ODE Solver Design Report

Mahmoud Adas
`Section:2, BN:21`

Evram Youssef
`Section:1, BN:9`

Remonda Talaat
`Section:1, BN:20`

Mohamed Shawky
`Section:2, BN:16`

March 20, 2020

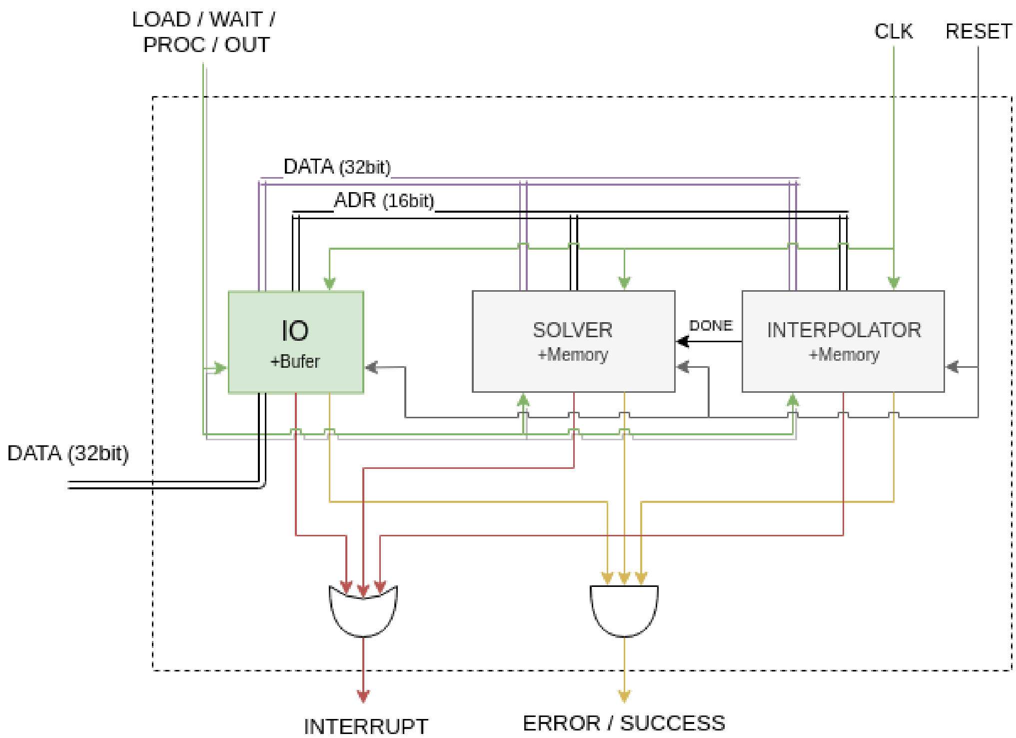

Figure 1: Overall Design

1 Variables Specs

- N : 6 bits [1:50]
- M : 6 bits [1:50]
- C : 3 bits [1:5]
- h : 64 bits
- err : 64 bits
- mode : 1 bit [0,1]
- fp: 2 bits [0,1,2]
- A : 160000 bits, [N*N] each number could be 16bit fixed point/ floating point 32/ or floating point 64
- B : 160000 bits, [N*M] each number could be 16bit fixed point/ floating point 32/ or floating point 64
- X : 3200 bits, [N*1] each number....
- U : 3200 bits, [M*1] each number....
- T_s : 320 bits, [5*1] each number....
- X_out : 16000 bits [N*1]*5

2 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: 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.

3 Simulation Workflow

3.1 Input Preparing

This stage is the responsibility of a script that runs 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.

3.2 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.

3.3 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.

3.4 Processing

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

3.5 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.