

Shitty Fucking Useless Draft/Design

Mahmoud Adas, Evram Youssef, Mohamed Shawky, Remonda Talaat

March 20, 2020

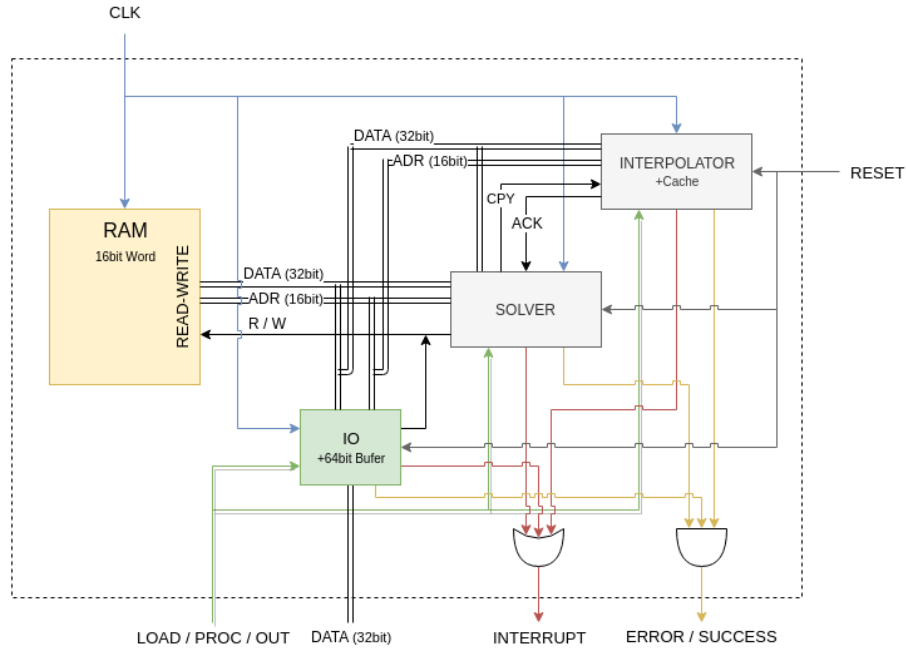


Figure 1: Overall Design

Interfaces and HW Summary

The hardware has the following interfaces that triggers some actions summarized below and detailed in the rest of the document.

- CLK: IN
- RESET: IN
 - clears all internal states of all modules:

- * IO internal buffer
- * ERROR/SUCCESS of all modules resets to SUCCESS(1)
- * INTERRUPT resets to zero
- * INTERPOLATOR invalidates all its cache, which means it needs to refill it from IO
- * SOVLER invalidates all its cache and registers, which means it needs to access the ram again
- * CPY from solver to interp, and ACK from interp to solver are both zeroed to stop any copy operations
- RAM is NOT cleared
- ASYNC
- CPU is expected next clock to turn the LOAD / PROC / OUT into LOAD state and we will start loading input again.
- LOAD / PROC / OUT (2bit): IN
 - set the current major state of the machine
 - LOAD(0):
 - * only IO, RAM, INTERPOLATOR work
 - * IO receives *compressed* data from the CPU
 - * IO decompresses data into buffer
 - * buffer is written into RAM and/or INTERPOLATOR CACHE depending on internal counter
 - * ends when IO flushes all buffer and raises INTERRUPT with either SUCCESS or ERROR
 - PROC(1):
 - * only RAM, SOLVER, INTERPOLATOR work
 - * SOLVER and INTERPOLATOR work concurrently to calculate their outputs
 - * INTERPOLATOR waits for SOLVER CPY to copy its output then proceeds to calculating next output
 - * ends when either SOLVER or INTERP raises INTERRUPT with either SUCCESS or ERROR
 - OUT(2):
 - * only IO, RAM work
 - * IO just copies final outputs to cpu from RAM
 - * 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(1), otherwise it's ERROR(0)
- ERROR / SUCCESS: OUT
 - CPU should operate on this value only when INTERRUPT is 1
 - errors that could happen include: divide by zero, $H > 1$, incomplete

input

Simulation Workflow

Input Preparing

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

- INPUT: json file that follows the format stated in main document
- create bit stream of the read data that follows the **Input Data Structure** specifications
- 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
- 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

Instantiating HW

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

Loading Input

- 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

Processing

- put HW at PROCESS state
- wait for the INTERRUPT positive edge
- check for ERROR / SUCCESS and only proceed if it is SUCCESS

Extracting Output

- 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

Sepecifications

Memory Mapping

Not all modules listen on all addresses.

If address bus is loaded with an address **A** that some module **M** is not assigned to, module **M** must ignore the data and address bus.

Address	Type	Size (words)	Name	Description
0x0000	struct Header	1	Header	Includes Dimensions and modes
0x0001	f64	4	H	Timestep (variable step mode)
0x0005	f64	4	Error	Error Tolerance (variable step mode)
0x0009	f64[64]	256	T	Time points where solutions are required
0x0109	f64[50][50]	10000	A	Matrix A
0x2819	f64[50][50]	10000	B	Matrix B
0x4F29	f64[50]	200	X	Initial value of X

Address	Type	Size (words)	Name	Description
0x4FF1	f64[50] [64]	12800	Xout	Final Output X
0x81F1	f64[50]	200	U0	Initial U vector
0x82B9	f64[50] [64]	12800	Us	U vector at required time steps
0xB4B9	f64[50]	200	Uint	Interpolated U Vector

Modules

RAM

TODO: figure showing its ports

- Role:
 - Store input data for solver to access
 - Store output data from solver that IO will later will transfer back to CPU
- Ports:
 - INOUT: 32bit data bus
 - IN: 16 bit address bus
 - IN: R/W control signal
- Word: 16 bit
- Size: 33265 words
- Address Range: [0x0000, 0x81F0] all readable and writeable

IO

TODO: role TODO: figure showing its ports TODO: ports

Solver

TODO: role TODO: figure showing its ports TODO: ports

Interpolator

TODO: role TODO: figure showing its ports TODO: ports

Fixed/Floating Point Unit (FPU)

TODO: role TODO: figure showing its ports TODO: ports

Header Data Structure

Index	Name	Description	Datatype	Total Size
15:10	N	Dimension of X	<code>uint</code>	6 bits
9:4	M	Dimension of U	<code>uint</code>	6 bits
3	Solver Mode	Fixed Step or Variable Step	<code>enum</code>	1 bit
2:1	FPU Precision	fixed point, f64 or f32	<code>enum</code>	2 bits
0	NOT USED	——	——	1 bit

Compression

TODO

Decompression

TODO