recovery Summary Molinium Pulse Width Worst-Case Timing I Slow 1200mV 0C Model Metastability Summa Slow 1200mV 0C Model Maticomer Timing Analy Advanced V0 Timing Clock Transfers 	S S S S S S S S S S S S S S S S S S S	This panel I clocks, incl Altera recover × 1200mV 8 Slack 0.687 0.683 0.697 0.744 0.788 0.793 0.794 0.799 0.802 0.813 0.814 0.818 0.896 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'cik' From Node mutwithcikinst[B_copy[1] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[2] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[3] mutwithcikinst[B_copy[7	n he design, regardle nored. For paths bety clock constraints and To Noc mult with clk.instproc mult with clk.instproc	ses of the user-specified cloc eween a clock and its inversion other slack reports for sign- fe Launch Clock duct[30] ck duct[31] ck duct[32] ck duct[31] ck duct[32] ck duct[31] ck duct[32] ck duct[33] ck duct[31] ck duct[32] ck <t< td=""><td>k periods. FMAX is , FMAX is compute off analysis. Latch Clock clk clk clk clk clk clk clk c</td><td>Relationship 4.000 4.</td><td>For paths where i and falling edges Clock Skew -0.063 -0.063 -0.062 -0.063 -0.062 -0.06</td><td>Data Delay 3.275 3.269 3.268 3.269 3.268 3.269 3.268 3.269 3.268 3.269 3.268 3.269 3.268 3.269 3.269 3.161 3.153 3.150 3.140 3.143 3.140 3.143 3.140 3.140 3.143 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047</td><td>stii wi</td></t<>	k periods. FMAX is , FMAX is compute off analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000 4.	For paths where i and falling edges Clock Skew -0.063 -0.063 -0.062 -0.063 -0.062 -0.06	Data Delay 3.275 3.269 3.268 3.269 3.268 3.269 3.268 3.269 3.268 3.269 3.268 3.269 3.268 3.269 3.269 3.161 3.153 3.150 3.140 3.143 3.140 3.143 3.140 3.140 3.143 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047	stii wi
> EDA Netlist Writer • Flow Messages • Flow Suppressed Message • Flow Suppressed Message • Flow Suppressed Message • Flow Summary • Flow Settings • Flow Non-Default Global Set • Flow Non-Default Global Set • Flow Lagsed Time • Analysis & Synthesis > • • Fater • Assembler • Sourmary Brow Sour 200mV 85C Mode • Stow 1200mV 85C Mode • Stow 1200mV 85C Mode • Stow 200mV 85C Mode • Stow 120mV 85C Mode • Stow 120mV 02 Model • Stow 120mV 02 Model • Stow 120mV 02 Model • Fast 1200mV 02 Model • Fast 1200mV 02 Model • Advanced W0 Timing • Advanced W0 Timing • Report TCCS	S Slow 1 2 3 4 5 6 7 8 9 10 11 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 9 30 31 24 20 20 21 20 21 20 21 21 20 21 21 21 21 21 21 21 21 21 21	This panel I clocks, incl Attera reco v 1200m V 8 Slack 0.677 0.744 0.783 0.793 0.793 0.793 0.793 0.795 0.799 0.802 0.812 0.812 0.813 0.814 0.818 0.839 0.860 0.813 0.814 0.818 0.839 0.860 0.875 0.822 0.812 0.813 0.814 0.839 0.860 0.906 0.906 0.906 0.906 0.906 0.906 0.906	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup; 'cik' From Node mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[4] mutwithcikinstB_copy[4] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[4] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[4] mutwithcikinstB_copy[3] mutwithcikinsB_copy[3] mutwithcikinstB_copy[3	n he design, regardle nored. For paths bety clock constraints and To Noc mult withclk instgroo mult withclk in	ass of the user-specified cloc ween a clock and its inversion other slack reports for sign- le Launch Clock duct[30] ck duct[31] ck duct[30] ck duct[31] ck duct[30] ck duct[31] ck duct[3	k periods. FMAX is compute ff analysis.	Relationship 4.000 4.	For paths where i and failing edgess and failing edgess 0.063 0.062	Data Delay 3.275 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.171 3.164 3.155 3.156 3.150 3.140 3.140 3.143 3.043 3.067 3.057 3.055 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047 3.047	stii
 EDA Netlist Writer Flow Messages Flow Suppressed Message Flow Suppressed Message Flow Suppressed Message Flow Summary Flow Settings Flow Settings Flow Settings Flow Settings Flow Source Flow Suppressed Message Flow Settings Flow Settings Flow Settings Flow Summary Flow Summary Flow Suppression Analysis & Synthesis Fitter Assembler Summary Parallel Compilation SDC File List Clocks Solu 1200mV 85C Mode Fmax Summary Hold Summary Recovery Summary Minimum Pube Width Worst-Case Timing 1 Solw 1200mV 0C Model Fast 1200mV 0C Model Fast 1200mV 0C Model Muticorner Timing Analy Advanced V0 Timing Clock Timing Clock Timing Clock Timing Clock Timing Analy Report TCCS Report RESKM 	S S S S S S S S S S S S S S S S S S S	This panel I clocks, incl Altera recover 200mV 8 Slack 0.683 0.683 0.683 0.784 0.783 0.793 0.794 0.799 0.802 0.813 0.814 0.885 0.896 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'cik' From Node mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[7] mutwithcikinstB_copy[n he design, regardle nored. For paths bety clock constraints and To Noc mult with clk instgroo mult with clk inst	ass of the user-specified cloc ween a clock and its inversion other slack reports for sign- de Launch Clock duct[30] clk duct[31] clk duct[30] clk duct[31] clk	k periods. FMAX is compute ff analysis. K Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000 4.	For paths where i and failing edgess and failing edgess -0.063 -0.063 -0.062 -0.063 -0.062 -0	Data Delay 3.275 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.256 3.269 3.164 3.155 3.156 3.153 3.150 3.140 3.143 3.143 3.143 3.143 3.143 3.047	sti
 EDA Netlist Writer Flow Messages Flow Suppressed Message Table of Contents Flow Suppressed Message Flow Summary Flow Settings Flow Capability Flow Capability Flow Capability Flow Lapsed Time Flow Lapsed Time Flow Lapsed Time Flow Cos Summary Flow Lapsed Time Analysis & Synthesis Fiter Analysis & Synthesis Fiter Assembler TimeQuest Timing Analyzer Sourmary Parallel Compilation SDC File List Clocks Stow 1200mV & SC Mode Fiter Summary Hold Summary Recovery Summary Molinium Pulse Widtl Worst-Case Timing I Stow 1200mV OC Model Fast 1200mV OC Model Fast 200mV OC Model Fast 200mV OC Model Report TCCS Report TCCS Report TCCS 	S S S S S S S S S S S S S S S S S S S	This panel I clocks, incl Altera recover × 1200mV 8 Slack 0.697 0.744 0.783 0.793 0.794 0.799 0.802 0.813 0.814 0.818 0.896 0.898 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup: 'cik' From Node mutwithcikinstlB_copy[1] mutwithcikinstlB_copy[2] mutwithcikinstlB_copy[2] mutwithcikinstlB_copy[3] mutwithcikinstlB_copy[3] mutwithcikinstlB_copy[3] mutwithcikinstlB_copy[7	n he design, regardle nored. For paths bety clock constraints and To Noc mult with clk.instproc mult with clk.instproc	ses of the user-specified clock ween a clock and its inversion other slack reports for sign- de Launch Clock duct[30] ck duct[31] ck duct[30] ck duct[31] ck <t< td=""><td>k periods. FMAX is , FMAX is compute off analysis. Latch Clock clk clk clk clk clk clk clk c</td><td>Relationship 4.000 4.</td><td>For paths where i and failing edgess Clock Skew -0.063 -0.063 -0.062 -0.063 -0.062 -0.</td><td>be source and detars called along of a source and detars are scaled along of a source and source a</td><td>stii</td></t<>	k periods. FMAX is , FMAX is compute off analysis. Latch Clock clk clk clk clk clk clk clk c	Relationship 4.000 4.	For paths where i and failing edgess Clock Skew -0.063 -0.063 -0.062 -0.063 -0.062 -0.	be source and detars called along of a source and detars are scaled along of a source and source a	stii
 EDA Netlist Writer Flow Messages Flow Suppressed Message Flow Suppressed Message Flow Suppressed Message Flow Summary Flow Settings Flow Settings Flow Settings Flow Settings Flow Source Analysis & Synthesis Fitter Assembler Summary Parallel Complation SDC File List Clocks Sow 1200mV 85C Mode Frank Summary Recovery Summary Hoid Summary Recovery Summary Hoid Summary Hoid Summary Fast 1200mV 0C Model Fast 1200mV 0C Model Metastability Summa Slow 1200mV 0C Model Fast 1200mV 0C Model Multicomer Timing Analys Advanced V0 Timing Clock Transfers Report RSKM Unconstraimed Paths 	s Slow 1 2 3 4 5 6 7 8 9 10 11 2 3 4 5 6 7 8 9 10 11 12 13 14 5 16 17 18 19 20 21 22 23 24 25 26 27 28 9 30 31 4 5 5 6 7 8 9 9 10 11 2 2 3 4 5 5 6 7 8 9 9 10 11 12 13 14 5 16 10 11 12 13 14 5 16 17 10 11 12 13 14 5 16 17 18 19 20 21 22 23 24 25 26 27 28 20 21 21 21 21 21 21 21 21 21 21	This panel I clocks, incl Attera reco v 200m V 8 0.677 0.683 0.697 0.744 0.788 0.793 0.793 0.793 0.793 0.793 0.793 0.795 0.799 0.812 0.812 0.813 0.814 0.818 0.818 0.839 0.860 0.813 0.814 0.818 0.839 0.860 0.813 0.814 0.839 0.860 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906 0.906	reports FMAX for every clock uding generated clocks, are ig mmends that you always use ISC Model Setup; 'cik' From Node mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[1] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[2] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[3] mutwithcikinstB_copy[4] mutwithcikinstB_copy[3] mutwithcikinsB_copy[3] mutwithcikinstB_copy[3	n he design, regardle nored. For paths bety clock constraints and To Noc mult withclk instpro- mult withclk instpro-	ses of the user-specified cloc ween a clock and its inversion other slack reports for sign- le Launch Cloc uct[30] ck uuct[31] ck uuct[30] ck uuct[31] ck u	k periods. FMAX is compute ff analysis. Calk compute cik	Relationship 4.000 4.	Tor paths where is and failing edgess of a straight of the second st	Data Delay 3.275 3.269 3.256 3.269 3.269 3.261 3.171 3.164 3.153 3.156 3.158 3.150 3.140 3.143 3.134 3.134 3.134 3.134 3.047	stii

Table	of Contents 🛛 📮 🖥	Slow	1200mV (85C Model Hold: 'clk'							
	Flow Summary		Slack	From Node		To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
	Flow Settings	1	0.373	multwithclk:inst[A_copy[12]	multwithcli	cinst A_copy[11]	clk	clk	0.000	0.063	0.593
	Flow Non-Default Global Set	2	0.374	multwithclk:inst[A_copy[5]	multwithcl	cinst A_copy[4]	clk	clk	0.000	0.063	0.594
	Flow Flopped Time	3	0.374	multwithclk:inst[A_copy[8]	multwithcl	cinst[A_copy[7]	clk	clk	0.000	0.063	0.594
	Flow Clapsed Time	4	0.374	multwithclk:inst[A_copy[9]	multwithcl	cinst[A_copy[8]	clk	clk	0.000	0.063	0.594
1 🗄	Flow OS Summary	5	0.374	multwithclk:inst[A_copy[10]	multwithcl	cinst(A_copy[9]	clk	clk	0.000	0.063	0.594
	Flow Log	6	0.374	multwithclk:inst[A_copy[13]	multwithcl	cinst[A_copy[12]	clk	clk	0.000	0.063	0.594
>	Analysis & Synthesis	7	0.374	multwithclk:inst[A_copy[14]	multwithcl	cinst[A_copy[13]	clk	clk	0.000	0.063	0.594
>	Fitter	8	0.375	multwithclk:inst[A_copy[4]	multwithcli	cinst A_copy[3]	clk	clk	0.000	0.063	0.595
>	Assembler	9	0.391	multwithclk:inst[B_copy[28]	multwithcl	cinst B_copy[29]	clk	clk	0.000	0.063	0.611
V 1	TimeQuest Timing Analyzer	10	0.391	multwithclk:inst[B_copy[23]	multwithcl	cinst[B_copy[24]	clk	clk	0.000	0.063	0.611
	E Summary	11	0.392	multwithclk:inst[B_copy[5]	multwithcl	cinst B_copy[6]	clk	clk	0.000	0.063	0.612
	Parallel Compilation	12	0.393	multwithclk:inst[B_copy[29]	multwithcl	cinst B_copy[30]	clk	clk	0.000	0.063	0.613
		13	0.393	multwithclk:inst[B_copy[27]	multwithcl	cinst B_copy[28]	clk	clk	0.000	0.063	0.613
	SDC File List	14	0.393	multwithclk:inst[B_copy[26]	multwithcl	cinst[B_copy[27]	clk	clk	0.000	0.063	0.613
	Clocks	15	0.393	multwithclk:inst[B_copy[25]	multwithcli	cinst[B_copy[26]	clk	clk	0.000	0.063	0.613
~	E Slow 1200mV 85C Mode	16	0.393	multwithclk:inst[B_copy[19]	multwithcli	cinst[B_copy[20]	clk	clk	0.000	0.063	0.613
	Fmax Summary	17	0.393	multwithclk:inst[B_copy[18]	multwithcli	cinst[B_copy[19]	clk	clk	0.000	0.063	0.613
	Timing Closure Recc	18	0.393	multwithclk:inst[B_copy[6]	multwithcli	cinst[B_copy[7]	clk	clk	0.000	0.063	0.613
	Setup Summary	19	0.394	multwithclk:inst[B_copy[22]	multwithcl	cinst(B_copy[23]	clk	clk	0.000	0.063	0.614
	Hold Summary	20	0.444	multwithclk:inst[acc[4]	multwithcl	cinst[A copy[1]	clk	clk	0.000	0.063	0.664
	Recovery Summary	21	0.445	multwithclk:inst[acc[4]	multwithcli	cinst[A_copy[2]	clk	clk	0.000	0.063	0.665
	Demoval Summary	22	0.447	multwithclk:inst[acc[4]	multwithcli	cinst[acc[4]	clk	clk	0.000	0.063	0.667
	E Removal Summary	23	0.453	multwithclk:inst[acc[4]	multwithcli	cinst[A_copy[0]	clk	clk	0.000	0.063	0.673
	Minimum Pulse Width	24	0.453	multwithclk:inst[acc[4]	multwithcli	cinst[acc[0]	clk	clk	0.000	0.063	0.673
	Y 📂 Worst-Case Timing I	25	0.454	multwithclk:inst[acc[4]	multwithcl	cinst[acc[3]	clk	clk	0.000	0.063	0.674
	Setup: 'clk'	26	0.455	multwithclk:inst[acc[4]	multwithcl	cinst[acc[2]	clk	clk	0.000	0.063	0.675
	Hold: 'clk'	27	0.456	multwithclk:instlacc[4]	multwithcl	cinst[acc[1]	clk	clk	0.000	0.063	0.676
	Metastability Summa	28	0.479	multwithclk:instA_copv[1]	multwithcl	cinstlA copv[0]	clk	clk	0.000	0.063	0.699
<pre>></pre>	Slow 1200mV 0C Model	29	0.498	multwithclk:instlB_copy[16]	multwithcl	cinstB copv[17]	clk	clk	0.000	0.063	0.718
	Fast 1200mV/ 0C Model	30	0.514	multwithclk:instlB_copy[24]	multwithcl	cinstB copv[25]	clk	clk	0.000	0.063	0.734
	Multisenses Timine Apple	31	0.514	multwithclk:instlB_copy[20]	multwithcl	cinstB copy[21]	clk	clk	0.000	0.063	0.734
	mulucorner finning Analy	32	0.550	multwithclk:instlA_copy[11]	multwithcl	cinstlA_copy[10]	clk	clk	0.000	0.063	0 770
	Advanced VO Timing	33	0.551	multwithclk:instA_copy[6]	multwithcl	cinstlA_copv[5]	clk	clk	0.000	0.063	0.771
>	Clock Transfers	34	0.551	multwithclk:instIA_copv[7]	multwithcl	cinstlA_copv[6]	clk	clk	0.000	0.063	0.771
	Report TCCS	35	0.551	multwithclk:instlA_copy[15]	multwithcl	cinstlA_copv[14]	clk	clk	0.000	0.063	0.771
	Report RSKM	36	0.568	multwithclk:instloroduct[15]	multwithcl	cinstloroduct[15]	clk	clk	0.000	0.063	0 788
>	Unconstrained Paths	37	0.568	multwithclk:instlproduct[13]	multwithel	cinstloroduct[13]	clk	clk	0.000	0.063	0 788
	 Messages 	38	0.568	multwithclk instloroduct[6]	multwithel	cinstloroduct[6]	clk	clk	0.000	0.063	0 788
5	EDA Netlist Writer	39	0.568	multwithclk:inst[product[3]	multwithel	cinstloroduct[3]	clk	clk	0.000	0.063	0.788
	V	40	0.560	multwithelk-instloreduct[29]	multwithel	cinetioroduct(20)	clk	clk	0.000	0.063	0.789
	 Flow Non-Default Globa Flow Elapsed Time Flow CS Summary Flow Log Analysis & Synthesis Fitter Assembler TimeQuest Timing Analy EDA Netlist Writer PowerPlay Power Anal Summary Settings Indeterminate Toggi Operating Condition Thermal Power Diss Thermal Power Diss Core Dynamic Ther Current Drawn fror Confidence Metric I Signal Activities Messages Flow Messages Flow Messages 	al Settin //zer yzer le Rates is Used sipation sipation mal Pov n Volta; Details	Proventies of the second secon	ion Name evel Entity Name y se ar Models Thermal Power Dissipation Dynamic Thermal Power Dissip Static Thermal Power Dissip nermal Power Dissipation er Estimation Confidence	sipation ation	CPU multitesting3 Cyclone IV E EP4CE6F17C6 Final 97.79 mW 6.19 mW 42.86 mW 48.74 mW Low: user provid	ed insufficient to	ggle rate data			

Figure analysis Radix 2

The ROM implementation had a slightly higher Fmax than using RAM when testing the unpipelined versions as seen below.

Instruction ROM implementation

	Fmax	Restricted Fmax	Clock Name	Note					
1	69.42 MHz	69.42 MHz	clk						
-	DI D								
Pow	erPlay Power	r Analyzer Summary							
Pow	erPlay Power /	Analyzer Status	Successful - T	Thu Jun 11 17:35:15 2020					
Qua	rtus Prime Vers	sion	16.0.0 Build 21	1 04/27/2016 SJ Lite Edition					
Revi	ision Name		CPU	CPU					
Тор-	level Entity Nar	ne	CPU_block_tim	CPU_block_timing					
Fam	ily		Cyclone IV E						
Devi	ice		EP4CE6E22C6	EP4CE6E22C6					
Pow	er Models		Final	Final					
Tota	I Thermal Powe	er Dissipation	129.61 mW	129.61 mW					
Core	e Dynamic Ther	mal Power Dissipation	60.57 mW	60.57 mW					
Core	e Static Therma	Power Dissipation	43.01 mW						
VO T	hermal Power	Dissipation	26.03 mW						
Pow	er Estimation C	onfidence	Low: user pro	Low: user provided insufficient toggle rate data					

Instruction RAM implementation:

1	Slow	1200mV 85C M	odel Fmax Summary		
		Fmax	Restricted Fmax	Clock Name	Note
	1	66.12 MHz	66.12 MHz	clk	

PowerPlay Power Analyzer Summary	
PowerPlay Power Analyzer Status	Successful - Thu Jun 11 18:06:48 2020
Quartus Prime Version	16.0.0 Build 211 04/27/2016 SJ Lite Edition
Revision Name	CPU
Top-level Entity Name	CPU_block_timing
Family	Cyclone IV E
Device	EP4CE6E22C6
Power Models	Final
Total Thermal Power Dissipation	133.94 mW
Core Dynamic Thermal Power Dissipation	61.56 mW
Core Static Thermal Power Dissipation	43.01 mW
VO Thermal Power Dissipation	29.36 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

Appendix 10: Fibonacci results

Fib(2) = 2

68 | Page

13

H

1_

010

0001 1 X or X or

14																
-	CLOCK	80														3 HL
	fetch	B 1														
	exec1	B 0														
-	> daddr	H 000		000		XO	01	000	001	X 000	X FFD X 001 X 0	IO X FFE	X 001 X 002)	FFF	012 0	101
out							^			^			0000 V 00			0000
-	/ 0004	HOUR											0001 00			
-	> instr_ad.	. н ооо	X 006	X 007	X 008	X 009	X 00A	<u>X 008 X</u>	020 X 0	21 <u>X</u> (012 X	013	X 01	4 <u>X</u>	015	X_
5	> instr	H 0000	3010	0000	4C48 X (30	i10	0020 4000	X 302A	4006	X 9017 X	900F	X	9037	Х 34	015
25	> portladd		2 X	0 X	1 X	• X :	2 χ	0	X 5	X 0	X 7 X 2	X 7	X 1)	7 X	6 X	2
010	Mi aal	8.1														
-	10,000															
	pop_sel	80														
	wenout	B 0														
010	wen	80														
out	CRITICIAN	8.0														
-	carryen															
-	carrysta	B 0														
	wrte_n	B 0														
8	> write_n	B 0000	X 0010	X	0001	0000	0010	X 0100 X	0000 X 01	101 X 0	1010	X 0001	(1001)	0110	1110	0010
out	write o	8.0										_				
-	> rdin	H 0000		0000	0000 X	0001		0020 1 0000	<u>t 0001</u>	10000	X PPPE X 0000		<u>1 0002</u>	0000	0003 1 0000 1	0001
5	> dmemin	H 0000	0000	X 000C X	0001 0000	X 0001 X 0000	0001	X 0020	X 0000 X 0001	X 0020	X FFFD X 0001 X 00	00 X FFFE	X 0001 X 0002)	FIFT X	0012 0003 00	000 0
5	> q1	H 0000	0000	X 000C X	0001 X 000C	X 0001 X 0000	0001	X 0020	X 0000 X 0001	0020	X FFFD X 0001 X 00	00 X FFFE	X 0001 X 0002)	FFFF X	0012 X 0003 X 00	000 X 0
-	> a2	H 0000	0000	Ý	000C X	0001		X 0020	0001	X 0012	X FFFD X	FFFE	Y FF	F Y	0000 00	001
94.0		11 0000	0000		0000	V	0.1									
-	> R0out	H 0000	0000		000C	X0	801	×					00	10		
5	> R1out	H 0000					0001									
*	> R2out	H 0000					X		0001		X			0000		X
944	> R3out	H 0000														
-																
-	> Heout	H 0000														
۳	> RSout	H 0000							X							
5	> R6out	H 0000						0012							X	
948	> R7out	H 0000			FFFD						Ý	FFFE	1	r Y	0	000
					_			-					=			
•	pc_cnt_e	n 00														
out	**	0.0														
_														1.00		
			1.2 us	1.24 us	1.28 us	1.32 us	1.36 us	1.4 us	1.44 us 1.48	us 1.52	2 us 1.56 us	1.6 us	1.64 us	1.66 68	1.72 us 1.76	us
	Name	Value at 0 ps	1.2 us	1.24 us	1.28 us	1.32 us	1.36 us	1.4 us	1.44 us 1.48	us 1.52	2 us 1.56 us	1.6 us	1.64 us	1.66 88	1.72 us 1.76	us
	Name	Value at 0 ps	1.2 us	1.24 us	1.28 us	1.32 us	1.36 us	1.4 us	1.44 us 1.48	us 1.52	2 us 1.56 us	1.6 us	1.64 us		1.72 us 1.76	us ;
in .	Name CLOCK	Value at 0 ps B 0	1.2 us	1.24 us	1.28 us	1.32 us	1.36 us	1.4 us	144 us 148	us 1.52	2 us 1.56 us	1.6 us	1.64 us		1.72 us 1.76	us
is	Name CLOCK fetch	Value at 0 ps B 0 B 1	12us	1.24 us	1.28 us	1.32 us	1.36 us		144us 148		2 us 1.56 us	1.6 va			172 US 1.76	
· · · · · · · · · · · · · · · · · · ·	Name CLOCK fetch exec1	Value at 0 ps B 0 B 1 B 0	12us	1.24 us	1.28 vs	1.32 us	1.36 US	1.4 us	144 us 148		2 us 1.56 us	1.6 µ3	1.64 us		1.72 us 1.76	
10 10 10 10 10 10 10 10 10 10 10 10 10 1	Name CLOCK fetch > daddr	Value at 0 ps B 0 B 1 B 0 H 000		1.24 us	1.28 us	1.12 us	1.36 us	1.4 us	144 us 148	us 1.52	2 us 1.56 us	1.6 us	1.64 us		1.72 us 1.76	
1 10 15 15 Au	Name CLOCK fetch exec1 > daddr	Value at 0 ps B 0 B 1 B 0 H 000		1.24 us	1.28 us	1.32 us	1.36 US	1.4 us	144 us 148		2 us 1.56 us	16 us	1.64 us		1.72 us 1.76	
	Name CLOCK fetch exec1 > daddr > dout	Value at 0 ps B 0 B 1 B 0 H 000 H 0000	12 us	1.24 us	1.22 us	1.32 us	1.36 us	1.4 us	144 us 145		2 us 1.56 us	16us	1.64 us	1.66 US	1.72 us 1.76	
10 10 10 10 10 10	Name CLOCK fetch exec1 > daddr > dout > instr_ad	Value at 0 ps B 0 B 1 B 0 H 000 H 0000 H 0000	12 us	1.24 us	1.28 us	1.32 us	000 0000 0000	1.4 us	1.44 us 1.48	us 1.52	2 us 1.36 us	1.6 us	1.64 us	1.65 us	1.72 us 1.76 000 000 000	US
al 19 10 10 10 10 10	Name CLOCK fetch exec1 > daddr > dout > instr_ad > instr	Value at 0 ps B 0 B 1 B 0 H 000 H 0000 H 0000 H 0000	1.2 us	124 us	1.28 us	1.32 us	1.36 us	1.4 us 7.7 FFE X 01C X 0002 X 01B 6400 X 50	1.44 us 1.48 005	us 1.52	2 us 1.56 us 0000 0000 0000 007 0000 007	1.6 us	1.64 us	1 00 US	172 us 1.75 000 000 000 000 000 000 000 000 000 0	US (00) (020)
al 19 10 10 10 10 10 10	Name CLOCK fetch > daddr > daud > instr_ad > portladd	Value at 0 ps B 0 B 1 B 0 H 000 H 0000 H 0000 H 0000 V 0 V 0	12 us	1.24 us	1.28 us	1.32 us	1.36 us	1 4 us	1.4 us 1.48 005 005 004 X 006 8 X 0	us 1.52	2 us 1.56 us	1.6 us	1.64 us	1.09 US 001 X 00A 3010 X 00 2 X	172 us 179	US
	Name CLOCK fetch exec1 double sec1 double instr_ad instr portladd tdl_sel	Value at 0 ps B 0 B 1 B 0 H 000 H 0000 H 0000 H 0000 F U 0 B 1	12 us	124 us	1.28 vs X FFD 0092 X X 018 A039 X 7	132 us	1.36 us	1 4 us	1.44 us 1.45	us 1.52	2 us 1.56 us 0000 0006 X 007 X 000c X 0 000 0006 X 007	1.6 is	1.64 us		172 us 179	US
	Name CLOCK fetch exec1 > daddr > dout > instr_ad > portladd Id_sel	Value at 0 ps 8 0 8 1 8 0 4 000 4 0000 4 0000 4 0000 6 1 0 0 8 0 8 0	12 us 01 055 2	124 us	1.28 us	1.32 us	1.36 US 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	1 4 us 7 FFE 0 01C 0002 000	1.44 us 1.45 005 005 004 84 0000 8 000 8 004	us 1.52	2 us 1.56 us 000 000 000 000 000 000 000 0	1.6 is	1.64 vs	1 66 UK	172 us 1.79 000 000 000 000 000 000 000 000 000 0	020 X 50
	Name CLOCK fetch kec1 daddr dout binstr_ad instr pop_sel	Value at 0 ps B 0 B 1 H 000 H 000 H 000 U 0 B 1 B 0 H 000 H 0	12 us	124 IS	1.28 vs 2.28 vs 2.2	1.32 us	1.36 us 000 0000 0000 0000 0000 0000 0000 00	14 is	144 us 149	us 1.52	Lus 150 us 000 000 000 000 000 000 000 000 000 0	1.6 is	1.64 is 0000 0000 0000 0001 0000 0001 0000	1 ee us 001 X 004 X 004 X 004 X 004 X 005 X	172 us 176 000 000 000 000 000 000 000 000 000 0	US
	Name CLOCK fetch kec1 daddr dout binstr_ad instr_ad portladd idi_sel pop_sel	Value at 0 ps B 0 B 1 B 0 H 000 H 000 H 000 H 000 H 000 B 1 B 1 B 1 B 0 B 0 B 0 B 0	12 us	124 is	1.28 us	1.32 us	1.36 US 0000 0000 0000 0000 0000 0000 0000 0	14 ss	144 uh 149	us 1.52	Lus 150 us 000 X 007 X 000C X 000 X 07 X 0 X	1.6 is		1 min and a min	172 vs 178	us
10 10 10 10 10 10 10 10 10 10 10 10 10 1	Name CLOCK fetch exec1 > daddr > daut > instr_ad > instr_ad pop_sel wenout	Value at 0 pps 0 ps 0	12 es	124 IS	1.28 us X FFD 0.092 X X 018 A.059 X 7	1.50 us	1.38 us	14 ss 776 X 01C X 076 X 01C X 0002 X 078 B 1 X 002 1 X 002	144 uh 149	us 1.52	Lus 159 us 000 000 000 000 000 000 000 0	1.6 is		1 ing us	0000 X 00000 X 00000 X 0000 X	us
	Name CLOCK fetch exec1 daddr dout instr_ad instr pop_sel pop_sel wenout wen	Value at 0 ps 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1.2 es	124 is	1.28 us	1.5; us			144 uh 149	us 1.52	2 un 150 un 000 000 X 007 X 000C X X 0 X 0 X 0 X 0 X 0 X 0 X 0 X	1.6 is 				
	Name CLOCK fetch exec1 > daddr > dout > instr_ad > instr > portladd digsel pop_sel wenout wen carryen	Value at Notes at 9 0 B 0 B 0 B 0 B 0 B 0 H 000 H	12 es	124 is	1.28 us	1.5 us	13 us 000 0000 0000 0000 0000 0000 0000 00	14 is 7 FFE 01C 0002 000 0002 0000 0000 0000 0	144 uh 149	us 1.52	2 us 150 us 000 000 000 000 000 000 000 0				000 000 000 000 000 000 000 000 000 00	
3 45 45 46 46 46 48 48 48 48 48 48 48 48 48 48 48 48 48	Name CLOCK fetch exec1 > daddr > dout > instr_ad > instr_ad > portladd di_sel pop_sel wenout wen carrysia	Value at 0 pp B 0 B 1 B 0 B 1 B 0 H 0000 H 0000 <th>12 es</th> <th>124 is</th> <th>1.28 us</th> <th>1.32 us</th> <th></th> <th></th> <th>144 va 149</th> <th>us 153</th> <th>2 un 150 un 150</th> <th></th> <th></th> <th></th> <th></th> <th></th>	12 es	124 is	1.28 us	1.32 us			144 va 149	us 153	2 un 150					
1 42 45 45 46 46 46 46 46 46 46 46 46 46 46 46 46	Name CLOCK fetch exec1 daddr dout instr_ad portado ff_sel pop_sel wenout wenout carryen carryen	Value at Notes at S S B B B B B B H H000 H H H H H H H H H H H H H H H H H H H <th>12 es</th> <th>124 is</th> <th>1.28 us</th> <th>1.50 us</th> <th></th> <th></th> <th>144 m 149</th> <th>us 157</th> <th>2 tei 150 tei 000 X 000 X 000 X 000 X 000 X 00 X 0 X</th> <th></th> <th></th> <th></th> <th></th> <th>us</th>	12 es	124 is	1.28 us	1.50 us			144 m 149	us 157	2 tei 150 tei 000 X 000 X 000 X 000 X 000 X 00 X 0 X					us
	Name CLOCK fetch secol dadd dadd base base base secol contract secol contract secol contract secol contract con	Value at 0 pg B 0 B 1 B 0 B 1000 H 000 H 00	13 es		1.28 us 7 0002 X 018 A039 X 019 0111	130 m	1 39 US 0000 0000 0000 0 01A 0 0002 0 X 0 X		144 vs. 149	us 157 005 X 2015 X 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1				102108 001 X 00A 2 2 X 00A X 00A X 00A X 00A X 00A X 00A X 00A		us
	Name CLOCK CLOCK Second	Value at 0 p 0 0 0 1 0 0 0 4 000 - 1000 - 1000 - 0	12,04		128 to X 770 002 X X 018 A595 X 7 0111	130 m			144 xx 149	965 X	2 un 150 un 000 X 000 X 000 X 000 X 000 X 000 X 000 X 000 X 000 X 000 X					us
	Name CLOCK Etch CLOCK Etch CLOCK Etch CLOCK Etch CLOCK Etch Etch Etch Etch Etch Etch Etch Etch	Value at 0 ps b 0 b 1 b 0 b 1 b 0			128 is				144 va 149	us 157	ten 159 es 000 000 000 000 000 000 000 0					us
	Name CLOCK fetch exec1 > daddr > instr_ad > inst	Value at 0 ps B 0 B 1 B 0 H 000			128 m	13/4 7/77C				a 15	150 us 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 0000 0000 0000 0000 0000 0000					
	Name CLOCK fetch accol daddr daddr daddr binstr_ad pop_sel pop_sel wenout dangr carrysia carrysia write_n. write_n. rdin daddr dadd	Value at 0 p 0 p 1 p			138 m 7 m 6002 7 m 6011 6011 902 7 m 6011 901 901 901 901 901 901 901				144 vs. 149		1 1 52 us 0 0 0 <th></th> <th></th> <th></th> <th></th> <th>us</th>					us
	Name CLOCK field f	Value at 0 p 0 0 p 1 0 p 1 0 p 0 1 p 0000			128 m X 770 0062 X 610 A59 X 770 A59 X 770 A50 A50 A50 A50 A50 A50 A50 A5	139 a 7 FFC (7 OFF) 7 OFF)			144 m 149 144 m 149 149 m 149 149 m 149 149 m 149 149 m 149 149 m 149 m 149 149 m 149 149 m 149 m 149 m 149 149 m 149 m 149	u 15	1 1 5% us 1 5% us 1 000 000 0 000 0 0 000 0 0 0010 0 0 0010 0 0 0010 0 0 0010 0 0 0010 0 0 0000 0000 0 0000 0000 0000					us X 000 X 000
	Name CLOCK C	Value at 0 p 0 p 1 p <			138 m X FFD 0002 X 519 A59 X FFD 011 011 001 011 002 X FFD 011 011 011 011 011 011 011 01	139 m 139 m X rrc X 99 A65A X 99 A65A X 77C 001 001 001 001			144 m 149 064 000 064 000 064 000 064 000 000 000 000 000 000 000 000	us 15	1 1 5 w 1 5 w 0 000 000 0 0 000 0 0 0 000 0 0 0					
4 10 10 10 10 10 10 10 10 10 10 10 10 10	Name CLOCK C	Value at 0 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 00 0 00 0 00 0 00 0 0	1244 1244		128 m 7 770 0002 X 100 A509 X 100 X 1	139 m 7 m 7 m 7 m 7 m 7 m 7 m 7 m 7	1 39 us				1 1 5% us 1 5% us 1 0 000 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0					us
	Name CLOCK CLOCK CLOCK Fetch F	Value at 0 p D <t< th=""><th></th><th></th><th>138 m 7 FFD 0602 X 0602 X 0602 X 0602 X</th><th>139 a 139 a 13</th><th></th><th></th><th>144 xh 149</th><th></th><th>1 50 us 000 000 000 000 000 000 000 000 000 000 000 0000 000 000 000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000</th><th></th><th></th><th>trap ta trap ta t</th><th>1/2/16 178</th><th>us 220 X 220 X 200 X 300 X 5 300 X 5 X 5 X 5 300 X 5 X 5 X 5 X 5 X 5 X 5 X 5 X 5</th></t<>			138 m 7 FFD 0602 X 0602 X 0602 X 0602 X	139 a 139 a 13			144 xh 149		1 50 us 000 000 000 000 000 000 000 000 000 000 000 0000 000 000 000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000			trap ta t	1/2/16 178	us 220 X 220 X 200 X 300 X 5 300 X 5 X 5 X 5 300 X 5 X 5 X 5 X 5 X 5 X 5 X 5 X 5
	Name CLOCK C	Uses at 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 00 0 0 <	12,44 12,44 14 14 14 14 14 14 14 14 14		128 m	139 a 139 a 13			144 m 149		un 150 un 000 000 000 000 000 000 000 000 000 000 0000 000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000					us 200 200 200 200 200 200 200 20
	Name CLOCK etc.0 etc.0 etc.0 etc.0 etc.0 b etc.0 b etc.0 b etc.0 b etc.0 b etc.0 etc.0 <tr< th=""><th>Numeral D<!--</th--><th></th><th>13248</th><th>13 as 2 7 770 002 X 7 011 011 011 011 011 011 011</th><th>139 a X FFC X 699 A69A X 699 A69A X 7FC X 699 X 7FC X 699 X 7FC X 699 X 7FC X 699 X 699 X 7FC X 699 X 6</th><th>1 2 4 44 000 000 000 000 000 000 000</th><th></th><th>144 vs. 149</th><th>us 1.52 us 1.52 us 0.05 vs 2 us 3 us 3 us 3 us 3 us 3 us 3 </th><th>1 1 50 us 000 000 000 000 000 000 000 0 0 000 0 0 000 0 0</th><th></th><th></th><th>1 1</th><th>112/06 178</th><th>us X 000 X 000 X 000 X 000 X 000 X 000 X 000</th></th></tr<>	Numeral D </th <th></th> <th>13248</th> <th>13 as 2 7 770 002 X 7 011 011 011 011 011 011 011</th> <th>139 a X FFC X 699 A69A X 699 A69A X 7FC X 699 X 7FC X 699 X 7FC X 699 X 7FC X 699 X 699 X 7FC X 699 X 6</th> <th>1 2 4 44 000 000 000 000 000 000 000</th> <th></th> <th>144 vs. 149</th> <th>us 1.52 us 1.52 us 0.05 vs 2 us 3 us 3 us 3 us 3 us 3 us 3 </th> <th>1 1 50 us 000 000 000 000 000 000 000 0 0 000 0 0 000 0 0</th> <th></th> <th></th> <th>1 1</th> <th>112/06 178</th> <th>us X 000 X 000 X 000 X 000 X 000 X 000 X 000</th>		13248	13 as 2 7 770 002 X 7 011 011 011 011 011 011 011	139 a X FFC X 699 A69A X 699 A69A X 7FC X 699 X 7FC X 699 X 7FC X 699 X 7FC X 699 X 699 X 7FC X 699 X 6	1 2 4 44 000 000 000 000 000 000 000		144 vs. 149	us 1.52 us 1.52 us 0.05 vs 2 us 3 us 3 us 3 us 3 us 3 us 3	1 1 50 us 000 000 000 000 000 000 000 0 0 000 0 0 000 0 0			1 1	112/06 178	us X 000 X 000 X 000 X 000 X 000 X 000 X 000
	Name CLOCK Education Internation Internatinterim	Viewer at organization 0 0 0 0 1 0 4 0000 4 0000 4 0000 4 000 4 000 4 000 4 000 5 0 6 0 7 0 8 0 1	1244 1244 1045	12/14 (777 X 772 6 X 617 A025 X 6 A025 X 7 FFF 6 A025 X 7 FFF 7 FFFF 7 FFFFF 7 FFFFF 7 FFFFF 7 FFFFF 7 FFFFF 7 FFFF 7 FFFFF 7 FFFFF 7 FFFFF	128 H	139 H			144 m 149		un 159 un 159 un 000 000 000 000 000 000 0 000 0 0000 0 0000 0 0000 0 0000 0 0000 0 0000 0 0000 0 0000 0 0000 0 0000 0 0000 0					
	Name CLOCK International state	Value at 0 p 0 0 0 0 0 0 0 0 0 0 1 0			13 as 2 7 770 002 X 7 011 011 011 011 011 011 011	139 a 139 a 13	1 3 4 44 00000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0		144 vs. 149	us 1.52 us 1.52 us 0.05 vs 2.01 vs 2.01 vs 2.01 vs 2.01 vs 2.01 vs 2.01 vs 2.000 vs 0.000 00000 0.000	1 1 50 us 000 000 000 000 0 0 000 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			1 1		
	Name CLOCK C	Vision att Open Service D D D D D D H D000 H D000 H D000 H D000 H D000 H D00 H D00 D D D			128 m	139 H			144 m 149		1 1 5% uk 1 5% uk 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0					90 (1000)
	Name CLOCK 64ch exec1 2 3 4 4 5 6 2 6 2 6 2 6 6 7 6 7	Value at organization 0 0 0 0 0 0 0 0 0 0 1 0		17248	13 as 2 7 770 0002 X 7 011 011 011 011 011 011 011	139 a 139 a 13	1 3 4 44 00000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0		144 xa 149	us 1.52 us 1.52 us 0.05 x 2 us 2	1 1 5% us 000 000 000 000 0 0 000 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			1 1		
	Name CLOCK C	Unite at one Second Secon			128 m	139 m				a 15	1 1 5 u 1 5 0 0 006 0 0 0 006 0 0 0 0070 X 0 0 0000 X 000C X 000 X 000C X					
	Name CLOCK etcl fetch eccl statut j add j j add j j add	Union 0 0 0 0 0 0 1 0 <tr td=""> 1 0<</tr>		12/14 	128 m 7 770 6002 7 7 6011 6111 0111 0111 0111 0111 0111 0111 0111 0111 002 002	139 a 139 a X rrc (X 99 A0JA (A0JA (X 99 A0JA (A0JA (A) A0JA (A) A0JA (A) A0JA (A) A0JA (A) A0JA (A) A0 A0 A0 A0 A0 A0 A0 A0 A0 A0			144 xa 149	005 005 005 005 000	1 1 50 µk 0000 0000 0000 0000 0 0 0000 0 0 0000 0 0 0000 0 0 0000 0 0 0000 0 0 0000 0 0000 0000 0 0000 0000 0 0000	18,es				
	Name CLOCK C	Under all D D <th></th> <th></th> <th>128 m 7 770 0002 X 019 A559 X 019 A559 X 019 0002 X 770 0002 X 7770 0002 X 777770 0002 X 7770 0002 X 7770 0000</th> <th>130 m</th> <th></th> <th></th> <th>144 m 149</th> <th></th> <th>1 1 5 1 5 1 5 1</th> <th>18 m</th> <th></th> <th></th> <th></th> <th></th>			128 m 7 770 0002 X 019 A559 X 019 A559 X 019 0002 X 770 0002 X 7770 0002 X 777770 0002 X 7770 0002 X 7770 0000	130 m			144 m 149		1 1 5 1 5 1 5 1	18 m				

	idi_sel	B 1																															
	pop_sel	80																															
-	wenout	B 0							1				_												1		1		1		1 1		
94	wen	80		-		-					-							<u> </u>					_	_	1		1	<u> </u>	1	H	1		Т
941	carryen	80																						_		_							
045	carrysta	80																															T
01/1	write o	80				-			-				_						- 1						-		1	_	1				T
				0000	-				\downarrow	-						_			\downarrow			-		_			-	-					ŧ
• >	wrte_n	8 0000		0000	-	+			\wedge				_^_	0010	+++		1001	+ + +	\wedge			0111			\wedge				\uparrow			+	÷
	wrte_n	80		-	-		-				+ +						+ +													±	\mapsto		+
* > 1	rdin	H 0000	poorx		0002			0003	X 0005	×		0000		Ľ.	0000	× ·	0000	X FFFF	X	FFFE	_X_	FFFD	X FFFC	X	0001		x 0000	0012	X_0005	<u> </u>	000	.0	+
* > 1	dmernin	H 0000	0000	X 0002	2 X 0	000 X	0002	0000	X 0003	X 0002	X	_	0000		X 000	× –	0002	X (0003	_X_	0002	_X_	0000	X 000C	X 0001	X 0002	(0001)	0003	0012	0001	×	0000	+
* > 1	q1	H 0000	0000	X 0002	2 X 0	000 X	0002	0000	X 0003	0002	х_		0000		X 000	X	0002	X 0000	Х	FFFF		FFFE	X FFFD	X 000C	X 0001	X 0002	0001	0003	0012	0001	×	0000	Ŧ
* > 1	q2	H 0000	00	00		0002		0	000	0002	\mathbf{X}		0	100		X	000C	X	003		0002		0000	X	002	X	101	0	200	0001		0	000
5 > 1	R0out	H 0000	0000	X			00	02			X		0000		X					000C					X		00	01			X	0000	
5 > 1	R1out	H 0000		0000													0002																
5 > 1	R2out	H 0000																				0000											t
5 > 1	R3out	H 0000																															T
* > 1	R4out	H 0000																															Ŧ
	RSout	H 0000																							0000							_	Ŧ
	REput	H 0000			000	0			Y-										0003									<u> </u>	x		=		Ŧ
94	R7out	H 0000						F			0000								V	FFFF	V	FFFF	Y					F			===		Ŧ
out					-	-					- 200		_						1					-			1			=			+
-	pc_ont_en	0 0			_				-	- 1	-	_		-					-	_				_	-	-		=			\square		+

		,																																			
2.1	15 us																																				
	Name	Value at 0 ps	0 ps		40	0 ns		80.0 ns		120	0 ns	1	160,0 ns		200,0	ns	240	0 ms	280	.0 ns	x	10,0 ns	360	0,0 ns	400	0 ns	4	40.0 ns		480.0 ns	•	520.0 n	5	560.0	/ns	600.0	ý ns
	CLOCK	80	ľ			1 1	_					iг	-							1 6				1 -			1 1	11									
10	fetch	81	F	11		F		H	7 T	H	=		F		H	F 1	H	=		H		F		F				F			i 1		4 F	H	= 1	H	= 1
1	exec1	80			=	l F		1	-	=	F	Ħ	- F		i		=	F	F	Ē		F	-		F	F	F				F	=	-	-	-	=	-
5	daddr	H 000	F		000	Ξx	0	02	X 003	X 00	5 X			-		000					mr X	FFE		FED	X	rc X	000		001	000	X 012	005	X-				
s ,	dout	H 0000	F																									T	Ŧ		00	00			_		_
	instrad	H 000		000	X	001		X	002	X	004	X	00	6	X	006	X	007	X	000	X	000	X	ODE	X	OOF	X	01	0)		011	X	004	X	005	$\overline{\mathbf{x}}$	006
4.5	instr	H 0000	1000	X	0002	X	3	08	X	8004	-X-	0000	, χ	3	010	X	0000	-x-	4048	X	A03E	X	A039	X	A03A	X	0001		64	08	X	8004	X	0000	-x-	3010	- <u>γ</u>
5	port1addr	U O	F	-	0	Ξx		1	X	6	-	0	Ξx	-	2	X	0	÷	1	Ξx			7			Ex-	0	=			X	6	X	0		2	=
1	idi_sel	81	F			F												_										Ŧ					-		=		
1	pop_sel	80	1																																		
1	wenout	B 0	IΓ		_	Ē		1					1		1			1										1								-	
4	wen	80			_		-	1		_				-	1			1													-	-		-			
4	carryen	80	1E																																		
4	carrysta	80																																			
4	write_n	80	1					1							1													1									
5.	wrte_n	B 0000	1E		000	0		X G	0001	$\overline{\mathbf{x}}$		0000	,		χ	0010			0001		X			0111			Τx	000	10)	0	001	X		0000			0010
	wrte_n	B 0																																			
5 >	rdin	H 0000	1000	X		000	2		0003	X 00	05 X		000	00		X	000C	X	0000		нит Х	FFFE	X	FFFD	X FF	FC X		0001		0000	X 0012	X 0005	X		0000		_X
5 >	dmemin	H 0000		0000	X.	002 🗙	0000	0002	X 0000	X 00	03 X 0	102 X		0	000		X 00	χ	0002		0003		0002		0000	X •	ooc X	0001	0002	0001	X 0003	0012	0001	\mathbf{x}		0000	
5	q1	H 0000		0000	X	002 X	0000	0002	X 0000	X 00	03 X 0	102 X		0	000		X 00	x x	0002		0000 X	FFFF		FFFE) FF	FD X O	00C X	0001	0002	0001	X 0003	0012	X 0001	\mathbf{x}		0000	
5 >	q2	H 0000			1000		0	02	X	0000		902 X			0000				0000		0003		0002		0000		0002		00	01	X	0000	0001	Σ	_	000	50
5	R0out	H 0000		0000				(0002					0	000							0000								0	001			Σ	_	0000	
5	R1out	H 0000			000	0													0002																		
5 >	R2out	H 0000																						000	0										_		
5 >	R3out	H 0000																															_		_		
5	R4out	H 0000																																	_		
5 >	RSout	H 0000																									0	000									_
5 >	REout	H 0000				0	1000			\mathbf{X}											0003											X					
5	R7out	H 0000										0	000									FFFF		FFFE													
1	pc_ont_en	B 0						1							1																			-		~	

600,0 ms 640,0 ms 600,0 ms 720,0 ms 760,0 ms 600,0 ms 640,0 ms 660,0 ms 920,0 ms 960,0 ms 1.0,0 ms 1.6,4 ms 1.6,4 ms 1.6,4 ms 1.72 ms 1.15,4 ms 1.

107 cycles

Value at 0 ps Name

	Name	Value at	1.64 us 1.68 us	1.72 us 1.76 u	is 1.8 us	1.84 us	1.88 us	1.92 us 1.9	Kius 2.0 us	2.04 us	2.08 us 2.12	us 2.16 us	2.2 us 2.24 ut
in	CLOCK	80											
out	fetch	81											
015	exec1	80				=	= =			=			
94	daddr	H 000	Y 001 Y	000	Y 001 Y	000 Y FFD	Y 001 Y	FFE Y O	100 Y 002 Y FFF	V 01C V 004	Y 002 Y 00	3 Y 002 Y	
-	dout	H 0000	0000				Y 0001 Y 000	0 0002	Y 0000 Y	0003 X			
-	instr ad	H 000	AD0 X 600 X	X 008 X	020 021	¥ 01C	X 011		01E	X O	nF ¥	020 021	Y
-	instr	H 0000	201 X 3010 X	0020 X 4000	X 302A X	4006	9017	SOOF	9037	X	5015	X 302A X	4006
-	port1addr	U O	0 X 2 X	0	X 5 X	0 X 7	X 2 X	7 X	1 X 7	X 6	X 2	X 5 X	
out	idi_sel	B 1				<u> </u>			~				
out	pop sel	80											
915	wenout	B 0											
015	wen	80		_									
941	carryen	B 0											
out	carrysta	B 0											
out	write_n	80											
-	write_n	B 0000	X 0000 X 0010	X 0100 X	0000 X 0101	X 0000	X 1010 X	0001 X	1001 0110	X 1110	0010	X 0101	- <u>x</u>
out	write_n	80											
-	rdin	н 0000	0001	0020 X 0000	X 0001 X	0000 X FFFE	X 0001 X 0000 X	FFFF X OC	002 0000	X 0003 X 0000	X 0002 X 000	13 X 0002 X	
-	dmernin	H 0000	X 0001 X 0000 X 0001	X 0020	X 0001 X	0020 X FFFD	x 0001 X	FFFE X OC	000 0002 FFFF	X 001C X 0003	X 0001 X 000	12 X 0001 X 0002 X	
5	q1	н 0000	X 0001 X 0000 X 0001	0020	X 0001 X	0020 FFFD	0001 X	FFFE X 00	000 X 0002 X FFFF	X 001C X 0003	X 0001 X 000	12 0001 0002	
=	q2	H 0000	000 X 0001	X 0020	X 0001 X	001C FFFD	FFF	ε	FFFF	X 0000 X	0001	X 0002 X	0003
8	ROout	H 0000	0001	X									
*	R1out	н 0000	0000						X				
*	R2out	H 0000	X				0001				X		
*	R3out	H 0000											
*	R4out	H 0000											
5	RSout	H 0000										X	
5	REout	н 0000		001C						X			
5	R7out	н 0000					X FFF	ε Χ	FFFF	X			
	pc_ont_en	B 0											
eus S		~ >	4		1 : : : -								

	Name	Value at 0 ps	0 ps 320,0 ns 640,0 ns 960,0 ns 128 us 1.6 us 1.92 us 2.24 us 2.56 us 2.88 us 3.2 us 3.52 us 3.84 us 4.16 us 4.48 us 4.8 us 5.12 us 5.44 us 5.76 us 6.08 us 6.4 us 6.72 0 ps
in	CLOCK	B 0	
out	fetch	B 1	้ โดกมนคนอาณากอนกอนกอนกอนกอนกอนกอนกอนกอนกอนกอนกอนกอนก
out	exec1	B 0	
#	> daddr	H 000	
*	> dout	H 0000	0000 XXXX00XX00XX 0000
*	> instr_ad.	Н 000	003
*	> instr	H 0000	B000
*	> port1add	ir UO	0
out	ldi_sel	B 1	
out	pop_sel	B 0	
out 🍆	wenout	B 0	
out	wen	B 0	
out 🍆	carryen	B 0	
out 🍆	carrysta	B 0	
<u>eut</u>	write_n.	B0	
8	> write_n.	B 0000	00000000000000000000000000000000000000
<u>eut</u>	write_n.	B0	
*	> rdin	H 0000	@#@V#V###20026000000###W###W###W###W##########
*	> dmemin	H 0000	0020
*	> q1	H 0000	0020
*	> q2	H 0000	0020
*	> R0out	H 0000	\$@\$\$\$000C\$@}\$
*	> R1out	H 0000	⊗ 0002 X 0001 X 0002 X 0000 X 0002
*	> R2out	H 0000	6000 X001X000X 0001 X000X 0001 X 0002
*	> R3out	H 0000	0000
*	> R4out	H 0000	0000
*	> R5out	H 0000	C 0000 X 0001 X 0002
*	> R6out	H 0000	000 0003 X 0012 X 0003 X 001C X 0003
*	> R7out	H 0000	0000 WC FFFD X0(00WK FFFD X0(0) 0000
out 🍆	pc_cnt_e	en BO	
out			

FIB(3) = 3 191 cycles 3.83us

	Na	me	Value at 0 ps	0 ps 320.0 ns 640.0 ns 960.0 ns 1.28 us 1.6 us 1.92 us 2.24 us 2.56 us 2.88 us 3.2 us 3.52 us 3.84 us 4.16 us 4.48 u 0 ps	us 4.8 us 5.12 us 5.44 u	us 5.76 us 6.08 us 6.4 us	6.72 us 7.04 us 7.36 u
in	CLC	оск	B 0				רמודות היותו היותו הניתו היותו הי
out	feto	:h	B 1				
out	exe	ic1	B 0		הה הההה הההה הההה הההה הההה ההה	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	
-	> dad	ldr	н 000	<pre>CODE 000 X(//r mt 000 X(//r mt 000)/0)(//r mt 6 // (/) //r (000)/(0)/ //r mt // //r mt // //r mt // 000)/(0) // (//r mt // // //r mt // // //r mt // //r mt // // // // // // // // // // // // //</pre>			000
-	> dou	rt	н 0000				0000
-	> inst	r_ad	H 000				003
-	> inst	r	н 0000	00000000000000000000000000000000000000			B000
-	> por	t1addr	U 0	000000070000000000000000000000000000000			0
out	ldi_	sel	B 1	ի ուս ուսու հետու հետու			
out	рор	_sel	в 0				
out	we	nout	в 0	ที่สุดการปลากสารที่สุดที่สุดในสารที่สุดที่สุดที่สุดที่สุดที่สุดที่สุดที่สุดที่สุดที่สุดที่สุดที่สุดที่สุดในสารที่สุดไม่ สารที่สุดที่สุดไ			
out	we	n	в 0	איז אין אין איז			
out	can	ryen	в 0				
out	can	rysta	в 0				
out	wri	te_n	В 0		กมนกมนกมนกมนกมนกมนกมนกมนกมนกมนกมนกมนกมนก	ากการการการการการการการการการการการการกา	
5	> wri	te_n	B 0000				0000
out 🍆	wri	te_n	В 0				
5	> rdin		н 0000	@#3000000000000000000000000000000000000			0000
*	> dme	emin	H 0000	Marc @A@OMMINGKOCOMMOROACHCOCOCACAMONCOMOCOMONO ANA MANANA MANANA ANA ANA ANA ANA ANA A			0020
	> q1		н 0000	punc@vaacammm@vaacammox@vamconcovmconcovmconcomoxacamconconconconconcommmmocomccanamcommc			0020
*	> q2		н 0000	()))@\@\()()@\()()@\@\@\()@\()@\()@\()@\			0020
*	> R00	out	H 0000	\$@\$\\${@@C\$@}\$\$\$@@C\$@}\$			0020
*	> R10	out	H 0000	00 0003 X 0002 X 0001 X 0002 X 0000 X 0002 X 0003 X 0001 X			0003
*	> R20	out	H 0000	0000 0001 0000 0001 0000 0001 0000 0002 0000 0002			0003
*	> R30	out	H 0000		0000		
۲	> R40	out	н 0000		0000		
۲	> R50	out	н 0000	C 0000 X 0001 X 0002 X 0001 X			0003
۳	> R60	out	H 0000	00X 0003 X 0012 X 001C X 0012 X 001C X			0003
5	> R70	out	H 0000	(0000 XXX FFFD XXX FFFA)\$(\$FFF)XX FFFA)\$(\$FFFD)\$(\$FFD)\$(\$XX FFFD)\$(\$XX FFFD)			0000
out 	pc_	cnt_en	B 0				

FIB(4)=5

359 cycles

7	1	9	us

	Name	Value at 0 ps	0 ps 2000 ns 640 0 ns 960 0 ns 12 pus 1 6 us 15 pus 22 4 us 25 6 us 2 88 us 32 us 35 2 us 38 4 us 41 6 us 4 48 us 4 8 us 51 2 us 54 4 us 5.76 us 6.08 us 6.4 us 6.72 us 7.04 us 7.36 us 7.88 us 6.04 us 8.32 us 8.99 u	8.64 us 8.96
in_	LOCK	B 0		
	etch	B 1		Unitational
	xec1	B 0		הההההההה
	laddr	н 000		000
	lout	н 0000	0000 WWW.0000 WWWW.0000 WWW.0000 WWWWWWWW	0000
-	nstr_ad	н 000		003
	nstr	н 0000		B000
5 > 1	ort1addr	U O		0
	di_sel	B 1		
	op_sel	B 0		
	venout	B 0		
	ven	B 0		
	arryen	B 0		
	arrysta	B 0		
	vrite_n	B 0		הההההההההה
5 > 1	vrite_n	B 0000	81848C01818C01818C01818C018180000000C081800000000	0000
	vrite_n	B 0		
5 > 1	din	н 0000	<u>88900000000000000000000000000000000000</u>	0000
5 > 1	Imemin	н 0000		0020
5 > 1	rt -	н 0000		0020
5 > 1	2	н 0000		0020
5 > 1	l0out	н 0000	\$@98(eeec)@98(eeec)@98(eeec)@98(b3b) eeze X@98(b3b) eeze X	
5 > 1	l1out	н 0000	00 0004 X 0003 X 0002 X 0001 X 0002 X 0000 X 0002 X 0003 X 0001 X 0003 X 0004 X 0007 X 0007 X 0007 X 0002 X 0000 X 0002 X	0004
5 > 1	12out	н 0000	0000 X0001 X0001 X0002 X002 X002 X0002 X0002 X0002 X0002 X0000 X0000 X 0000 X0000 X000000	0005
5 > 1	13out	н 0000	0000	
5 > 1	l4out	н 0000	0000	
5 > 1	Sout	H 0000	K 0000 X 0001 X 0002 X 0011 X 0003 X 0001 X 0022 X	0005
5 > 1	16out	H 0000	80X 0003 X 0012 X 001C X 0012 X 001C X 0012 X 0003 X 001C X 0012 X 001C X	0003
5 > 1	t7out	H 0000	8000 W FFED W FFFA W FFF7 X00(FW FFF7 X00(FW FFFA X00(FW FFFA X00(FW FFFA X00(FFFA))), FFFA X00(FFFA)), FFFA	0000
* 1	c_cnt_en	В 0	กันวิทนาวีกันวิทนาที่เกิดกันวิทนามานาน และเป็นการและเป็	
ur .				

FIB(5)=8

611 cycles 12.23 us

		Name	Value at 0 ps	0 ps 0 ps	5.12 us	10.24 us	15.36 us	20.48 us	25.6 us	30.72 us	35.84 us	40.96 us	46.08 us	51.2 us	56.32 us
in_		CLOCK	в 0												
out		fetch	B 1												
out		exec1	B 0												
*	>	daddr	н 000												000
₩	>	dout	н 0000	0000											0000
₩	>	instr_ad	н 000												003
₩	>	instr	н 0000	н кононон (сонде))e andebe anange andebebe	анана (санос) санос) к									B000
₩	>	port1addr	U 0		an <mark>tananan tanta</mark> t	nan en al en a									0
eut		ldi_sel	B 1												
<u>eut</u>		pop_sel	B 0		1 11 1 11										
° ut		wenout	В 0												
<u>eut</u>		wen	В 0												
° ut		carryen	В 0												
<u>eut</u>		carrysta	В 0												
° ut		write_n	В 0												
₩	>	write_n	B 0000	•)••)•)••••••)•)••)••									0000
° ut		write_n	В 0			└── ── ─ ─			+++++++++				+++++++++		
₩	>	rdin	н 0000	-0-0-0-000000000	-18001800 <mark>9</mark> -0018000-0										0000
₩	>	dmemin	н 0000		of #1000000 0111001 #100 #0										0020
*	>	q1	н 0000	+0+0+0+0+00 0+001	01 0 1000000010 1001 0 100 10 10	10+0+0+0310+011110+0+0+0+0									0020
*	>	q2	н 0000))+)D(D	19										0020
₩	>	R0out	н 0000	NINNIKAK S	G W32/WK)K02	XXXXXXXXXX									0020
₩	>	R1out	н 0000		*****	WXXXXXXXX									0005
₩	>	R2out	н 0000	0000	NXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	X29XXXXXXXXX									0008
₩	>	R3out	н 0000											0000	
₩	>	R4out	н 0000											0000	
8	>	R5out	H 0000	0000 (00)	0000000	0005\00\00									0008
8	>	R6out	H 0000	(X 0012 X X	000000	()(01;)()(01)									0003
8	>	R7out	H 0000	XXXXXXXXXX	MCMMACHCHIM	NAKAKAN KANC									0000
<u>eut</u>		pc_cnt_en	B 0	_											
out					1 1 1 1 1 1 1 1 1 1 1 1										

FIB(6)=13

1031 cycles 20.63 us

		Value at	0 ps	5.12 us	10.24 us	15.36 us	20.48 us	25.6 us	30.72 us	35.84 us	40.96 us	46.08 us	51.2 us	56.32 us	61.44
	Name	0 ps	0 ps												
in	CLOCK	В 0													
out	fetch	B 1													
out	exec1	В 0													
3	> daddr	H 000	2012		an e in an	rijej njem jen	i) - in a								000
*	> dout	H 0000	0000												0000
-	> instr_ad	н ооо	-												003
3	> instr	H 0000		0 C 30 (C) C 30 (C) C 30 30 (C 30 0)	2000 M 20 M (C 20 20) (20 (C) (c m max m (c m 20)(<mark>0</mark> 00(0))	xx2xx0000								B000
*	> port1ad	dr UO	STATUTE STA	INDERSONAL AND DESIG.	ann <mark>a an an</mark>	nun numuss kind	niumiere 👘								0
out	ldi_sel	B 1													
out	pop_sel	B 0													
out	wenout	B 0													
out	wen	B 0													
out	carryen	B 0													
out	carrysta	ВО													
out	write_n	B0													
*	> write_n	B 0000	- <u>(</u>)))			.)))	•)•••)••••(0000
out	write_n	B0		1 11 11 11											
-	> rdin	H 0000	0.000	(m 0 = 000000-00 = 0 = 0= 0000-00		00 10 1010 1000 1000 1000 10	8-0-0800080 0								0000
-	> dmemin	H 0000		-d)	0.0) (0									0020
-	> q1	H 0000		-0) 0 -0000 000 000 000 000 000	(1 - 0 - 0 - 0 - 0 - 0 - 0 - 0 - 0 - 0 -	0 0-0 -0 0-00 0 -0000 -00 0	6-0-mit-mit-6-0								0020
-	> q2	H 0000)))-	00+000-0001+00-01	60) - - 0 - 00		1000-000								0020
-	> R0out	H 0000	omo	00000000000	02)))))))))))))))))))))))))))))))))))))										0020
-	> R1out	H 0000				000000000000000000000000000000000000000									0006
-	> R2out	H 0000	0000												000D
-	> R3out	H 0000											0000		
-	> R4out	H 0000											0000		
-	> R5out	U 0	0	X1X2XX3X1)	5 1 2 M	8 1 2	3 \ 1 \ ()								13
-	> R6out	H 0000	(X 001:	2 XXXXXXXXXX01	XIX101:XXX01)	XX0012XXXXXXX	(X) ()01CX								0003
-	> R7out	H 0000	XXXXXX												0000
out	pc_cnt_	en BO													
			- Lininia		i n'minin i nin i nin										

FIB(7)=21

1703 cycles





Appendix 11: Pseudo-random integer generator simulations
Using "typical" parameters in overview

A=25385=0x6329, b=3, n=8

213 cycles

Execution time: 4.27 us

Pattern: 0, 3, 10625, 34994, 47758, 51917, 49639, 25364, 39500 (does not seem to repeat)

	Nama	Value at	0 ps 3	320.0 ns	640.0 ns	960.0 ns	1.28 us	1.6 us	1.92 us	2.24 us	2.56 us	2.88 us	3.2 us	3.52 us	3.84 us	4.16 us	4.48 us 4
	Nume	0 ps	0 ps														
*	> dout	H 0000	(32)(1)(1)(00)	6329	<u>X 0000 X</u>	6329 X 0	000 X 6329	X 0000	X 6329	<u>X 0000 X</u>	6329 X 001	6329	X0000 XXX	6329	(0000)) 63	29 X0000X	K
*	> instr_ad	H 000	()()(0)(0)(0)(0)	X0X000000	000000000000000000000000000000000000000	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX)<0<1/1<0<0<2<0<	000000000	X1X0X0X0X0X0X2X	0000000000	00000000000	000000000000000000000000000000000000000	0,	000000000	0,0,0,1,1,0,0,0,0		XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
۳	> instr	H 0000	KOXOXOXOXO O	1)(1)(1)(1)(1)(1)(1)(1)(1)(1)(1)(1)(1)(1	00000000000	x; X;;X <u>0000</u> X;;X;(X))	(XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	000 XIIXIXIIXIXIX	(1)(1)(1)(1)(1)(1)(1)(1)(1)(1)(1)(1)(1)(X00000XX000X	X+X:X0000X+X:X+X	X; X;(X;(X; X; X;X0000	XXXXXXXXXXX	XXXXX0000XXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX		(%)%XX
8	> port1addr	U 0	(1)89898	0)(3)(4)(X 4 X5X3X_0_	34 <u>(</u> (4)	(5\(3\)(3\(4\	0 (4)(3)	0_33(4)0	X 4 X\$X9X_0	X3X9XX4X	X3X_0_X3X4X0	_X 4 X5X3X_0	_X3X4XX	<u>4)(5/3)(_0_)(3/4</u>	<u> </u>	
eut	ldi_sel	B 1			·····Γ												UU
eut	pop_sel	B 0															
out	wenout	B 0	ιΓ	ւլ_որ	ากกกกเก			որորոր		nnnnlr		munlm	mmm	עריעע	INNNILIL		ուուտորո
eut	wen	B 0		Ատ	mmu	Lmm	տուր	որորորո	wum	nnnnlr	Lmm	տուրա	mmu	n nuur	MMUUL	ուսուսու	աստուսու
<u>eut</u>	carryen	B 0															
out	carrysta	B 0															
<u>eut</u>	write_n	B 0	mm	www	mmm	תתתתת	mmm	ուսուսու	www.	ກກກກກກ	MMMM	mmmm	mmm	ກກກກກກ	տոտող	ուսուս	mmmm
8	> write_n	B 0000	0000000	01)()()(000	0 \10\()(\00)	XXX 0000 X10	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	0000 (10)	X00XXXX 000	0 (10)(0)(0)	(XXX 0000 X10)	()() X00() X X X 00	00 10000000	WXXX 0000		<u>X 0000 X10X</u>	X00)(X)(X
<u>eut</u>	write_n	B 0	Γ														
*	> rdin	H 0000	32000000	1/32/0000	0,0000,000,000,000	321,0000,0000	\$\$\$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$		0000320000	X900000000	32 0000 0000000000000000000000000000000	0000	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	()(32)(000)()()	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX		0000C
*	> dmemin	H 0000	0000	X0000	X:@X:X:@X	()()()()()()()()()()()()()()()()()()()	*****	0000	000000000000000000000000000000000000000								
*	> q1	U 0	00000000	X8XX •	#3### <u>5</u> X	(73)(•)(3)	XXXX5XXXX	•)\$\$\$\$\$\$\$\$	<u>(5)</u>)))))))))))))))))))))))))))))))))))	****	\$\$\$\$\$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$)W3AAAAS	X20X	00005002	(NOR CONTRACTOR OF CONTRACTON	K5X0X
₩	> q2	U 0	0	XXX 💿	000000000	000C • V60	000000000	•)33(13()	96003XO	x2000000	XXXO XXX	XXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	000000000	00000000	())))))))))))))))))))))))))))))))))))	050@
*	> R0out	U 0	0	22 (0	χ_1_χ5χ	22 0 1	X5 X 22 X	0 X 1	5 22 0	X_1_X5)	22 \ 0 \ 1	<u> </u>	χ_1_χ_5	<u>22</u> <u>0</u> <u></u>	1 (5) 22	<u> </u>	(5)(22)
*	> R1out	U 0	0														
**	R2out	U 0	• X														
8	> R3out	U 0		8	X	7	X	, Х	5	Х	4	Х 3	X	2	X	1 X	
*	> R4out	U 0		0	X	з Х	10622		24369)()(1	2764 3	4159))(63258	(3) 41261	X	
*	> R5out	U 0		0	X	3	1062	5 X	34994	X	47758	51917	X	49639	25	364	
*	> R6out	U 0															
8	R7out	U 0															
**	> mul_res_1	U 0			0	X	10619	X	24366	X 1	2761	4156	X	53255	41258	X	
-	> mul_res_2	U 0			0	X	1	X	4114	Х 9	439	4944	X	1610	24502	X	
out	pc_cnt_en	B 0	mmu	www	mmin	กกกกกก	mm	mmm	umm	mmu	mmm	mmm	mm	mmm	տուտ	mmm	แทบทางกา
out	pc_sload	B 0			Л		<u>n</u>		Л	Л		Л	Π		Π		
out	ram_wren	B 0															

	Name	Value at	0 ps	5.12 us	10.24 us	15.36 us	20.48 us	25.6 us	30.72 us	35.84 us	40.96 us	46.08 us	51.2 us 56.32	
		Name	0 ps	0 ps										
5	>	dout	H 0000		ж									6329
5	>	instr_ad	H 000											
5	>	instr	H 0000	50010010000000000	* (0000
5	>	port1addr	U 0		₩									0
out		ldi_sel	B 1											
out		pop_sel	в 0											
out		wenout	в 0											
out		wen	в 0											
out		carryen	B 0											
out		carrysta	B 0											
out		write_n	B 0											
*	>	write_n	B 0000		K									0000
out		write_n	в 0											
*	>	rdin	H 0000	2)#)#2028)#0#2										0000
*	>	dmemin	H 0000											0000
-	>	q1	U O	11ETTLEPW21ET										0
-	>	q2	U 0		K									0
-	>	R0out	U 0		*									0
-	>	R1out	U 0										25	5385
*	>	R2out	U 0											3
-	>	R3out	U O	0000000	3									0
-	>	R4out	U O	CONTRACTOR	×									14136
-	>	R5out	U 0	60000000	X									39500
-	>	R6out	U 0											0
-	>	R7out	U 0											0
-	>	mul res 1	UO	TO XXXXXXX	X									14133
-	>	mul res 2	UO		Ŷ									15982
out		pc cnt en	В 0											
out		pc sload	BO											
out		ram wree	B 0											
-		an_wen								+++++++++++++++++++++++++++++++++++++++	+++++++++++++++++++++++++++++++++++++++			

Using smaller values and longer loop, see if it repeats:

A=0x500, b=1, n=0x10

421 cycles

Doubled loop length, execution time doubled: 8.43 us

		Name	Value at 0 ps	0 ps 0 ps	1.28 us	2.56 us	3.84 us	5.12 us	6.4 us	7.68 us	8.96 us
in		clock	B 0	3							
out		fetch	B 1	nnemi	nmnaninnninn	anmininmininin	nalarinmnamr	Inninaninnnar	nimininininin	naminiminanin	inninarimini
out		exec1	B0		ninnnmnnmn	nninnninnnmr	nmanniannia	nnmnnmnnn	nninnnmnnmr	Inninninninnin	nnmnnninn
		daddr	H 000								
		dout	H 0000							SOUNDSOUNDSOC	×m.
eut		instrad	H 000								
Т 9Щ	、	instr_au	H 0000			nnnnnnnnnnnnnnnnn Amminikannnna		nonanananananan Manimikani	//////////////////////////////////////	WARNAR AND	NINN
eut		nisu		- William - Will	nin Annin Annin Al-Albara Albar	(√1000000√100000√10 (⊲\A&A&A\A\A\A\A\A\A\A	00000/000000/00000 864867388738678	n Annonin Annonin An Rugadh an Anna Anna Anna Anna Anna Anna Anna	aaan vaaan vaan weede	WANDARANA ANA ANA ANA ANA ANA ANA ANA ANA AN	11111111 1118/14/
out	/	portradur	00 B4	<i>₩₩₩</i> ₩₩ □	איילאילאעליאיילאיילא מי רבי מי רבי מי	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	מי ריינים אישאישאיש מי ריינים	₩S/₩Y/₩S/₩Y/₩S/	אייטאיטאיטאיעאיע חריי חת ריי חת	איצאאיאאיצאאיאאיש זרי חחרי חחרי	₩₩\/₩\
out					00 000 000	ו נווען נווען נ	יונין נוונין נווין	ן ראמה ראמה ו	-00 1-00 1-00	U LUUU LUUU I	
-		pop_sei	80		n nnnn nnmi	nnine ninne mr			neu annea anmr	i nninn ninnn mr	nnninnninn
-		wenout	80	רביים אנטעריים און און און איז	11.11.11.11.11.11.11.11.11.11.11.11.11.	1010000,000000,000 mining mining min	NARAL RADALAN MULAN RADA Tarah dari kara tarah dari kara tarah	נות המשלה המשלה היות היות היות היות היות היות היות היו	ANGULANAN MULANAN MULANAN MULANAN	1.0000000000000000000000000000000000000	NURURURURU NANANANAN
-		wen	80			ANNUNE INVENTE IN	NIKALUMININI ULUKININI		ANUT WALKEN AND A MARKANA		
-		carryen	B 0	-							
-		carrysta	B 0		nınnamanma	anınnınnımı	nmnnnnnn	nnmnnmnnn	nninnnmnnmr	Inninninnnmi	nomonino
-		write_n	B 0			In an					
-	>	write_n	B 0000	MARCON .	ANK AND	KOMMKOMMKO	MARKOM MKOMA	RUXAMIKUXAMIKU	ANNE CARANE CARE	MCXMAN XMANO	XWKWK
eut		write_n	B 0					L A LUMINI L A AMULTU L A A		a a anna a a anna a a	
*	>	rdin	H 0000)))))))))))))))))))))))))))))))))))))	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	()))))))))))))))))))))))))))))))))))))	nne (China) (China)	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX		NXX MM IXXX MM IXXX	MMX
*	>	dmemin	H 0000		an canana an a	n orange and the second se	noxonoxono	NAME OF A DECK	MOOR AMORE AND		MKX
*	>	q1	U 0		(M)(HEN)(HIC)	, and the second se	ana)) (an t ai) (an t a	X NINGAX NINGAX		ALCHINE DI CHINE DI CI	
*	>	q2	U 0		EDIX XUOEDIX XUOEDI	EXORADE/ORDADE/O	NANK MORANK MORA	a nomina nomina n	ADDIA NOMINIA NOME	OCXOCOC XOCOC	
*	>	R0out	U 0	0	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXX	DXXXDXXXDX	XXXXXXXXXXX	DXXXXXXXXX		0;;;(
*	>	R1out	U 0								
*	>	R2out	U 0	<mark>X</mark>							
₩	>	R3out	U 0		15 \ 14 \	13 🗙 12 🔪 11	X 10 X 9 X	8 (7) 6	X 5 X 4 X	3 2 1	Х
*	>	R4out	U 0		1 1281 128	31 1281 1281	(1281)(1281)(12	81 (1281) (1281)	(1281)(1281)(12	81 1281 1281	X
**	>	R5out	U 0	0 X	1 1282 25	63\3844\5125	6406 7687 8	968 10249 11530	12811/14092/1	5373 16654 1793	5
*	>	R6out	U 0								
*	>	R7out	U 0								
*	>	mul_res_1	U 0	0							

• Series still does not seem to repeat

Keeping big and longer loop:

0x6329, b=9, n=0x20 837 cycles Execution time: 16.75 us

	Name	Value at 0 ps	0 ps 0 ps	1.28 us	2.56 us	3.84 us	5.12 us	6.4 us	7.68 us	8.96 us	10.24 us	11.52 us	12.8 us	14.08 us	15.36 us	16.64 us
in.	clock	в 0														
out	fetch	B 1	nnamnn	mnanınnını	IAMANMAAAAA	uunnmnami	INNINANIANAN	umnnaann	inamanmaaaa	nnnanmnnn	IANNNAANN	nanannana	Innmanmnnau	INNINNAMAN	naninnnnanm	Innmanninna
out	exec1	B 0	Innnnr	nnnnnnn	nnnnnnnn	nmaaaaaaa	aamaamaaaa	nnnnnnnn	nnnnnnnn	INMANANAN	nannannan	nnnnnnnm	nnnnnnnm	namanananan	nnmnnnnn	anninanmann
*	> daddr	H 000	#X200X#0	000000000000000000000000000000000000000	0.000.000.000.000.000	000 000 000 000	00,00,000,00,000	HC000 (HC000 (HC	000,000,000,000,000,000	10000110000	000 00 000 000	1000 1000 100	000100000000000000000000000000000000000	000000000000000000000000000000000000000	000000000000000000000000000000000000000	000000
*	> dout	H 0000	*** 32\}}	32!\()32!\()(3	2\$\0\32\$\0\32\$\	1/132!/11/132!/11/1	32\$\()(32\$\()(32\$)	1)(1329)1)(1329)1)	32!(1)(32!(1)(32!	()(32!)()(32!)())(32!)()(32!)()(32!	(1)(32!(1)(32!(1)	(32!\)(32!\)(32	\$ }} } 32 \$ } } }}}}	(32!)(X 132!)(X 13	2\$\\$\\$32\$\\$
*	> instr_ad	H 000				Martin Martin Ma										
5	> instr	H 0000	(MORONO)	ancianancianana	()))))))))))))))))))))))))))))))))))))	and an	ROMMIC MOMENT	aaniyaaaniyyaaa	(X)((((X)))	ann cannacan	anomanomano	and the second	nikanankanank	MINING MINING MINING MINING M	naronnan sanana	
*	> port1addr	U 0	*** **	***	0(#:#0(#:#0(*#0#:#0#:	10/#X0/#X0X	***	****	HCHO(HCHO)	K#0###0###0	(#\$#0(#\$#0(#	*****)#(#)#(#)#	(#0;#()#0;#()#	omonomotic
out	ldi_sel	B 1		տտո	ոտո		տատ		տորոր	ուտ		ստորո	տերութ	ստող	տուտուտ	
out	pop_sel	B 0														
out	wenout	B 0	_nnnnr	LUUUUUUUUU	nnni ninni mn	nlinnniinnii	LIAMULIMAULIA	nwanatam	n man waa n	IAN AANULANI	nlamalmaalu	nnunnnunm	WANDUUAAWA	AAN INNIN INN	namananan	ANNU ANMUNA
out	wen	B 0	זוווווווו	LUUUU UUUU	nnn nnn m	n nanunau	linmuumnuun	nwanaanm	n man unan n	INA MANLANI	MUNANUMANUN	nnunnnunm		ANN INNN INN	NIMANAAAN	AUNTUNUN
out	carryen	В 0														
out	carrysta	. B0														
out	write_n	В 0	Innninnr	nnnnnnn	nnnnnnnn	nmaaaaaaa	nnmnnmnnn	nnnnnmnnm	nnnnnnnmi	INMANANAN	nnnnnnnnn	nnnnnnnn	nnnnnnm	namanananan	nnmnnmnn	anninnamann
*	> write_n	B 0000		AND	(CANANGCANANCO)	MMC/MMC/M/	MANNA ANNA ANNA ANNA ANNA ANNA ANNA ANN	XOMOROMOX	MOMONOMONO (onon@onon@o		XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	(M)()(M)(M)()(M)	Dahahadahahadah	omonomonomon	OMMONIA C
out	write_n	В 0														
*	> rdin	H 0000		actine composition			actining and a second	n dixxon dixxonid		NICOM CON	n contracontraco	and on the other other	XXXIII (XXXIII (XXXIIII (XXXIII (XXXIIII (XXXIIIII) (XXXIIIII) (XXXIIIII))XXXIIIII)XXXXIIIIIIXXXXIIIIIXXXXIIIIIXXXXIIIIIXXXX	San an a		
*	> dmemin	H 0000	MCMCMMC	an a	(CHOXOHOXOD	KOXXIII (KOXXIII)	icollicitatiosci	non nikolanik	analka kalka ka	akan da kan d	NAX HANDA ANNAX	nox nox m	on and a children and	NICOCON CONTRACTOR	once nto nce nto n	
*	> q1	U 0	MOMORAN	na china alica	CHIRODONIKO/CN	KOX AIKOX AIKO	actaticación (de la calenda)	ntar takan tak	新物が動かり	nn ann a	kod kalkod (alkod)	an contractal	(A)	糊和甜椒和甜	on callen callen	(##C#C##C#
*	> q2	U 0	O ####	n an in a	canna canna ca	n de la competition d	a onna onna d		an a	nin anin a an	ox mox mox	nna anna an	araa ahaa ahaa ahaa ahaa ahaa ahaa ahaa	ooden ooden ood		
*	> R0out	U 0	0000	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	00000000000	XXXXXXXXX		DXXXXXXXXXX	XXXXXXXXXX	DXXXXXXXI	XXXDXXXDXXX	0,000,000	KXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	000000000000	000000000000000000000000000000000000000	\$0\$\$\$\$0\$#C
*	> R1out	U 0	_ X													
*	> R2out	U 0	_ X													
*	> R3out	U 0	32	31 X 30 X 2	29 (28 (27	26 25 1	24 23 22	<u>21 20</u>	19 X 18 X 17	X 16 X 15	(14)(13)(12	<u>X 11 X 10 X</u>	9 (8) 7	X 6 X 5	4 X 3 X 3	2 <u> 1</u> <u> </u>
*	> R4out	U 0	O X	9 31866 757	1 38292 2477	8702 8247 12	408 32753 (15218)	(\$1435)(\$2828)(\$8	3545 11894 4847	30032 17577	14746 7267 34500	x17549x33182x54	6407 (\$117¢)(171	8907	5004 17781 2426	628490878
*	> R5out	U 0	• X	9 31875 394	146 12202 24679	(17845)(10556)(52	2964 20181 65399	61298 28590 1	1603 23497 2834	58376 40417	19627 26894 1585	8 33407 1053	57460 53100 648	13 62495 5866	10870 28651 529	36464
8	> R6out	U 0														
5	> R7out	U 0														
5	> mul_res_1	U 0	0	31857 756	2)38283/12468/5	58693\58238\42	399\32744\45209\	61426(32819)(48	11885 4838	(30023)(47568)(4	4737 7258 54491	(17540)(33173)(56	5398 51167 1170	4×63209×8898×4	995 (17772)(2425	53 49078
out	5 mil 0		N	V a Vina	Vices N cone Ves	4000 100707000	le Me knev (nede)	A TE & AVOD TOEVAD	TAEVADOREV ADAT	4077 446014	0410V12001V1014	Vation Var	020000000000000000000000000000000000000	eV reak for deak Va	460 V 4000 V 600	Vood

Appendix 12: "Find value in list" Benchmark results Length 7:

107 cycles

, 2.15 us

	Nama	Value at	0 ps 160.	0 ns 320.0 ns	480.0 ns	640.0 ns	800.0 ns	960.0 ns	1.12 us	1.28 us	1.44 us	1.6 us	1.76 us	1.92 us	2.08 us 2.2
	Hume	0 ps	0 ps												
in_	clock	B 0	տոտո	ກກກກກກກກ	www.	տորոր	տոտուս	mmm	տորոր	տոտող	տուսող	ոոոու	տոտո	տուու	տոտոտո
°₩	fetch	B 1	UUUU	unnn	JUUUU	UUUU	տու	MM	որող	nnn	ուու	ուու	JUUU	ாஸ	unnn
<u>out</u>	exec1	B 0	תתת	nnnn	uuuu	nn	nnn	UUUU	ՄՄՄՄ	ww	UUUU	ՄՄՄՍ	UUUU	uuu	nnnn
۲	> daddr	H 000	000 007 000 007	0000000000010 FF#	F(300)302 000	03 FFB F00	x004 000 x06x	FFC (F) 200 (207)	000 (JOC FFD	F 200 10C 00C	JO9 FFE (F)	00(10# 000	DOF FFF F JOO	210 000 211	1 000 (F 500 511)
۲	> dout	н 0000	0000 000 000	00,00,00,00	0000 \000 000	0 \00 \ 0000	X00X 0000 X00	0000 000	0000 000	0000 X00X 0	000 (00) 000	0000 0000	0000 0000	X01X 0000 X	00\ 0000 \00\
۲	> instr_ad	H 000	0,001,002,003,0	04/305/306/307/ 008	000×00E	07 008 2092	04\10E\10C\107	008 2092104208	008	00/10/10E	x107 008 x109	X02X0EX00X0	00 008 000 00	\$\00E\00C\007\	008 X09 X00
*	> instr	H 0000	00,000,000,01	0000200001064	11 \41\04\40\00	01 6411 41	04/40/00/01/	6411 (41)(04)	40\00\01\(641	1 41 04 40 4	00010 6411	1) 04 40 00	01) 6411 41	04 40 00 01	6411 41 02
**	> port1addr	U 0	0 1 0 3	X0X4X0X 2	X o	2	X 0 X	2	0 X 2	X o	2	X 0)	(<u>2</u>)	_ 0 X	2 5
eut	ldi_sel	B 1													
out	pop_sel	В 0													
<u>out</u>	wenout	в 0	תתת	سيسسر	1			UTL		11					ாப்பட
<u>eut</u>	wen	B 0	תתת	للملمم		JULL				uuru			ллш	ــــــــــــــــــــــــــــــــــــــ	
<u>eut</u>	carryen	В 0													
eut	carrysta	. во													
out	write_n	В 0	תתת	سيسبس	ւսուս	nnn	سممر	uuu	uuuu	ww	unur	տու	uuuu	uuu	nnnnn
*	> write_n	B 0000	0000 00 000	01/00/10/00/01/	0010 (00) 000	0 01 0010	X00)X 0000 X0	0010 00	0000 01	0010 (00) 0	000 010 0010	0000 0000	01 0010	000 0000	011 0010 10
<u>94</u>	write_n	в 0		Г_											Л
**	> rdin	н 0000	0007		000 \00:\ 000	0 00 FF) 300	00,000,0000,000) FFI (10000 00	0000 00 FF	00000000	00 (00 FF) () (0	0000 (00) 0000	00 FF 10000	i)(01)(0000)	00\00\1\00\01\0
*	> dmemin	H 0000	20(0007))(000(000(000/000	1)FFFA 00 0003	FF/0002 FFFB	00: 0006 FFIX	0003 FFFC 00	000C (FF) 0004		9 FFI 0005 FFI	E 00 000F	FFI 0006 FFFF	00 0011 /FF	0007 0000
*	> q1	H 0000	20(0)0007()(0	000/000/000/000/000	1)FFFA 00 0003		00: 0006 FFI	0003 FFFC 001	000C (FF)(0004		9 (FF) 0005 (FFI	E 00 000F	FFI 0006 FFFF	00 0011 /FF	0007 0000
**	> q2	H 0000	00000000	00,00,00,00,00	07 \00\00 \$00\00	00 0007 000	00,00,00,00	0007 000 00	000 000 000	7 0000000000	0000 0007 0007	00,000,000	00) 0007 000	00 00 00 01	0007 000
**	> R0out	H 0000	0007		001	0003	X	1006 X	0000	X	0009	X	000F		
**	> R1out	н 0000	0000												
*	> R2out	H 0000	00	00 X000	1 FFFA	0002	FFFB	0003 FFI	FC X0004	FFFD	0005	FFFE	0006	FFFF	0007
*	> R3out	H 0000	0000 X												
*	> R4out	H 0000	0000	X											
*	> R5out	H 0000						0	000						χ
*	> R6out	H 0000													



Length 10 (typical): 149 cycles 2.99 us



Length 15:

219 cycles



Length 20:



Appendix 13: Instruction testing process

LDI test

Instruction mem:

LDI 1	0001
STP	B000



LDR

LDR (no offset) test:

Data memory: 0x0004 0x00AB

Instruction mem:

LDI 0x1	0001
LDR 0 0 R1 R0	1008
LDI 0x0	0000
STP	B000

• Note - does not work yet - implemented later



• Removed Idi and Idmfd from wen and added the OR gate so that it writes when the write_next bit in status bus is active even when wenout is off

Another issue:

- Daddr remains at 0
- Daddr pin was not named



- During STP, R0 is written to since the initial values in memory is 0000, which is LDI 0000, which loads R0 with 0000
- Need to add a STP bit in status register to say that the program has stopped and wenout in ALU should be off
- After doing this, wenout was stuck at 0
- This is because stp_en is initialised to 1
 - Should expect 0 as register is initialised to 0, so initial status[5] should be 0

Status register:

• Do we need to have separate registers so that I can individually enable writing to the different bit fields - actually maybe not as it needs to be returned to 0 if it is done



- Could not see stp_en as bit size was not right in simulation
- It is always at 1

Stp_en is actually not needed as instr stays ta B000, for some reason thought it went to 0000

- But wenout is off while R0 is changing???
- Status[3] is on, which turns on wenout
- Need it to turn off one cycle afterwards

• Don't need stp_en stuff and we need to separate status register to cin register and write_next register



• Same problem, wen is on with stp as write_next_out[3] is on and is not switched off after loading in next cycle

Issue was that write_next_en was only on for Idi and Idmfd, when it should always be on during exec1, so that it can actually go back to 0

// write_next_en enables writing to write_next register - needs to update for every instruction during exec1 so that it returns to 0
assign write_next_en = exec1;

Simulation working:

*	> daddr	Н 000	000 X 001 X 000
*	> dout	н 0000	0004 X 00AB X 00AB
*	> instr_ad	н 000	000 X 001 X 002 X 003
*	> instr	H 0000	0 D001 X 1008 X D000 X B000
out	ldi_sel	B 1	
out	wenout	B 0	
out	wen	B 0	
out	carrven	B 0	
out	carrysta	B 0	
out	write n	B 0	
	write n	B 0000	
-	/ 1110_1	0 0000	
*	> q1	H 0000	
*	> q2	н 0000	C 0000 X 0001 X 0000
*	> R0out	H 0000	0000X 0001 X 0000
#	> R1out	H 0000	
#	> R2out	H 0000	0000
-			

• Realised this doesn't work as it will be pipelined - can't actually LDI after LDR - can't write to registers as there is a conflict for what to have as port1addr

Test STP right after LDR Data memory:

0x0004 0x00AB

Instruction mem:

LDI 0x1	0001
LDR 0 0 R1 R0	1008
STP	B000

Issue:

1				
in	CLOCK	В 0		
out	FETCH	B 1		
out	EXEC1	В 0		
*	> daddr	H 000		000
*	> dout	H 0000	00X 0004 X 00AB X	0004
34	> instr_ad	н 000	(000 X 001 X	002
*	> instr	H 0000	00 [×] 0001 × 1008 ×	B000
out	ldi_sel	B 1		
out	wenout	B 0		
out	wen	в 0		
out	carryen	B 0		
out	carrysta	В 0		
out	write_n	В 0		
₩	> write_n	B 0000	(0000 X 0001 X 1000 X	0000
*	> q1	н 0000	(0000 X0001 X 0000 X0001 X	00AB
₩	> q2	H 0000	(0000 X 0001 X	00AB
*	> R0out	н 0000	(0000 X 0001 X	00AB
₩	> R1out	H 0000	0000 X	00AB
34	> R2out	н 0000	0000	

• R0 is changing since wen is enabled while daddr is 0 during STP exec1

• Port1addr needs to come from status register, not instruction word - forgot to actually take status register address and put it into port1addr -> enabled if status[3] is enabled

• Added MUX:



Old simulation:

in	CLOCK	в 0	
out	FETCH	B 1	
out	EXEC1	В 0	
*	> daddr	н 000	000 X 001 X
*	> dout	н 0000	001X 0004 X 00AB X
*	> instr_ad	H 000	000 X 001 X 002 X
*	> instr	H 0000	001 0001 0000 X
*	> port1addr	B 000	000 X 001 X
out	ldi_sel	B 1	
out	wenout	B 0	
out	wen	В 0	
out	carryen	В 0	
out	carrysta	В 0	
out	write_n	В 0	
*	> write_n	B 0000	0000 X 0001 X 1000 X
*	> q1	н 0000	0000 X 0001 X 0000 X 0001 X 00AB X
*	> q2	н 0000	0000 X 0001 X 00AB X
*	> R0out	H 0000	0000 X 0001 X 00AB X
*	> R1out	H 0000	
*	> R2out	н 0000	k i i i i i i i i i i i i i i i i i i i

• Port1addr is 1 during ldr instruction rather than one cycle after when dout has updated



- Status is not storing 1001
 - Storing 1000 so it loads to R0 (sees register address as 0)

```
• It was taking Rs into write_next rather than Rd address, corrected:
```

```
// Output to write_next register
assign write_next_out = {write_next_flag, instr[5:3]};|
```

Working;

	Nama	Value at	0 ps	80.0 ns	160.0 ns	240.0 ns	320.0 ns	400.0 ns	480.0 ns	560.0 ns
	Name	0 ps	0 ps							
in	CLOCK	B 0					лллг		תתת	
out	FETCH	B 1								
out	EXEC1	B 0								
*	> daddr	H 000	000	001						000
*	> dout	H 0000	0004							0004
*	> instr_ad	H 000	000 00	01 X						002
*	> instr	H 0000	0001	(1008)						B000
*	> port1addr	B 000	000	(001 X						000
out 🍊	ldi_sel	B 1								
out	wenout	B 0								
out 🍝	wen	B 0								<u></u>
out	carryen	В 0								
out	carrysta	В 0								
out	write_n	В 0								
*	> write_n	B 0000	0000	X 1001 X						0000
₩	> q1	H 0000	0000 (00)	(0000 X0A)						0001
₩	> q2	H 0000	0000							0001
*	> R0out	H 0000	0000						0(01
*	> R1out	H 0000	000	0 X						00AB
*	> R2out	H 0000							0000	

• R1 loaded with 0x00AB or mem[1] and R0 remains at 1

Test two LDR instructions after each other

• Works since the first cycle of the second LDR does not write to registers Data memory: 0x0004

0x00AB

Instruction mem:

LDI 0x1	0001
LDR 0 0 R1 R0	1008
LDR 0 0 R2 R0	1010
STP	B000

lss	ue:									
	Name	Value at	0 ps	80.0 ns	160.0 ns	240.0 ns	320.0 ns	400.0 ns	480.0 ns	560.0 ns 6
		0 ps	0 ps							
in	CLOCK	B 0		תתתת	תתתת	ייייייי	uuuu		uuuu	ստու
out	FETCH	B 1								
out	EXEC1	B 0								
*	> daddr	H 000	000	X 001	X					000
*	> dout	H 0000	10 000	4 X 00AB						0004
*	> instr_ad	H 000	000 0	01 🗙 002 🗶						003
*	> instr	H 0000	10 0001	1008 1010	X					B000
#	> port1addr	B 000	000	X 001 X010	X					000
out	ldi_sel	B 1								
out	wenout	в 0								
out	wen	В 0								
out	carryen	в 0								
out	carrysta	B 0								
out	write_n	В 0								
*	> write_n	B 0000	0000	<u> 1001 X 0</u>	010 X					0000
*	> q1	H 0000	0000000	001XA0X 0000	X					0001
*	> q2	H 0000	0000	X					000	
*	> R0out	H 0000	0000						0001	
*	> R1out	H 0000	000	ο <u>χ</u> ο						00AB
*	> R2out	H 0000							0000	
			<u> </u>						<u> i i i i i</u>	

• If the next instruction is an LDR, need to make sure that it does not reset write_next_flag to 0 as seen above

// write_next tells next instruction that the data that is now at dout can be written into the Rs of the previous instruction (for load instructions)
// if write_next_flag is already 1, set to 0 (if it is not another load), otherwise set to 1 if it is an ldr or ldmfd instruction
assign write_next_flag = (write_next_status[3] & ~(ldr|ldmfd)) ? 0 : (ldr | ldmfd);

	Name	Value at	0 ps	80.0 ns	160.0 ns	240.0 ns	320.0 ns	400.0 ns	480.0 ns	560.0 ns	640.0 ns
		0 ps	u ps								
in_	CLOCK	B 0		uuu	UUU	ww	ww	תתתת	תתתת	JUUU	uun
out	FETCH	B 1									
out	EXEC1	B 0									
#	> daddr	H 000	000	001						000	
*	> dout	H 0000	11 0004	00AB	X					0004	
#	> instr_ad	H 000	000 \ 00	1 🗙 002 🗶						003	
#	> instr	H 0000	(i) 0001 X	1008 (1010)						B000	
#	> port1addr	B 000	000 X	001 01	o X					000	
out	ldi_sel	B 1									
out	wenout	B 0									
out	wen	B 0									
out	carryen	В 0									
out	carrysta	B 0									
out	write_n	B 0						ΠП			
*	> write_n	B 0000	0000	X 1001 X 10	10 X					0000	
	> q1	H 0000	0000 00 0							0001	
	> q2	H 0000	0000						00	01	
	> R0out	H 0000	0000						0001		
	> R1out	H 0000	000	γ						00AB	
-	> R2out	H 0000		0000						00AB	
-	> R3out	H 0000							0000		

Test multiple LDR instructions

Data memory: 0x0004 0x0002 0x00AB

Inst	ruction	mem:								
LD	0x1		0001							
LD	R 0 0 R1	RO	1008							
LD	R 0 0 R2	RO	1010							
LD	R 0 0 R3	R2	101A							
ST	כ		B000							
000	1 10	008	1010	101A B	000					
Wo	rks:		Ib							
in	CLOCK	B 0	תת	תתתתת		บบบ	บบบ	บบบบบบ		uur
out	FETCH	B 1								
out	EXEC1	B 0								
₩	> daddr	н 000	000	X 001 X 00)2 X				000	
₩	> dout	н 0000	000	4 X 0002 X	00AB				0004	
₩	> instr_ad	н 000	000	01 🗙 002 🗙 003 🔪				c	004	
*	> instr	н 0000	HX 0001	X 1008 X 1010 X 10					B000	
₩	> port1addr	B 000	000	X 001 X 010 X	011 X				000	
out	ldi_sel	B 1		1						++++++
out	wenout	В 0		L						
out	wen	В 0					 			
out	carryen	В 0								
out	carrysta	В 0								
out	write_n	B 0								
*	> write_n	B 0000	0000	<u> 1001 (1010</u>	(1011)				0000	
*	> q1	H 0000	0000 000	x 0000 x00;x100)x100;					0001	
*	> q2	н 0000	0000	X 0001 X 00	02 X				0001	
*	> R0out	H 0000	0000					0001		
*	> R1out	н 0000	00	00 X				0002		
*	> R2out	H 0000		0000				000	2	
*	> R3out	H 0000		0000	X				00AB	
*	> R4out	H 0000						0000		
							 	+ + + + + + + + + + + + + + + + + + + +		+ + + + + + + -

Test positive offset LDR

Data memory: 0x0001 0x002 0x003 0x006 0x004

Instruction mem:

LDR 0 1 R1 R0 1048

LDR 0 2 R2 R0	1090
LDR 0 2 R3 R1	1099
STP	B000

Working:

	Nama	Value at	0 ps	80.0 ns	160.0 ns	240.0 ns	320.0 ns	400.0 ns	480.0 ns	560.0 ns	640.0 ns
	Name	0 ps	0 ps								
in	CLOCK	В0	лл		תתתר	uuuu	лллг	uuur	תתתו		
out	FETCH	B 1									
out	EXEC1	В 0					mm				
₩	> daddr	н 000	0 001	002 004	X					000	
₩	> dout	н 0000	0000	1002 🗙 0003 🗙 00	004 X					0001	
₩	> instr_ad	н 000	000	001 🗙 002 🗶						003	
₩	> instr	н 0000	1048	1090 1099	X					B000	
₩	> port1addr	B 000	0 00	1 X 010 X 0	11 X					000	
out	ldi_sel	B 1									
out	wenout	В 0									
out	wen	В 0									
out	carryen	В 0									
out	carrysta	В 0									
out	write_n	В 0									
*	> write_n	B 0000	0000 1	001 🗙 1010 🗶 10)11 X					0000	
*	> q1	н 0000	0000	<u> 100:100100:100</u>	X100-X					0000	
*	> q2	н 0000	00	100 🛛 🗙 0002	X					0000	
5	> R0out	н 0000							0000		
*	> R1out	н 0000	0000	X					00	102	
*	> R2out	н 0000	00	100 X						0003	
*	> R3out	н 0000		0000	X					0004	
*	> R4out	H 0000							0000		

Test negative offset LDR

Data memory: 0x0002 0x003 0x004 0x005 0x006

Instruction mem:

Instruction	Machine code	Action
LDI 0x4	0004	R0 = 0x0004
LDR 1 1 R1 R0	1848	R1 = mem[R0-1] = 0x0005
LDR 1 2 R2 R0	1890	R2 = mem[R0-2] = 0x0004
LDR 1 2 R3 R1	1899	R3 = mem[R1-2] = 0x0005
STP	B000	

				P			
in —		CLOCK	В 0			ПЛ	лл
out		FETCH	B 1			ШГ	
out		EXEC1	В 0			ΠL	
₩	>	daddr	H 000	000 X 003 X 002 X 003 X	000		
5	>	dout	H 0000	0002 X 0005 X 0004 X 0005 X	0002		
5	>	instr_ad	н 000	(000 X 001 X 002 X 003 X	004		
*	>	instr	н 0000	(X 0004 X 1848 X 1890 X 1899 X	B000		
*	>	port1addr	B 000	000 X 001 X 010 X 011 X	000		
out		ldi_sel	B 1				
out		wenout	В 0				
out		wen	В 0				
out		carryen	В 0				
out		carrysta	В 0				
out		write_n	В 0				
5	>	write_n	B 0000	<u>0000 X 1001 X 1010 X 1011 X</u>	0000		
*	>	q1	н 0000	0000 X00 X00 X00 X00 X00 X00 X00 X	0004		
5	>	q2	н 0000	C 0000 X 0004 X 0005 X	0004		
*	>	R0out	H 0000	0000			
*	>	R1out	н 0000	<u>0000</u>	005		
*	>	R2out	H 0000		0004		
5	>	R3out	H 0000		0005		
*	>	R4out	H 0000	0000			
out				$\amalg_{\ell^{+}} + + + + + + + + + + + + + + + + + +$			++++

- Can't write register after LDR, can only LDR, STR, JMP, STMFD and STP
 - What if must write register after LDR, how to wait?
 - Messy: Could have 2 separate registers for stack and link register to write at the same time may be more messy
 - Hacky: have a JMP instruction to next PC value to basically do nothing or STR a value in memory
 - Best?: Have logic that says if when write_next_status[3] is on and the current instruction is a write instruction, PC stays at the current instruction
 - Can add input to decoder that stops pc_cnt_en just like stp
 - I believe the current order of MUXes will make LDR writing to register take priority over LDI for example

Implementation:

ALU:

```
// write_next_stp goes to decoder to stop PC
assign write_next_stp = write_next_status[3] & ~(ldr | str | jmp | stmfd | stp);
```

Decoder:

```
// Need to think of all decode signals and then implement
assign pc_sload = exec1 & jmp;
assign pc_cnt_en = exec1 & ~(jmp | stp | write_next_stp);
assign ram_wren = exec1 & (str | stmfd);
assign ldi col ldi;
```

Test LDR and then LDI

Data memory: 0x0004 0x00AB

Instruction mem:

LDI 0x1	0001
LDR 0 0 R1 R0	1008
LDI 0x0	0000
STP	B000

Issue:



R1 loads 0

- When port1addr is still at 1, ldi has started already
- should make sure when write_next_stp is on, ldi_sel is off
- Also changed Idi_sel so it is in alu makes more sense

Working:

	Name	Value at 0 ps	0 ps 40.0 ns	80.0 ns 12	20.0 ns	160.0 ns	200.0 ns	240.0 ns	280.0 ns	320.0 ns	360.0 ns	400.0 ns	440.0 ns	480.0 ns	520.0 ns	560.0 ns	600.0 ns
in	CLOCK	в 0				лл	лл					лл	лл	лл			
out	FETCH	B 1															
out	EXEC1	в 0															
5	> daddr	H 000	000	001												000	
*	> dout	H 0000	0004	OOAB X												0004	
*	> instr_ad.	. H 000	000 001	X 002	2	X										003	
*	> instr	H 0000	00 <u>001</u>	1008	0000											8	000
#	> port1add	B 000	000	001 X												000	
out	ldi_sel	B 1															
out	wenout	B 0															
out	wen	B 0															
out	carryen	B 0															
out	carrysta.	B0															
out	write_n	B 0				1											
#	> write_n	B 0000	0000	X 1001 X												0000	
out	write_n	B 0															
3	> q1	H 0000	0000 X0001X	0000 X00ABX	0001	χ										0000	1
*	> q2	H 0000	0000 X	0001		X										0000	,
-	> R0out	H 0000	0000 X	0001		χ										0000	
-	> R1out	H 0000	0000	χ											0	0AB	
-	> R2out	H 0000												000	0		
out	> D2out	H 0000												0.00	0		

Test Mov after LDR Data memory: 0x0004

0x00AB

Instruction mem:

LDI 0x1	0001
LDR 0 0 R1 R0	1008
MOV 0 0 0 R2 R0	3010
STP	B000

• Added register file din as output

		Name	Value at 0 ps	0 ps 40.0 ns 80 0 ps	0.0 ns 120.0 ns	160.0 ns	200.0 ns	240.0 ns	280.0 ns	320.0 ns	360.0 ns	400.0 ns	440.0 ns	480.0 ns	520.0 ns	560.0 ns	600.0 ns
in_		CLOCK	B 0	L			תת		лл	лл		лл			лл	лл	
out		FETCH	B 1														
out		EXEC1	B 0														
-	>	daddr	H 000	000	001	Ξx										(100
-	>	dout	H 0000	<u>50) 0004 X</u>	00AB		X										0004
3	>	instr_ad	H 000	000 X 001 X	002	X										003	
*	>	instr	H 0000	00 0001 1000	18 X 3010	X										B	000
*	>	port1addr	B 000	000	001 X 0	10 X										(100
out		ldi_sel	B 1														
out		wenout	B 0														
out		wen	B 0														
<u>out</u>		carryen	B 0														
out		carrysta	B 0														
out		write_n	B 0														
5	>	write_n	B 0000	0000 X	1001	0010	X										0000
out		write_n	B 0														
3	>	q1	H 0000	0000 0001 0000	000ABX 0000	X										0001	
5	>	q2	H 0000	0000											0001		
₩	>	R0out	H 0000	0000											0001		
₩	>	R1out	H 0000	0000	XOOABX											0001	
≝	>	R2out	H 0000	00	000											0001	
*	>	R3out	H 0000											00	00		
-	>	R4out	H 0000	lk										00	00		

- Need a way to de-activate the current instruction from changing the
- Din should not be changed while port1addr is still at 1



• Din_sel should still select from din even though mov has started if the write_next_stp is enabled

// din_sel should be on if writing back to registers from alu (mov, add, sub, mul, bl)
// If ldr is writing, din should not be changed yet|
assign din_sel = (mov | add | sub | mul | bl) & ~write_next_stp;

۱Λ/	nr	кı	n	σ٠
• •	U.	N		۶.

		Name	Value at 0 ps	0 ps 0 ps	80.0 ns	160.0 ns	240.0 ns	320.0 ns	400.0 ns	480.0 ns	560.0 ns 64
	>	dout	H 0000	000	4 X III	00AB					0004
	>	instr_ad	н 000	000 (101 X 00	2					003
	>	instr	н 0000	0001	X 1008 X	3010					B000
	>	port1addr	B 000	000	χ <u>001</u> χ	010 X					000
out		ldi_sel	B 1		1						
out		wenout	В 0								
out		wen	В 0								
out		carryen	В 0								
out		carrysta	В 0								
out		write_n	В 0								
	>	write_n	B 0000	0000	X 1001 X	0010					0000
out		write_n	В 0	-							
	>	rdin	Н 0000	0001	X100X 00AB X	0001 X0AIX					0004
	>	q1	H 0000	0000 00	X 0000 X0AX	0000 X					0001
	>	q2	H 0000	0000	X					000)1
	>	R0out	H 0000	0000						0001	
-	>	R1out	Н 0000	00	00 X						00AB
	>	R2out	H 0000		0000	χ					0001
	>	R3out	н 0000							0000	

STR

STR (no offset) test:

Data mem: 0x0 0x2

Instruction mem:

Instruction	Machine code	Action
LDI 0x1	0001	R0 = 0x1
LDR 0 0 R1 R0	1008	R1 = mem[R0] = 0x2
LDI 0x0	0000	R0 = 0x0
STR 0 0 R1 R0	2008	Mem[R0] = mem[0x0] = R1 = 0x2
LDR 0 0 R2 R0	1010	R2 = mem[R0] = 0x2
STP	B000	

- CANT LDI AFTER LDR

Issue

Need to add STR to also change daddr •

Working:



MOV

// carryen enables writing to carry register - writes when cwen is enabled for add, sub, and mov assign carryen = exec1 & cwen & (add | sub | mov);

- Need to actually write carry ff during all mov instructions •
 - "No shift" writes 0 •
 - LSL writes MSB .
 - Shift right writes LSB •

Test: Data mem: 0xFFF0

Instruction mem:

Instruction	Machine code	Action				
LDI 0xF0	00F0	R0 = 0xF0				
MOV 0 0 0 R1 R0	3008	R1 = R0 = 0x0F0				
MOV 1 1 0 R2 R0	3610	R2 = R0 + 1 = 1 0xF1				

MOV 0 1 1 R3 R0) 3258		R3 = L	SL RO = 0×	(1EO					
MOV 1 1 1 R4 R0) 3660		R4 = 0	R4 = 0x1E1						
MOV 0 1 2 R5 R0) 32A8		R5 = L	SR R0 = 0	k78					
LDI 0	0000		R0=0	R0=0						
LDR 0 0 R6 R0	1030		R6 = 0	R6 = 0xFFF0						
MOV 3 1 2 R6 R6	5 3EB6		R6 = A	SR 0xFFF	D = 0xFFF8]				
STP	B000									
00F0 3008	3610	3258	3660	32A8	0000	103				

3EB6 B000



JMP

- As the conditions for jump need to determine if sload is loaded, the decision was made to put the entire decoder in the ALU
 - This makes it easier and less messy, so multiple signals do not need to leave each block and result in longer time to compile and test

JMP ALU logic:

	// conditional operators - compares Rd and comparator (in cond) - used for if jump should occur wire eq = (rddata == cond); // Rd == comparator wire mi = (rddata < cond); // Rd < comparator
	reg jmp_cond; // change jmp cond to determine if jmp will occur depending on comparison and jump condition in instruction word
	// PC and RAM control signals assign pc_sload = execl & jmp_cond; assign pc_cnt_en = execl & -(jmp_cond stp write_next_stp); assign ram_wren = execl & (str stmfd);
	// determine alusum - need to change opcodes Jalways @(*) begin case (op) 4'b0001, 4'b0010 : alusum = sign ? (rsdata - {11'b0,offset}) : (rsdata + {11'b0,offset}); // LDR and STR: calculate daddr
	<pre>4'b0100 : begin</pre>
1	

Did not work - pc_sload and pc_cnt_en were undefined during exec1

JMP test

Instruction mem:

РС	Instruction	Machine code	Action						
0	LDI 0x1	0001	R0=0x1						
1	MOV 0 0 0 R1 R0	3008	R1 = R0 = 0x1						
2	LDI 0x5	0005	R0 = 0x5						
3	JMP 0 0 R1 R0	4008	JMP 5 // set PC=5						
4	LDI 0xFFF	OFFF	R0=0xFFF // JMP not working						
5	LDI 0x8	0008	R0 = 0x8						
6	JMP 1 1 R1 R0	4448	JEQ 8 // set PC=8						
7	LDI 0xFFF	OFFF	R0=0xFFF // JEQ not working						
8	LDI 0xB	000B	R0=0xB						
9	JMP 2 2 R1 R0	4888	JMI 11 // Set PC=11						
10	LDI 0xFFF	OFFF	R0=0xFFF // JMI not working						
11	LDI 0xE	000E	R0=0xE						
12	JMP 3 0 R1 R0	4C08	JMB 14 // Set PC=14						
13	LDI 0xFFF	OFFF	R0 = 0xFFF // JMB not working						
14	STP	B000							
0001 3 000B 4	008 0005 40 888 0FFF 00	08 0FFF 00 0E 4C08 0	008 4448 0FFF FFF B000						

	Name	Value at 0 ps	0 ps 40.0 ns 0 ps	80.0 ns 120.	0 ns 160.0 ns	200.0 ns	240.0 ns	280.0 ns	320.0 ns	360.0 ns	400.0 ns	440.0 ns	480.0 ns	520.0 ns	560.0 ns	600.0 ns	640.0 ns	680.0 ns	720.0 ns	760.0
in_	CLOCK	B 0											лп	лл	лп		лл	лл	лл	
<u>Sut</u>	fetch	B 1																		
≌ t	exec1	B 0																		
*	> daddr	н 000	000	001										D	00					
*	> dout	н 0000	0004	X 00AB											0004					
**	> instr_ad	H 000	000 X 001	X 002 X	003 X 005	5 X 006	X 008	X 009	X 00B	X 00C	X							00E		
**	> instr	H 0000	20 0001	3008 0005	X 4008 X	0008	4448	000B	4888	000E	4C08							80	0	
**	> port1addr	B 000	000	001 X 000	χ 001 χ	000 X	001 X	000 X	001 X	000 X	001 X							00	0	
out	ldi_sel	B 1																		
out	wenout	B 0																		
out	wen	B 0																		
out	carryen	B 0																		
out.	carrysta	B 0								_										
out.	write_n	B 0																		
5	> write_n	B 0000	0000	X 0001 X	0000 X			0001				X							0000	
<u>eut</u>	write_n	B 0																		
*	> rdin	н 0000	0001	X 0005	X 0004 X	0008	0004	000B	0004	000E								0004		
*	> q1	н 0000	0000 X0001 X00	100X 0001 X00	05 <u>0001</u>	0005 (0008)	0001 (000	08X000BX	0001 (000	DBX000EX	0001							00)E	
**	> q2	H 0000	0000	0001 X 0000	0005	X 000)8 X	0000	000B	0000								000E		
**	> R0out	H 0000	0000	0001	0005	_X	8000	X	000B	х								000E		
**	> R1out	H 0000	0000	Х										0001						
₩	> R2out	H 0000											0000							
₩	> R3out	H 0000											0000							
*	> R4out	H 0000											0000							
*	> R5out	H 0000											0000							
8	> R6out	H 0000											0000							
5	> R7out	H 0000											0000							
out.	pc_cnt_en	B 0																		
<u>eut</u>	pc_sload	B 0																		
<u>Sut</u>	ram_wren	B 0																		

ADD/SUB

Multi-register Add/Sub test (from ARMish testing)

Instruction mem:		
Instruction	Machine code	Action
SUB 0 1 R1 R3	620B	
SUB 2 1 R0 R2	6A02	R0:R1 := R0:R1 - R2:R3 - 1
ADD 1 1 R1 R3	560B	
ADD 2 1 R0 R2	5A02	R0:R1 := R0:R1 + R2:R3 + 1
LDI 0x1	0001	R0=0x1
MOV 0 0 R3 R0	3018	R3=0x1
MOV 0 0 R2 R0	3010	R2=0x1
SUB 1 1 R1 R3	660B	
SUB 2 1 R0 R2	6A02	R0:R1 := R0:R1 - R2:R3
ADD 0 1 R1 R3	520B	
ADD 2 1 R0 R2	5A02	R0:R1 := R0:R1 + R2:R3
STP	B000	

	Name	Value at 0 ps	0 ps 40.0 ns 80. 0 ps	0 ns 120.0 ns	160.0 ns	200.0 ns	240.0 ns	280.0 ns	320.0 ns	360.0 ns	400.0 ns	440.0 ns	480.0 ns	520.0 ns	560.0 ns	600.0 ns	640.0 ns	680.0 ns	720.0 ns	760.0 ns
in_	CLOCK	B 0	Ľппп		пп		лл	пп						пп						
out	fetch	B 1						TH I												
out	exec1	B 0		Fir	F F	F F	ŦΕ	ΞĒ	ΞĒ	F F	FF	F F	7 F	ŦΕ	F F	F F	7 F	ΞF	F F	ŦÆ
-	> daddr	H 000	OX FFF X FFE X FFF X F	FE X 000 X 001 X 0	00 X 001 X	000 X	001	X FF	F X FFE X F	FF X FFD X O	00 001	X 003 X							000	
-	> dout	H 0000											00	00						
-	instr_ad	H 000	000 X 001 X	002 X 003	χ 004	X 005	χ 006	X 007	χ 008	X 009	X 00A	X							00B	
-	> instr	н 0000	00 620B X 6A02	X 560B X	5A02	0001	3018	3010 🗙	660B X	6A02	520B	5A02							B000	
-	> port1addr	B 000	(0) <u>(</u> 001 <u>(</u> 000	χ 001 χ	000	×	011	010 X	001 X	000 X	001 X								000	
out	ldi_sel	B 1																		
<u>eut</u>	wenout	B 0																		
out	wen	B 0																		
out	carryen	B 0																		
out	carrysta	B 0																		
<u>eut</u>	write_n	B 0																		
*	write_n	B 0000	0000 0001	0000 X 0001	X	0000	0011	X 0010	0001	0000	X 0001	X							0000	
eut	write_n	B 0																		
**	> rdin	H 0000	00 FFFF FFFE FFFF	FEX0000X0001X0	000	00	01	XFF	FFXFFFEXF	FFXFFFDX00	00 0001	0003							0000	
-	q1	H 0000	0000 XFFFF X0000 X	FFFF X0000 F	FFFX 0000	0001 00	00\0001\000	00\0001\00	00XFFFFX00	001 FFFF		FF							0001	
8	q2	H 0000		0000												0001				
85	R0out	H 0000	0000 X	FFFF	0000	X		0001		X	FFFF	X							0001	
*	R1out	H 0000	0000 FFFF	X		0000			X	FFFF	X							000	D	
*	R2out	H 0000		000	00			X									0001			
-	R3out	H 0000		0000			X									000	1			
#	R4out	H 0000											00	00						

MUL

Instruction	Machine code	Action
LDI 0x3	0003	
MOV R1 R0	3008	
LDI 0x4	0004	
MOV R2 R0	3010	
MUL R2 R1	7011	
LDI 0x0	0000	
LDI 0x0		
STP	B000	

• Need to make sure regfile is not written during MUL

	Name	Value at 0 ps	0 ps 40 0 ns 80 0 ns 120 0 ns 160 0 ns 200 0 ns 240 0 ns 220 0 ns 320 0 ns 360 0 ns 400 0 ns 440 0 ns 480 0 ns 520 0 ns 560 0 ns 660 0 ns 640 0 ns 680 0 ns 720 0 ns 760 0 ns 800 0 ns 0 ps
in	clock	в 0	
<u>sut</u>	fetch	B 1	
out	exec1	B 0	║╻╪┑┊┍╪┑┊┍╪┑┊┍╪┑┊┍╪┑┊┍╪┑┊┍╪┑┊┍╪┑┊┍╪┑┊┍╪┑┊┍
-	> daddr	H 000	<u> 000 X 003 X 000 X 004 X 000 000 000 </u>
-	> dout	н 0000	0000
*	> instr_ad	H 000	E000 X 001 X 002 X 003 X 004 X 005 X 006 X 007 X 008 X 009 X 004 X 008 X 007 X 000 X E00E E00E
34	> instr	H 0000	90X 0003 X 3008 X 0004 X 3016 X 7011 X 0000 8000
8	> port1addr	B 000	K 000 X 001 X 000 X 010 X 000
2	ldi_sel	B 1	
<u>sut</u>	wenout	B 0	
out	wen	B 0	
<u>sut</u>	carryen	B 0	
3	carrysta	B 0	
<u>eut</u>	write_n	B 0	
*	> write_n	B 0000	K 6000 X 0001 X 0000 X 0010 X
<u>eut</u>	write_n	B 0	
*	> rdin	H 0000	90X 0003 X 0004 X 0000
*	> dmemin	H 0000	
*	> q1	H 0000	
8	> q2	H 0000	K 0000 X 0003 X 0004 X 0003 X 0004 X 0003 X 0004 X
*	> R0out	H 0000	
*	> R1out	H 0000	K 0000 X 0003
8	> R2out	H 0000	K 0000 X 0004
**	> R3out	H 0000	6000
*	> R4out	H 0000	0000
*	> R5out	H 0000	0000
8	> R6out	H 0000	0000
8	> R7out	H 0000	0000
*	> mul_res_1	H 0000	6000 X6000X6000X6000X6000X600X600X600X60
*	> mul_res_2	H 0000	<u>x 0000 X0000X</u>
out			

Largest Multiplication: Data RAM: 0xFFFF

Instruction	Machine code	Action
LDR 0 0 R1 R0	1008	
MUL R1 R1	7009	
LDI 0x0	0000	
LDI 0x0		
STP	B000	

	N	ama	Value at	0 ps 40.0) ns 80.0 i	ns 120.0	ns 160.0	ns 200,0 r	s 240.0 ns	280.0 ns	320.0 ns	360.0 ns	400.0 ns	440.0 ns	480.0 ns 520.0 ns	560.0 ns	600.0 ns 640	0.0 ns	680.0 ns	720.0 ns	s 760.0 r
		unic	0 ps	0 ps រ																	
#	> da	ıddr	H 000												000						
*	> do	ut	H 0000	20											FFFE						
8	> in	str_ad	H 000	000	001	X	002 (003 0	04 005	006	X 007	X 008	X 009	X 00A	X						00B
8	> in	str	H 0000	00 1008	X	009	X				0000				X						B000
#	> po	rt1addr	B 000	0	001		X									000					
out	ld	sel	B 1																		
out	w	enout	B 0												1						
out	w	en	B 0				HL.	HL													
out	ca	irryen	B 0																		
out	CE	irrysta	B 0																		
out	w	rite_n	B 0					H											1 📂	1 1	
-	> w	rite_n	B 0000	0000	1001	0001										0000				_	
out	w	rite_n	B 0		H																
-	> rd	in	H 0000	DOX FF	FF X					0000					X						FFFF
-	> dr	nemin	H 0000	0000	X	FFF	χ									0000					
	> a1		H 0000	0000	Ŷ	FFF	Ŷ									0000					
	> 02		H 0000	0000	Ŷ	FFF	Ŷ									0000					
-	> R)out	H 0000				^								0000						
	> R	lout	H 0000	0000	Y										FFFF						
	> 80	Pout	H 0000		~										0000						
944		Rout	H 0000												0000						_
-	> D/	lout	H 0000												0000						_
944	5 0	iout	H 0000				_						_		0000						
944		Rout	H 0000												0000						
-	> 10	Tout	H 0000												0000						
-	2 8	d are d	11 0000		0000	Veres				07							000			+++	
- 	2 0	a_res_1	H 0000		0000		Veccverr	VEDEEVER	VECDE VECEE VE	A							000				_
-	> m	J_res_2	H 0000		0000	—											rrr				
-	po	_cnt_en	RO		_			┦╎┞┷													
-	po	_sload	B 0																		_
-ut	ra	m_wren	B 0																		

Storing MSB and LSB into regfile

Instruction	Machin	e code	Action		
LDR 0 0 R1 R0	1008				
MUL R1 R1	7009		R1 * R1		
LDI 0x0	0000				
LDI 0x0					
LDI 0x0					
LDI 0x0					
LDI 0x0					
LDI 0x0					
LDI 0x0					
LDI 0x0					
MOV 0 0 3 R3 0	30D8		R3 = LSE	Bs of prod	uct
MOV 0 0 3 R4 1	30E1		R4 = MS	Bs of pro	duct
STP	B000				
1008 7009 0000 0000	0000 30D8	0000 30F1	0000 B000	0000	000

	Name	Value at	0 ps 40.0 ns 80.0 r	ns 120.0 ns 160.0 ns	200.0 ns 240.0 ns	280.0 ns 320.0 ns	360.0 ns 400.	0 ns 440.0 ns	480.0 ns 520.0 ns	560.0 ns 600	.0 ns 640.0 ns	680.0 ns	720.0 ns	760.0 ns 80
		0 ps	0 ps											
in	clock	B 0												
-	fetch	B 1												
-	exec1	B 0												
#	> daddr	H 000			000			X	001 X FFE X					000
۲	> dout	H 0000	90X		FFFF				0000	X				FFFF
۳	> instr_ad	H 000	000 001	X 002 X 003	X 004 X 005	X 006 X 007	Y 008 X	009 X 00A	Х 00В Х					00C
	> instr	H 0000	00 1008 7	7009 X		0000		X	30D8 X 30E1 X					8000
۲	> port1addr	B 000	0) 001	X		000		Х	011 X 100 X					000
out 🍆	ldi_sel	B 1												
out 🍆	wenout	B 0												
out 🍆	wen	B 0												
out	carryen	B 0												
out	carrysta	B 0												
out	write_n	B 0												
#	> write_n	B 0000	0000 1001	0001		0000			X 0011 X 010	ю Х				0000
out	write_n	B 0												
#	> rdin	H 0000	00 FFFF	X X		0000		X	0001 X FFFE X	0000				FFFF
*	> dmemin	H 0000	0000 X F	FFF		0000			X0001X0000XFFFEX					0000
#	> q1	H 0000	0000 X F	FFF X		0000			X0001X0000XFFFEX					0000
#	> q2	H 0000	0000 X F	FFF X		0000			X FFFF X					0000
#	> R0out	H 0000							0000					
#	> R1out	H 0000	0000 X						EFFF					
-	> R2out	H 0000							0000					
-	> R3out	H 0000			0000				X				0001	
-	> R4out	H 0000			00	00			X					FFFE
#	> R5out	H 0000							0000					
#	> R6out	H 0000							0000					
#	> R7out	H 0000							0000					
#	> mul_res_1	H 0000		0000		tx++T+++					0001			
-	> mul_res_2	H 0000		0000							FFFE			
out					+	┉╵┝┷┥╹┍	+							

Testing add and multiply:

Instruction	Machine	e code	Action			
LDR 0 0 R1 R0	1008					
MUL R1 R1	7009		R1 * R1			
LDI 0x1	0001					
MOV R3 R0	3018					
LDI 0x1	0001					
MOV R4 R0	3020					
LDI 0x0	0000					
LDI 0x0						
LDI 0x0						
LDI 0x0						
MOV 0 0 3 R3 0	30D8		R3 = LSE	s of prod	uct+1	
MOV 0 0 3 R4 1	30E1		R4 = MS	Bs of pro	duct+2	
STP	B000					
1008700900000000	0001 30D8	3018 30E1	0001 B000	3020	0000	0000

	Nar	me	Value at	0 ps 40.0 ns	80.0 ns	120.0 ns	160.0 ns	200.0 ns	240.0 ns	280.0 ns	320.0 ns	360.0 ns	400.0 ns	440.0 ns	480.0 ns	520.0 ns	560.0 ns	600.0 ns	640.0 ns	680.0 ns	720.0 ns	760.0 ns	800.0 ns
			0 ps	u ps																			
in -	cloc	ck	B 0																	ЦЦ			
°ut 🍆	fetc	ch	B 1																				╧┤┊╞
eut	exe	ec1	B 0																				
۳	> dad	idr	H 000		000		X_	001 X	000 X	001 X		000		Xo	02 X 003 X F							000	
۳	> dou	ıt	H 0000	00X	FFF	F		X 0000	X FFFF	X 0000	_X		FFFF		_X	0000	Х					FF	FF
8	> inst	tr_ad	H 000	000 X	001	X 002	X 003	X 004	X 005	X 006	<u>X 007</u>	<u>X 008</u>	X 009	X 00A	X 00B	_X						DOC	
8	> inst	tr	H 0000	00X 1008 X	7009	X_	0001 X	3018 X	0001 X	3020 X		0000		X	30D8 X	30E1 X						B000	
*	> port	t1addr	B 000	0X	001		000 X	011	000	100		000		X	011	100						000	
2 ^{ut}	ldi_s	sel	B 1																				
<u>eut</u>	wer	nout	B 0																				
<u>eut</u>	wer	n	B 0																				
2	carr	ryen	В 0																				
<u>eut</u>	carr	rysta	B 0																				
out	writ	te_n	B 0																				
8	> writ	te_n	B 0000	0000 1001	X	0001	0000	0011	X 0000	0100	X		0000		0011	0100	X					00	00
<u>out</u>	writ	te_n	B 0																				
*	> rdin	n	H 0000	00	FFFF			0001		X		0000		X0(002 (0003 (FF	FFXFFFDX00	x (0					FF	FF
#	> dme	emin	H 0000	0000 X	FFFF	X00	00\0001\00	00 00	01 🗙 🛛 🕹	000 0001	X	00	000	Xoo	001 0002 00	IO1 FFFF						0000	
8	> q1		H 0000	0000	FFFF	X00	00\0001\00	00 00	01 (00	00 0001	X	00	00	X00	001\0002\00	01 FFFF						0000	
*	> q2		H 0000	0000 X		FFFF	X	0001	FFFF X	0001	X		0000		X	FFFF X						0000	
*	> R0o	out	н 0000		0000		X		0001		X								000	0			
8	> R1o	out	н 0000	0000 X												FFFF							
#	> R2o	out	н 0000												00	100							
*	> R30	out	н 0000		0000			X			0001				X						000	2	
*	> R4o	out	H 0000			000)			X			0001			X						FFFF	
*	> R50	out	н 0000												00	100							
*	> R6o	out	H 0000												00	00							
8	> R7o	out	н 0000												00	100							
-	> mul	_res_1	H 0000			000)			X									0001				
-	> mul	_res_2	H 0000			0000)			X									FFFE				
out															-								

BL

Test if link register loading works

Instruction mem:

РС	Instruction	Machine code	Action
0	BL 2	8002	For now just sets R6=0x1
1	STP	B000	

			Value at	0 ps 40.0 ns	80.0 ns	120.0 ns	160.0 ns	200.0 ns	240.0 ns	280.0 ns	320.0 ns	360.0 ns	400.0 ns	440.0 ns	480.0 ns	520.0 ns	560.0 n
		Name	0 ps	0 ps													
in		CLOCK	B 0														
out		fetch	B 1														
out		exec1	B 0														
5	>	daddr	H 000	0 001 002												000	
5	>	dout	H 0000												00	00	
5	>	instr_ad	H 000	000												001	
5	>	instr	H 0000	00 8002												B000	
5	>	port1addr	B 000	0/ 110 /												000	
out		ldi_sel	B 1														
out		wenout	B 0														
out		wen	B 0														
out		carryen	B 0														
out		carrysta	B 0														
out		write_n	B 0														
#	>	write_n	B 0000												00	00	
out		write_n	B 0														
#	>	rdin	H 0000	000010002												0000	
#	>	q1	H 0000	0000 0001												0000	
#	>	q2	H 0000												00	00	
*	>	R0out	H 0000												00	00	
5	>	R1out	H 0000												00	00	
#	>	R2out	H 0000												00	00	
5	>	R3out	H 0000												00	00	
5	>	R4out	H 0000												00	00	
5	>	R5out	H 0000												00	00	
5	>	R6out	H 0000	0000												0001	
5	>	R7out	H 0000												00	00	
out		pc_cnt_en	B 0														
out		pc_sload	B 0														
out		ram_wren	B 0														

Test if full BL works

РС	Instruction	Machine code	Action
0	BL 2	8002	For now just sets R6=0x1
1	STP	B000	
2	LDI 0xFFF	OFFF	R0=0xFFF
3	JMP 0 0 R0 R6	4006	PC=0x1



STACK

LDMFD and STMFD

Test:					
Instruction	Machine code	Action			
LDI 0x1	0001				
MOV R1 R0	3008	R1=0x1			
LDI 0x3	0003				
MOV R6 R0	3030	R6=0x3			
LDI 0x2	0002	R0=0x2			
PUSH R1	A00F / A039	PUSH 0x1			
PUSH RO	A007 / A038	PUSH 0x2			
PUSH R6	A037 / A03E	PUSH 0x3			
POP R2	9017	R2=0x3			
POP R3	901F	R3=0x2			
POP R4	9027	R4=0x1			
STP	B000				
0001 3008 9017 901F	3 0003 303 - 9027 B00	30 0002 00	A039	A038	A03E

Issue:

- Stack pointer is not changing
 - Rdin is rd-1
 - Also need to have wen during stmfd
- We also need aluout to be selected for din
- Thus, ldmfd instructions cannot be done in parallel like ldr as something is written every cycle
- Now it is writing R6, instead of LDMFD and STMFD
- Rdin should be



- Stmfd works but not ldmfd
- 2 Idmfd instructions do not work after each other as write_next_stp is stuck at 1

Issue:

	Name	Value at 0 ps	0 ps 40.0 ns 80.0 0 ps) ns 120.0 ns	160.0 ns	200.0 ns	240.0 ns	280.0 ns	320.0 ns	360.0 ns	400.0 ns	440.0 ns	480.0 ns	520.0 ns	560.0 ns	600.0 ns	640.0 ns	680.0 ns	720.0 ns	760.0 ns	800.0 ns	840.0 ns
in .	CLOCK	в 0	Горор	ппг		пп	пг	пп	пп	пп	пп	пп	пп	пп	пп	пп	пг	пп	пп			
	fetch	B 1	16.0-9.0-				TH.							7-FL	H.		H					Ξ Π .
	exec1	B 0															1					
⇔⇒	daddr	H 000	000 X 001	X 000 X	003 X	000 X FF	F FFE	X FFD	XFFC XFF	D X 000 X 00	3 FFD	X 000 X 003)	FFD	X 000 X						000		
# >	dout	H 0000			0000)				X0003X	0000	0003 🗶 00	00 X	0003						0000		
# >	instr_ad	H 000	000 X 001 X	002 X 003	X 004	X 005	X 006	X 007	X 008	X	009	X 01	0A	X						00B		
3 >	instr	H 0000	00 0001 3008	X 0003 X	3030	0002	A039 X	A038	A03E	9017	901F	X	9027							B000		
6 >	port1addr	B 000	000 X 001	χ 000 χ	110 X	000 X		111		X 010	X 111	X 011	(111	X 100	X					000		
	ldi_sel	B 1																				
	wenout	B 0																				
out	wen	B 0																				
out	carryen	B 0																				
out	carrysta	B 0																				
	write_n	в 0																				
6 >	write_n	B 0000	0000	0001 X 0000	0110	0000	X	0111		1010	0011	1011	0100	X 1100	X					000)	
	write_n	B 0																				
# >	rdin	H 0000	20 0001	X 0003	X	0002 X0F	FFX OFFE	V OFFD	OFFCOF	FD 0003 000	00 OFFD	0003(0000)	OFFD	(0003)						0000		
# >	dmemin	H 0000	0000 0001 0000	0001 0003 00	000 0003	X0002X	0001 X	0002 X	0003 X0F	FDX0000X000	03 OFFD	0000 0003	OFFD	000000000	3)					000	2	
# >	q1	H 0000	0000 0001 0000	0001 0003 00	000 0003	0002	00X OFFF	X OFFE	X OFFD	000000000	03X OFFD	0000 0003	OFFD	000000000	3)					000	2	
# >	q2	H 0000	0000 X 0001	χ 0000 χ	0003 🗙	0000 X	0001 X	0002 X	0003			OFFD		X						0002		
65 >	R0out	H 0000	0000 0001	X	0003											0002						
# >	R1out	H 0000	0000 X											0001								
# >	R2out	H 0000			0	000				X000	13							0000)			
# >	R3out	н 0000				0	000					X0003)							000)		
₩ >	R4out	н 0000					(0000						X000	зХ					000)	
# >	R5out	H 0000											000	00								
# >	R6out	H 0000	0000)	X										0003							
6 >	R7out	H 0000		0000			X OFFF	X OFFE	X								OFFI)				
	pc_cnt_er	B 0										٦		7								
		~ ~																				

• STMFD works but in LDMFD, stack pointer is not updating

Issue:

			Makes at	0 ps 40.0 ns 80.0 ns	120.0 ns	160.0 ns	200.0 ns	240.0 ns	280.0 ns	320.0 ns	360.0 ns	400.0 ns	440.0 ns	480.0 ns	520.0 ns	560.0 ns	600.0 ns	640.0 ns	680.0 ns	720.0 ns	760.0 ns	800.0 ns	840.0
	Nar	me	0 ps	0 ps		•	•				•							•	•	•	•	•	
in_	CLC	оск і	в 0			uп								urur		лл	лл					лл	п
<u>eut</u>	fetc	ih I	B 1																				
out	exe	c1 I	B 0																				٦
*	> dad	dr I	H 000	000 001	000 X	003 X	000 X FF	FX FFE	X FFD	XFFC XFF	D X 000 X 00	I3 FFE	X 000 X 01	02 FFF	X 000 X						000		
*	> dou	t I	H 0000			000)				0003	0000 X	0002	0000 X	0001 X						0000		
*	> inst	r_ad I	H 000	000 X 001 X 0	003 003	X 004	X 005	X 006	X 007	X 008	X	009	X	00A	X						00B		
*	> inst	r I	H 0000	00 0001 3008	0003	3030 🗶	0002	A039	A038 X	A03E	9017 X	901F		9027	X						B000		
8	> port	1addr I	B 000	000 001	000 X	110 X	000 X		111		X 010	X 111	X 011	X 111	X 100	X					000		
<u>sut</u>	ki_s	sel I	B 1																				
<u>out</u>	wer	nout I	B 0																				
out	wer	n I	в 0																				
out	carr	ryen I	в 0																				
out	carr	rysta I	в 0																				
out	writ	le_n I	B 0																				L
*	> writ	le_n I	B 0000	0000 X 00	01 0000	0110	0000	X	0111		1010	0011	X 1011	0100	1100	X					0000		
out	writ	ie_n I	B 0																				
**	> rdin		H 0000	000 0001	0003	X	0002 XFF	FFX FFFE	FFFD	FFFC	E 0003 00	00 FFFF	0002	0000	X0001X						0000		
**	> dme	emin I	н 0000	0000 0001 0000 00	01 (0003)00	000 0003	X0002X	0001	0002	0003 XFF	DX0000X00	03 FFFE	X0000X00	02 FFFF	X0000X000	иχ					0002		
**	> q1		H 0000	0000 0001 0000 00	01 (0003)00	000 0003	0002 000	00 FFFF	X FFFE	FFFD	00000000	03 FFFE	00000000	02 FFFF	X0000X00	иχ					0002		
**	> q2		H 0000	0000 0001	0000	0003 X	0000 X	0001 X	0002 X	0003 XFF	σχ	FFFE	X	FFFF	X0000X						0002		
*	> R00	ut I	н 0000	0000 0001	X	0003	X										0002						
*	> R10	ut I	н 0000	0000 X											0001								
*	> R20	ut I	н 0000			(1000				X00	03							0000)			
*	> R3o	ut I	н 0000				0	000					X00	102						0000)		
*	> R4o	ut I	н 0000					0	000						X00	иХ					0000		
*	> R5o	ut I	н 0000											00	00								
*	> R6o	ut I	н 0000	0000		X										0003							
**	> R7o	ut I	н 0000		0000			FFFF	X FFFE	X FFFD	X	FFFE	X	FFFF	X						0000		
<u>sut</u>	pc_	cnt_en I	B 0																				
<u>sut</u>	pc_	sload I	в 0																				
out								$\neg = \Box$	- H														(1 I

- Works but the registers are rewritten to 0 as rdin becomes 0
- This is because dout changes to 0 for the instructions that need to wait when write_next_stp is enabled
 - Daddr switches to 0 during fetch as aluout is reset
- Saw that when write_next_flag is on, wen should be the inverse of wenout, so that it is not on in that last cycle where 0 is written is at daddr, so could use an XOR gate to carry out selective inversion
 - May cause problems when pipelining

 Added stp to wenout, this may cause problems for other instructions writing during stp - check later

Working:

_												
	Name	Value at 0 ps	0 ps 40.0 ns 80.0 ns 120.0 ns 160.0 ns 0 ps	200.0 ns 240.0 ns 28	80.0 ns 320.0 ns	360.0 ns 400.0 ns	440.0 ns 480.0 ns	s 520.0 ns 560.0 ns	600.0 ns 640.0 n	s 680.0 ns 720.0	ns 760.0 ns	800.0 ns 840
in_	CLOCK	B 0							uuu		ллл	
out	fetch	B 1										
out	exec1	B 0						+				
-	> daddr	H 000	000 X 001 X 000 X 003 X	000 XFFF X FFE X	FFD X FFC X FF	D X 000 X 003 X FFE	X 000 X 002 X FF	F X 000 X			000	
84	> dout	H 0000	0000			X0003X 0000 X	0002 (0000)	0001			0000	
-	> instr_a	н ооо	000 X 001 X 002 X 003 X 004	X 005 X 006 X	007 X 008	X 009	X 00A	X 00B	X			00C
-	> instr	H 0000	00X 0001 X 3008 X 0003 X 3030 X	0002 X A039 X A03	18 X A03E X	9017 X 901F	χ 90;	27 X 0000	X			B000
-	> port1ad	fr B 000	000 X 001 X 000 X 110 X	000 X	111	X 010 X 111	X 011 X 11	1 X 100 X			000	
out	ldi sel	B 1										
out	wenout	B 0										
out	wen	B 0									HTH	
out	carryer	B 0										
out	carryst	B0										
out	write_n	B0										
-	> write n	B 0000	0000 X 0001 X 0000 X 0110	X 0000 X	0111	X 1010 X 0011	X 1011 X 010	10 X 1100 X			0000	
out	write n	B0										
944	> rdin	H 0000	00X 0001 X 0003 X	0002 XFFFFX FFFE X	FFFD XFFFCXFFF	FE 10003 0000 FFFF	X0002X 0000	X0001X			0000	
944	> dmemin	H 0000	0000 X0001X0000X 0001 X0003X0000X 0003	X0002X 0001 X 000	2 X 0003 XFFF	FDX0000X0003X FFFE	X0000X0002X FFI	F X0000X0001X 0002	Y			0000
94	> at	H 0000	0000 X0001 X0000 X 0001 X0003 X0000 X 0003	Y0002 X0000 Y FFFF Y	FFFE Y FFFD	X0000 X0003 X FFFE	X0000 X0002 X FF	F Y0000 Y0001 Y 0002	F¥ H H H			0000
	> 62	H 0000	0000 Y 0001 Y 0000 Y 0003 Y	0000 ¥ 0001 ¥ 000	2 Y 0003 YEFE	FDY FFFF	FFFF	Y0000Y 0002				0000
944	> R0out	H 0000	0000 X 0001 X 0003			0002			+ ` +++++			0000
944	> Plout	H 0000	0000 Y	^				0001				
out	> P2out	H 0000		000		- v				0003		
944	> P3out	H 0000		0000			V V			00	102	
-	> Réout	H 0000	Ç <u></u>	0000				- v			0001	
944	DEaut	H 0000						0000				
944	> Rout	H 0000	0000						13			
tup:	> R0001	11 0000	0000	V EEEE V		V	V EDEE				0000	
ent.	> R/out										000	
	pc_cnt	en Du										
-	pc_sloa	0 80	$\mathbf{H} \rightarrow \mathbf{H} \rightarrow $			+++++++++++++++++++++++++++++++++++++++		+++++++++++++++++++++++++++++++++++++++	+++++++			++++++
		1 14	<u> </u>									

Issue: loading 0 in because enabled stp in wrenout
assign wenout = exec1 & (ldi | mov | add | sub | mul | bl | ldmfd | stmfd | stmfd | stp & write_next_status[3]);

Added this - may help - temporary solution - may be different when pipelining

Working:

	Name	Value at 0 ps	0 ps 40.0 ns 80.0 ns 120.0 ns 120.0 ns 200.0 ns 240.0 ns 280.0 ns 320.0 ns 380.0 ns 440.0 ns 440.0 ns 480.0 ns 520.0 ns 580.0 ns 640.0 ns 640.0 ns 680.0 ns 720.0 ns 720.0 ns 800.0 ns 640.0 ns 6 0 ps
in	CLOCK	B 0	
	fetch	B 1	║┍┶┚╤┶┚╤╲┙╤╲┙╤╲┙╤╲┙╤╲┙╤╲┙╤╲┙╤╲┙╤╲┙╤╲┙╤╲┙╤╲┙╤╲┙╤
out	exec1	B 0	
*	> daddr	H 000	x 000 X 001 X 000 X 003 X 000 XFFF X FFE X FFD XFFC XFFD X000 X003 X FFE X 000 X002 X FFF X 000
*	> dout	H 0000	\$2000 X0000 X 0000 X 0001 X 0001 X 0001 X 0001 X
*	instr_ad	H 000	600 X 001 X 002 X 003 X 004 X 005 X 006 X 007 X 008 X 009 X 00A X 008 D08
*	> instr	H 0000	800 0001 X 3008 X 0003 X 3030 X 0002 X 4039 X 4038 X 4032 X 9017 X 901F X 9027 X 9007
*	> port1addr	B 000	<u>000</u> X 001 X 000 X 110 X 000 X 111 X 010 X 111 X 011 X 111 X 100 X 000
	ldi_sel	B 1	
out	wenout	B 0	
	wen	в 0	
out	carryen	в 0	
<u>eut</u>	carrysta	в 0	
out	write_n	B 0	
*	write_n	B 0000	K 0000 X 0001 X 0000 X 0110 X 0000 X 0111 X 1010 X 0011 X 1011 X 0001 X 1000 X 1000 X 0000 0000
<u>sut</u>	write_n	B 0	
*	> rdin	H 0000	00X 0001 X 0002 XFFFYX FFFE X FFFD XFFFYX 0002 0000X FFFF X0002 0000 X 6001X 0000
*	dmemin	H 0000	0000 X0001X0000X 0001 X0003X0000X 00003 X0002X 0001 X 0002 X 0003 XFFFDX0000X0003X FFFE X0000X0002X FFFF X0000X0001X 0002
*	⊳ q1	H 0000	K doody Xaadah Xaadaa Xaad
*	q2	H 0000	0000 X 0001 X 0000 X 0001 X 0000 X 0001 X 0002 X 0003 XFFFDX FFFE X FFFF X0000X 0002
#	R0out	H 0000	(dood X dood X dood X
*	R1out	H 0000	C 0000 X 0001
84	R2out	н 0000	0000 X 0003
*	R3out	H 0000	0000 X 0002
8	R4out	H 0000	0000 X 0001
5	RSout	H 0000	0000
**	R6out	H 0000	0000 X 0003
*	R7out	H 0000	0000 X FFFF X FFFE X FFFE X FFFF X 0000
out 📥	pc_cnt_en	B 0	
	pc_sload	B 0	

Test doing LDI after this to make sure it works after POP

• Works with LDI

in_	CLOCK	B 0		
<u>sut</u>	fetch	B 1		
<u>sut</u>	exec1	B 0		
8	> daddr	H 000	L D00 X 001 X 000 X 003 X 000 XFFFX FFE X FFD XFFC XFFD X000 X003 X FFE X 000 X002 X FFF X 000 X	000
#	> dout	H 0000		0000
#	> instr_ad	Н 000	CO01 X 002 X 003 X 004 X 005 X 006 X 007 X 008 X 009 X 004 X 008 X	000
#	> instr	H 0000	88X 0001 X 3008 X 0003 X 3030 X 0002 X Ad39 X Ad38 X Ad38 X 9017 X 9017 X 9027 X 0000 X	8000
#	> port1ad	fr B 000	<u>000 X 001 X 000 X 110 X 000 X 111 X 010 X 111 X 011 X 111 X 100 X</u>	DOG
3 ^{ut}	ldi_sel	B 1		
<u>sut</u>	wenout	B 0		
out	wen	B 0		
out	carryen	B 0		
<u>sut</u>	carryst	B0		
<u>sut</u>	write_n	B0		
#	> write_n	B 0000	6000 X 0001 X 0000 X 0110 X 0000 X 0111 X 1010 X 0011 X 1011 X 100 X 1100 X	0000
out	write_n	B0		
#	> rdin	H 0000	00X 0001 X 0002 X 0002 XPFFFX FFFE X FFFD XFFFCXFFFEX0003X0000X FFFF X0002X 0000 X0001X	0000
#	> dmemin	H 0000	2000 X0001 X0000X 0001 X0000X 0003 X0002X 0001 X 0002 X 0003 XFFF0X0000X0003X FFFE X0000X0002X FFFE X0000X0002X X	0000
#	> q1	H 0000	K 0000 X0001 X0000 0001 X0000X 0000 X0000X 0000 FFFF X FFFE X FFFE X0000X0000	0000
8	> q2	H 0000	0000 X 0001 X 0000 X 0000 X 0000 X 0001 X 0002 X 0003 XFFF0X FFFE X FFFF X0000X 0002 X	0000
8	> R0out	H 0000		0000
8	> R1out	H 0000	C 0000 X 0001	
#	> R2out	H 0000	0000 X 0003	
#	> R3out	H 0000	0000	0002
8	> R4out	H 0000	K 0000 X	0001
#	> RSout	H 0000	0000	
#	> R6out	H 0000	K doob X doos	
#	> R7out	H 0000	0000 X FFFF X FFFE X FFFE X FFFF X	0000
<u>sut</u>	pc_cnt_	en BO		
<u>sut</u>	pc_sloa	B 0		
out				