ODE Solver Design Reference

Team #5

April 14, 2020

Contents

List of Figures

List of Tables

Part I

Introduction

0.1 Abstract

This document is intended to be used as a comprehensive reference for the coming phases, and to be up-to-date with all future changes in the design.

0.2 Interfaces and Actions

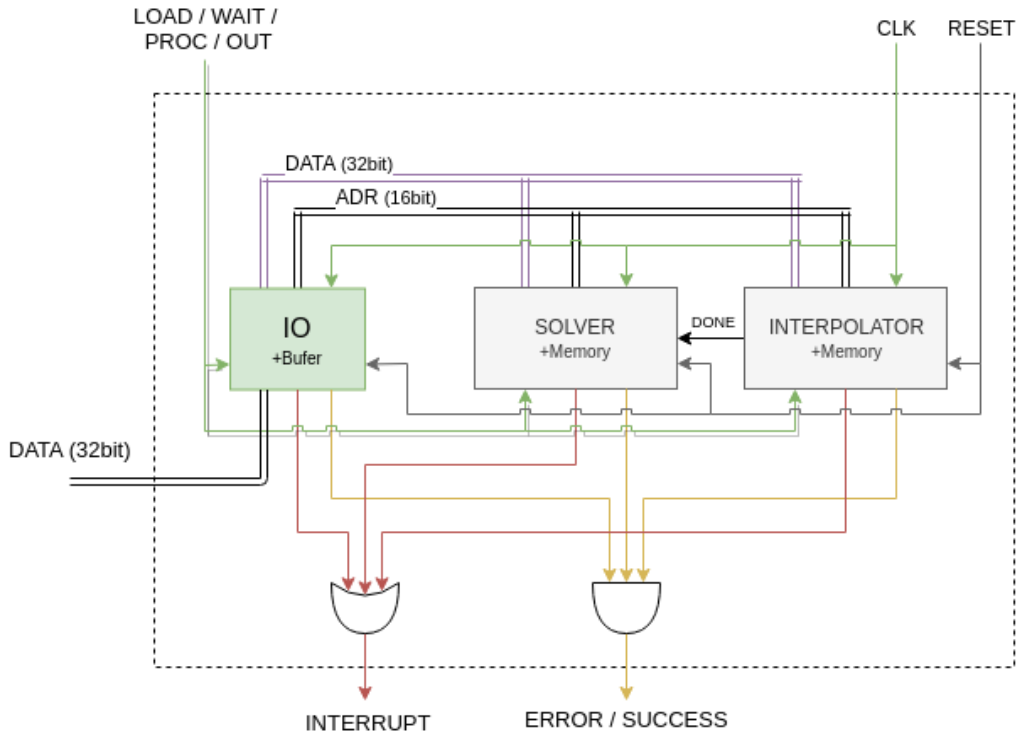

Figure 1: Overall System Design

The main module, as shown in Figure 1, has the ports listed in Subsection 0.2.1 that triggers some actions.

Ports and their actions are summarized below and detailed in the rest of the document.

0.2.1 Ports

CLK: IN

RESET: ASYNC IN

- clears all internal states of all modules:
  - IO internal buffer.
  - ERROR/SUCCESS of all modules resets to SUCCESS.
  - INTERRUPT resets to zero.
- Memory at solver and interpolator are NOT cleared.
- At next clock, CPU is expected to turn the LOAD / WAIT / PROC / OUT into LOAD state and we will start loading input again.

LOAD / WAIT / PROC / OUT (2bit): IN

Set the current major state of the machine.

- LOAD(0):
  - IO receives **compressed** data from the CPU.
  - IO decompresses data into buffer.
  - buffer is flushed into data bus with appropriate address.
  - ends when cpu finishes its data loading and switches to WAIT state.
- WAIT(1):
  - Same state as LOAD, but IO doesn't receive anymore data from CPU.
  - ends when IO flushes all its buffer and raises INTERRUP with either ERROR or SUCCESS.
- PROC(2):
  - SOLVER sends time step to calculate U at.

  - SOLVER and INTERPOLATOR work concurrently to calculate their outputs.
  - INTERPOLATOR sends DONE signal to SOLVER when it finishes the interpolated U.
  - SOLVER can request to copy the interpolated U.
  - INTERPOLATOR waits for SOLVER to send next time step.
  - ends when either SOLVER or INTERP raises INTERRUPT with either SUCCESS or ERROR.

- OUT(3):
  - IO just copies final outputs to cpu from SOLVER memory.
  - ends when IO raises INTERRUPT with either SUCCESS or ERROR.

DATA (32bit): INOUT

- Data bus between cpu and io.

INTERRUPT: OUT

- Raised from 0 to 1 when some internal module (IO / SOLVER / INTERPOLATOR) finishes its task.
- If task finished with success the ERROR / SUCCESS is set to SUCCESS, otherwise it's ERROR.

ERROR(0) / SUCCESS(1): OUT

- CPU should operate on this value ONLY when INTERRUPT is 1.
- Errors that could happen include: divide by zero, H ¿ 1, incomplete input.

0.3 Pipelining

Main module has only one pipeline with 2 stages, the solver and the interpolator. The solver calculates the next X_h while the interpolator calculates U_{h2}.

0.3.1 Possible Hazards

In Fixed Step Mode

The only problem is when the unit reachs the last X, then the upcoming h will be garbage or unwanted h, and the unit will terminate either ways.

In Adaptive/Variable Step Mode

When solver is calculating X_h, interpolator is busy calculating $U_{h/2}$, and when solver is calculating $X_{h/2}$ and X_h, interpolator will be calculating U_h*, $h*$ is the h from the algorithm equation, which is $h = \frac{0.9 \times h^2 \times L}{e}$, this U_h* might not even be used, and then, solver will interrupt the interpolator, and orders it to calculate U_{hnew}.

0.4 Parallelism

> Parallelism: having multiple instances of module M as $m_0, m_1, m_2 \ldots$, and splitting inputs $x_0, x_1, x_2 \ldots$ between them, so each module m_i takes input x_i.

Based on the previous definition, this design doesn't implement parallelism. As the solver only solves one and only one X_{out} at a given time. And the main module has one instance of the solver and interpolator.

0.5 Inter-module Communication

0.5.1 IO and CPU

- CPU orders to Load, IO reads 32bits of data.
- When CPU finishes, raises WAIT, when IO finishes that last package of data int the cpu with success.
- CPU raises PROC, so that all of other components starts acting.
- When output data is ready, CPU is interrupted with error or success, then CPU raises OUT signal which implies that it's ready to take out the resulting output from IO.

- If there's an error, CPU should know which component sent it.

0.5.2 IO and Solver/Interpolator

- IO puts the data and address on their busses, each component of them two reads that address and check if this address is within its range, if so this component reads the data and stores it.
- When Solver is ready to flush the result it raises int and success or even error.

0.5.3 Solver and Interpolator

- See Section 0.3.
- Solver is working on X_h so it needs U, solver checks on DONE signal, if raised, reads the U from the data bus, Interpolator has already placed it there.
- Then Solver places h_new at the data bus, interpolator reads it and they both work...
- and repeat.

Part II

Simulation Workflow

0.6 Input Preprocessing

This stage is the responsibility of a script that runs before the simulation:

Stage Input

JSON file that follows the format stated in the main project document [1].

Steps

- Create bit stream of the read data that follows the Input Data Structure specifications.

  NOTE: if the mode is 16 bit (fixed), then each 16bit number must be padded with 16 bits of zeros, so that the number is in the lowest part (bits $0 \ldots 15 \rightarrow$ num, $16 \ldots 31 \rightarrow 0$ padding.)

- Encode the bits following the Compression specifications.
- Collect encoding output in ASCII string, each byte in string is either '0' or '1' in ASCII format.
- When the string reaches the length of 32 bytes, push it to output file.
- If the last created string didn't reach the length of 32 bytes, complete the rest with '0' and push it to the output file.

Stage Output

- ASCII file that contains multiple lines of compressed data.
- each line has exactly 32 '0' or '1' ASCII characters.
- ONLY the ASCII characters 0 or 1 are permitted in the file and NOTHING ELSE.
- there is NO EMPTY LINE/s in the file or spaces.

0.7 Instantiating Main Module

This stage and all the next ones are the responsibility of the CPU simulation code.

CPU is a non-synthesisable HDL test-bench that:

- Instantiates the HW main module.
- Attaches the appropriate signals to the HW main module.
- Generates CLK with fixed frequency.
- Loads data into HW.
- Puts HW into PROCESS state.
- Load output out from the HW and into a file.

0.8 Input Loading

- Load the output of the former script into array of vectors each is 32bit wide that will hold one line in the file.
- Put HW at LOAD state.
- RESET for one cycle.
- For each 32bit vector in the former array:
  - At the positive edge of CLK:
    - Load vector into DATA bus.
- Load DATA with 0s.
- Wait for the positive edge of INTERRUPT signal.
- Check for ERROR / SUCCESS and only proceed if it is SUCCESS.

0.9 Processing

- Put HW at PROCESS state.
- Wait for the INTERRUPT positive edge.
- Check for ERROR / SUCCESS and only proceed if it is SUCCESS.

0.10 Output Extraction

- Put high impedance on DATA bus.
- Put HW at OUT state.
- Keep receiving data into array of vectors and outputting them into file in the same format of the input file.
- Wait for the positive edge of INTERRUPT signal.

Simulation is done!

You can turn the output into human-readable json using output-formatting script.

Part III

Specifications

0.11 Memory Mapping

The Following addresses are only meant for internal communicating between modules, and they don't need to resemble actual addresses stored at some memory.

The address loaded at the bus resembles what kind of data is on data bus or what kind of data this module should output.

If address bus is loaded with an address *Adr* that some module *M* is not assigned to, module *M* must ignore the data and address bus so the rest can communicate.

This way communication is simplified.

A column for module *M* is the action of the address taken at module *M* when it sees that address. It's either:

- *W*: **Write to** module *M*. Module *M* is expected to **read** the data bus and store data internally, so the other module *wrote* to module *M*.
- *R*: **Read from** module *M*. Module *M* is expected to **write** some data to the data bus as response to this address, so the other module **reads** from it.

0.11.1 Solver Memory Mapping

Solver module listens at the addresses listed in Table 1.

Table 1: Solver Memory Mapping

Address	A	Type	Words	Name	Description
0x0000	W	struct Header	2	Header	Includes Dimensions and modes
0x0002	W	f64	4	H	Timestep (variable step mode)
0x0006	W	f64	4	Error	Error Tolerance (variable step mode)
0x000A	W	f64[50][50]	10000	A	Matrix A
0x271A	W	f64[50][50]	10000	B	Matrix B
0x4E2A	W	f64[50]	200	X	Initial value of X
0x4EF2	R	f64[50][5]	1000	X_{out}	Final Output X
0x52DA	W	f64[50]	200	U_0	Initial U vector

0.11.2 Interpolator Memory Mapping

Interpolator module listens at the addresses listed in Table 2.

Table 2: Interpolator Memory Mapping

Address	A	Type	Words	Name	Description
0x0000	W	struct Header	2	Header	Includes Dimensions and modes
0x52DA	W	f64[50]	200	U_0	Initial U vector
0x53A2	W	f64[5]	20	T	Time points of solutions
0x53B6	W	f64[50][5]	1000	U_s	U vector at required time steps
0x579E	R	f64[50]	200	U_{int}	Interpolated U Vector
0x5866	W	f64	4	h_{new}	Time to calculate U at

0.12 Header Data Structure

Table 3 shows the structure of header data. Note the whole struct takes 2 words.

Table 3: Header Data Structure

Address	Bit	Name	Description	Datatype	Total Size
0x0000	15:10	N	Dimension of X	uint	6 bits
	9:4	M	Dimension of U	uint	6 bits
	3	Solver Mode	Fixed Step(0) or Variable Step(1)	enum	1 bit
	2:1	FPU Precision	fixed point(0), f64(1) or f32(2)	enum	2 bits
	0	NOT USED	——	——	1 bit
0x0001	15:13	T_{size}	Count of timesteps needed	uint	3 bits
	14:0	NOT USED	——	——	13 bits

0.13 Input Format

Table 4 shows the IO input structure. IO reads the header and figures out the addresses.

NOTE: if the mode is 16 bit (fixed), then each 16bit number is padded with 16 bits of zeros, so that the number is in the lowest part (bits 0 ... 15 $\rightarrow$ num, 16 ... 31 $\rightarrow$ 0 padding.)

Table 4: Inputs Specifications

Address	Type	Max Words	Name
0x0000	struct Header	2	Header
0x0002	f64	4	H
0x0006	f64	4	Error
0x000A	f64[50][50]	10000	A
0x271A	f64[50][50]	10000	B
0x4E2A	f64[50]	200	X
0x52DA	f64[50]	200	U_0
0x53A2	f64[5]	20	T
0x53B6	f64[50][5]	1000	U_s

0.14 Modules

0.14.1 IO

Role

- Receive packets of 32 bits from the CPU through *DATA* bus.
- Decompress the data.
- Send the decompressed data to the data-bus.

Ports

- INOUT: 32bit data bus with other modules.
- INOUT: 32bit data bus with CPU.
- OUT: 16bit address bus.
- IN: CLOCK.
- ASYNC IN: Reset.
- IN: 2bit Load/Wait/Process/Out.
- OUT: Interrupt to CPU.

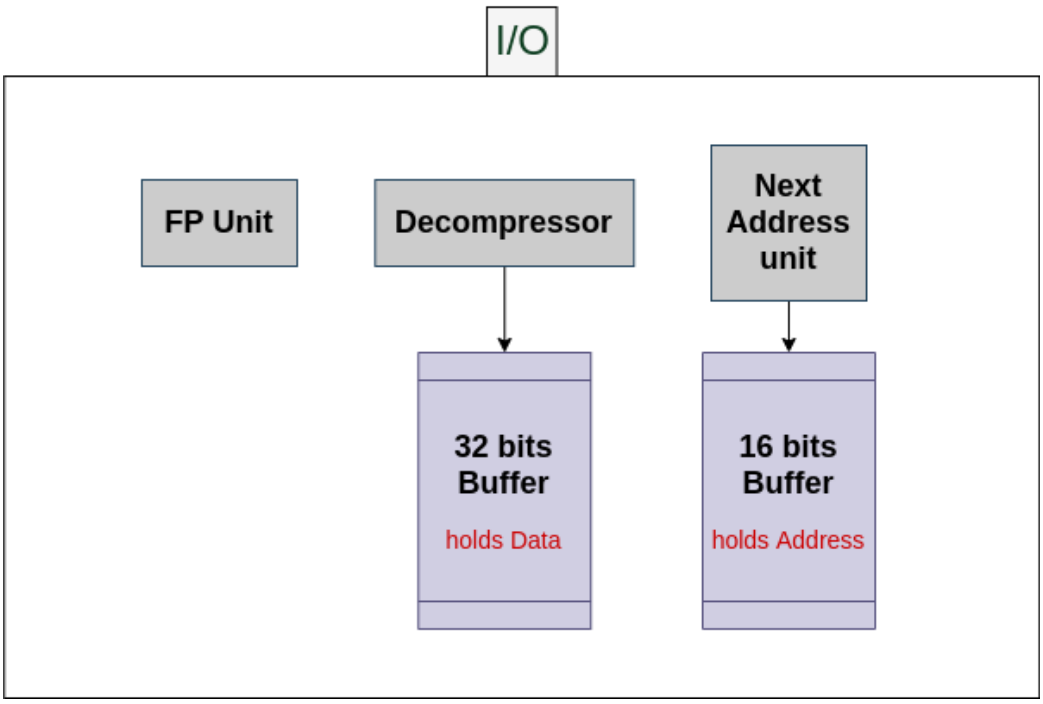

Figure 2: I/O Internal Design

- OUT: Error/Success to CPU.

Sub-modules

- Decompressor: For decompression, see section 0.16.
- Next Address Unit: Calculates the next address to load at the address bus.

  NAU has the header (the first 32 input data) that contains N, M and T_{size}. When reading data (e.g. matrix A) it calculates where it will end and proceeds with the next address.

0.14.2 Solver

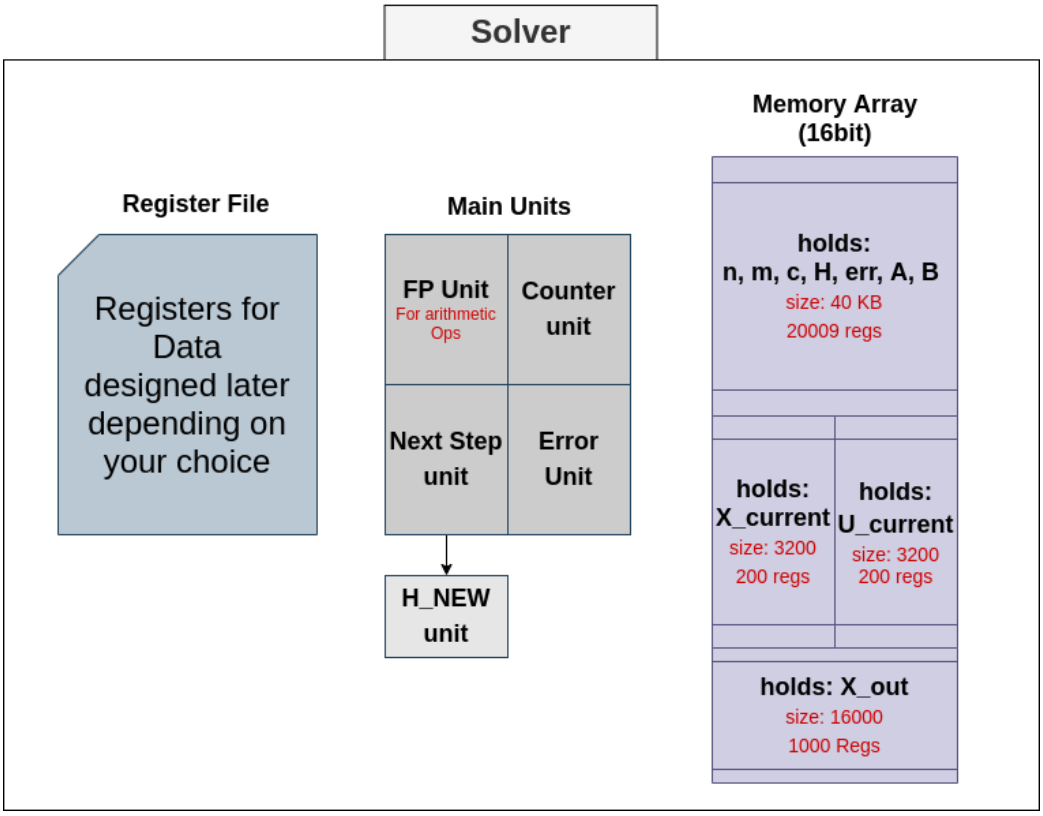

Figure 3: Solver Internal Design

Role

Receives U_h from interpolator, gives it another h to compute $U_{h_{new}}$ at, then computes X_h and decides to stop and flush output to I/O or continue.

At the beginning it receives its data from I/O such as N, M, err, h ... etc.

- Computes the upcoming X knowing h, the previous X and U.
- Counts the error difference and the new h.
- Checks for arithmetic errors that may occurs (e.g. div. by zero).
- Outputs the final X's at the desired times to the RAM.

Ports

- IN: Done signal from interpolator.
- INOUT: 32bit data bus with other modules.
- INOUT: 16bit address bus.
- IN: CLOCK.
- ASYNC IN: Reset.
- IN: 2bit Load/Wait/Process/Out.
- OUT: Interrupt to CPU.
- OUT: Error/Success to CPU.

Solver Memory

- Main Part: 40 KB $\rightarrow$ 20009 (16 bits) Registers
  - N, M, C = 16 bits
  - h = 64 bits
  - err = 64 bits
  - A = [50*50]*64 = 160000 bits
  - B = [50*50]*64 = 160000 bits

- $X_current$: 3200 bits → 200 (16 bits) Registers
  - X = 50*64 = 3200 bits
  - 50: max of M
  - 64: max of numbers
- $U_current$: 3200 bits → 200 (16 bits) Registers
  - U = 50*64 = 3200 bits
  - 50: max of M
  - 64: max of numbers
- X_out: 16000 bits → 1000 (16 bits) Registers
  - X = 5 * 50 * 64 = 16000 bits
  - 50: max of M
  - 64: max of numbers
  - 5: max of times answer is required

Note: main module stores all the 5 output values and flushes them to the CPU all at once at the end.

Sub-modules

- FPU: see Section 0.14.4.
- Error Unit: to detect any error in sizes, h, numbers ... etc.
- Next Step Unit: helps create the upcoming h_{new} so that when solver is busy calculating X_h, interpolator is calculating $U_{h_{new}}$ [1], this unit represents the stepper unit, holds the logic of calculating the adaptive h, and detects when to stop, in summation it calculates the next h, even if it was fixed step.
- Counter Unit: tells you when to calculate more, when to advance to next time (in T_s), and when to stop the whole operations.

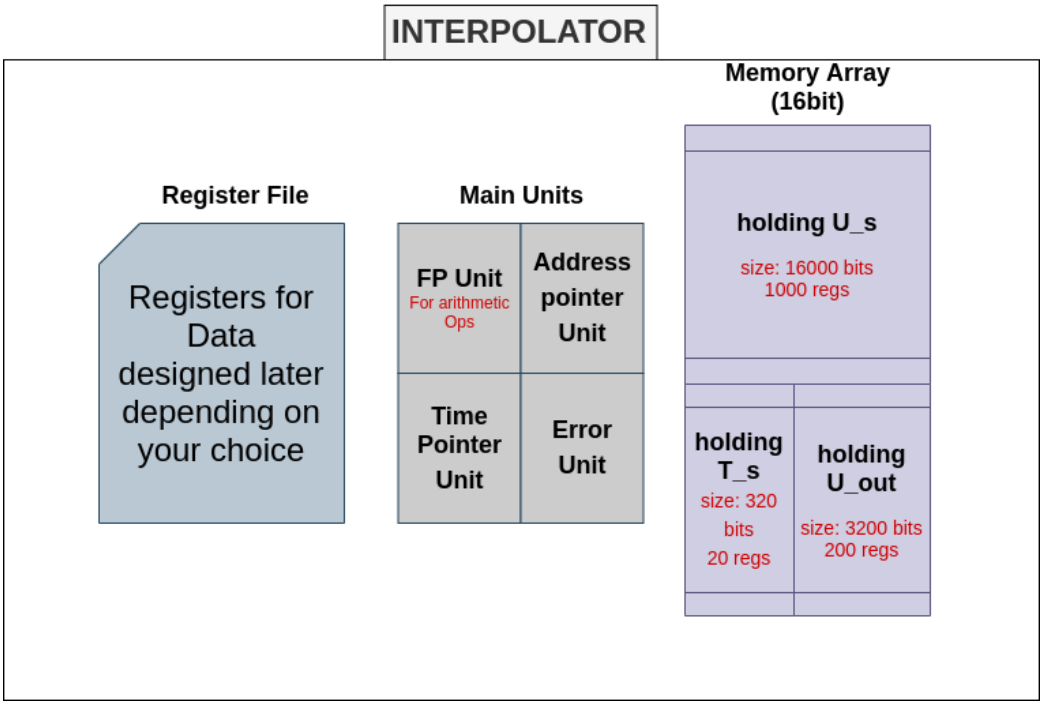

Figure 4: Interpolator Internal Design

0.14.3 Interpolator

Interpolator Job Description

- Only computes U at a specific time.
- At the beginning it receives from IO the U_s and T_s fixed variables/data, and stores them.
- Each U_s is at most of size [64*50] = 3200 bits = 200 registers
- Each T_s is at most of size [64] bits = 4 regs.
- T_s keeps hold of the time where each U_s represents, for example, T_s = [1, 2, 3], there fore the first 200 regs. In U_s are the value of U at time 1, and so on...
- You will need an iterator to target the appropriate U_s from the array of memory, that's the **Address pointer Unit**.
- Time Pointer Unit: Identifies which T of the T_s array the unit is currently handling.
- Note: at the beginning of the program $T_{init} = 0$, $T_{final} = T_s[0]$, after a successful output, $T_{init} = T_{final}$, and $T_{final} = T_s[1]$, and so on...

Role

- Calculates the upcoming U knowing h, U initial and U final.

Ports

- OUT: Done signal to Solver.
- INOUT: 32bit data bus with other modules.
- IN: 16bit address bus.
- IN: CLOCK.
- ASYNC IN: Reset.

[1] See Section 0.3 to learn more about pipelining in design.

- IN: 2bit Load/Wait/Process/Out.
- OUT: Interrupt to CPU.
- OUT: Error/Success to CPU.

Interpolator Memory

- U_s: 16000 bits → 1000 (16 bits) Registers
  - U_s = 5*[50]*64 = 16000 bits
  - 50: max of M
  - 64: max of numbers
  - 5: max of times answer is required
- T_s: 320 bits → 20 (16 bits) Registers
  - T_s = 5*64 = 320 bits
  - 5: max of times answer is required
  - 64: max of numbers
- U_{out}: 3200 bits → 200 (16 bits) Registers
  - U = 50*64 = 3200 bits
  - 50: max of M
  - 64: max of numbers

Sub-modules

- Error Unit: responsible for detecting when an arithmetic error may occur, like dividing by zero, **h** is getting bigger every time, different sizes....etc.
- FPU: see Section 0.14.4.

0.14.4 Fixed/Floating Point Unit (FPU)

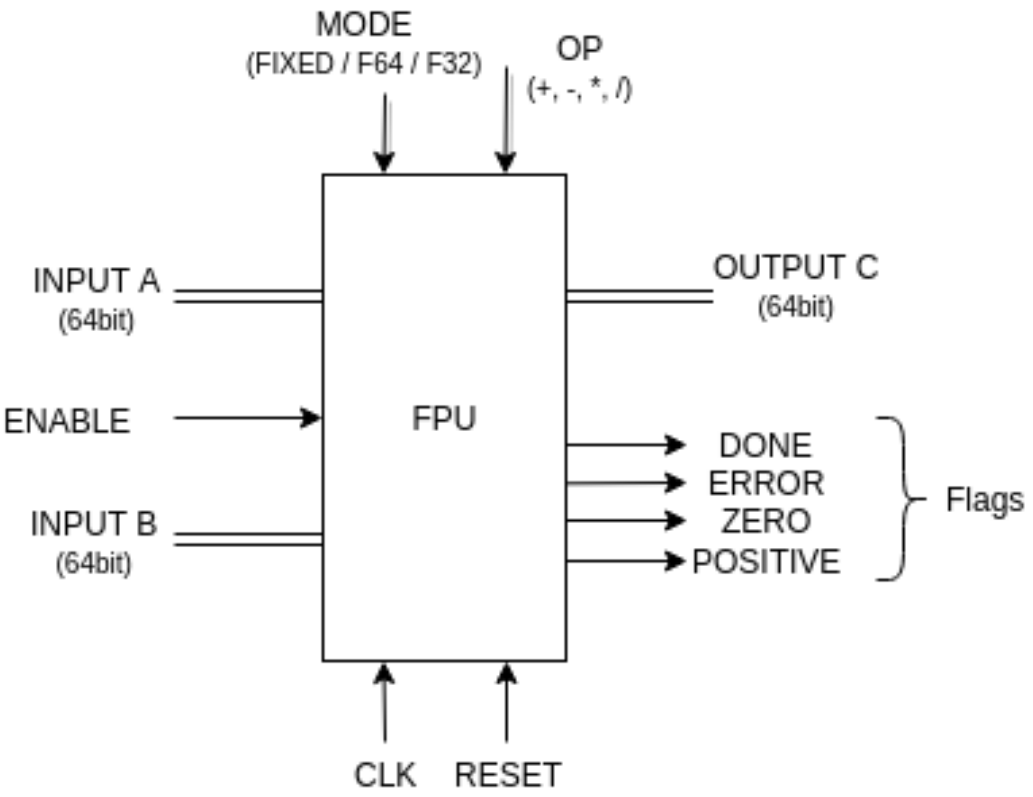

Figure 5: FP Unit

Role

- Instantiated, in the rest modules, multiple times and for different purposes.
- perform the following operations given in port *OP*.
- perform them in different modes given in *MODE*.

Ports

- MODE: IN
  - FIXED(0): operate on LOWEST 16bits of both A and B following Fixed point specifications.
  - F64(1): operate on ALL 64bits of both A and B following Floating point specifications.
  - F32(2): operate on LOWEST 32bits of both A and B following Floating point specifications.
- OP: IN
  - 0: $C = A + B$
  - 1: $C = A - B$
  - 2: $C = A * B$
  - 3: $C = \frac{A}{B}$
- CLK: IN
  FPU operates on POSITIVE edge.
- DONE: OUT
  one operation needs multiple clocks, so FPU must RAISE DONE to 1 when finished for EXACTLY ONE clock cycle.
- ENABLE: IN
  FPU must only operate when ENABLE is set to 1 AND there is NO ERROR.
- A, B: IN
  Input busses, all are 64bit wide.

- C: OUT
  Output bus, 64bit widw.
- ERROR: OUT
  - FPU sets ERROR to 1 when an exception takes place.
  - Each mode has its possible set of exceptions, see the corresponding specifications.
  - MUST stay at 1 after setting it, until RESET is set to 1.
- RESET: ASYNC IN
  clears ERROR state REGARDLESS of ENABLE input.
- ZERO: OUT
  Set when output C is zero.
- POSITIVE: OUT
  Set when output C is positive, otherwise cleared.

Fixed point specifications

- 16bit input
- 16bit output
- CONSTANT scale factor for both input and output = 7. see Subsubsection 0.14.4.
- in case of overflow or division by zero, ERROR MUST be set to 1 and MUST stay at 1 until RESET is set to 1.

Floating point specifications

- MUST adhere to IEEE-754 2019-revision [2] for both fp32 and fp64 modes.
- ERROR is set to 1 when ANY of the exceptions stated in the IEEE-754 takes place, and stay at 1 until RESET is set to 1.

Scale Factor Definition

Scale Factor is an integer used to obtain the real number from the fixed point number and vice versa using the following formula: $y = \frac{x}{2^s}$; where:

- $x \in \mathbb{N}$ is the fixed point number stored as an integer.
- $y \in \mathbb{R}$ is the real number that represents x with some error.
- $s \in \mathbb{N}$ is the scale factor.

0.15 Compression

Follow bit-level Run-length encoding to compress ram content before sending them, by taking each (one to eight) [1:8] repeating bits and compressing them into four bits, using RLE (Run length encoding) algorithm.

0.15.1 Input

Bit stream input data to IO.

0.15.2 Output

Y compressed 4bit packets, where $X \geq Y \geq \lceil \frac{X}{8} \rceil$ Each packet must follow the format in Table 5.

Table 5: Compression Output

Bit Index	Description	Size
3:1	Number of bits to generate - 1	3 bits
0	Bit to generate	1 bit

Note: the 3 bits must be incremented before decode.

0.15.3 Example

Table 6: Compression Example

Original	Compressed
11111111	1111
0000	0110

0.15.4 Algorithm

```
c = first bit in bit_stream
count = 0
for b in bit_stream:
    if c == b and count < 7:
        count++
    else:
        emit_packet(count-1, c)
        count = 1
        c = b
```

Compression Design Justification

Because the occurence of more that 7 ones or zeros simultaneously is very rare.

Problems

This compression algorithm may not compress the data, rather than that it may increase the number of bits.

0.16 Decompression

Follow this simple algorithm to fill the buffer, once its full (has 32bits) flush it to data bus and update the address.

```
for packet in packets:
    # bit to repeat
    b = packet[0]

    # extract 3 bits
    n = packet[3:1]
```

```
    # increment by 1 to get number of repetitions
    n++

    # generate n bit of b
    for _ in range(n):
        buffer.push(b)

    if buffer.full():
        buffer.flush()
```

Part IV

Bibliography

Bibliography

[1] *ODE Solver – System Description*. CMP Engineering Department, Cairo University. Spring 2020.
https://docs.google.com/document/d/1eHHQ9NC2HlrqLECSz6iQl5c6tR_y5AxtJBdxklAY5g4

[2] Wikipedia contributors. (2020, March 15). IEEE 754. In Wikipedia, The Free Encyclopedia. Retrieved 19:00, March 21, 2020, from *https://en.wikipedia.org/w/index.php?title=IEEE_754&oldid=945680688*