Interfaces for PLIC and CLINT

Read/Write exchanges through ReadData, WriteData ports – During interrupt service routines.

Text

Description automatically generated

Core not connected to PLIC.

Only Machine (M) privilege mode.

Integration of GPIO

General Master Process (Test bench):

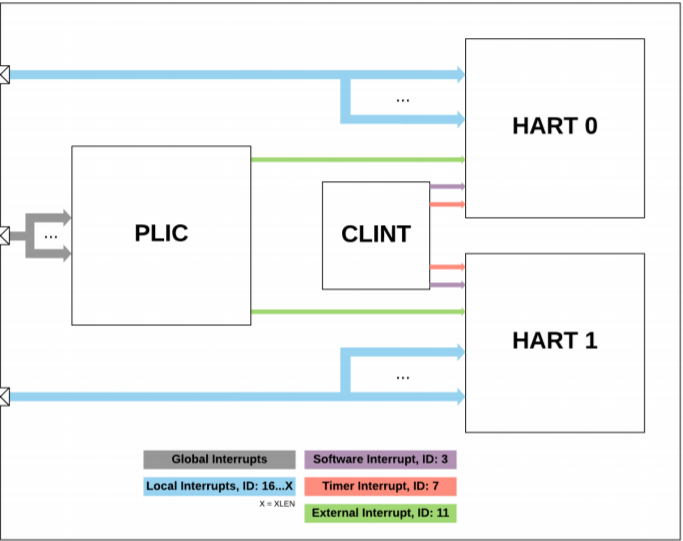
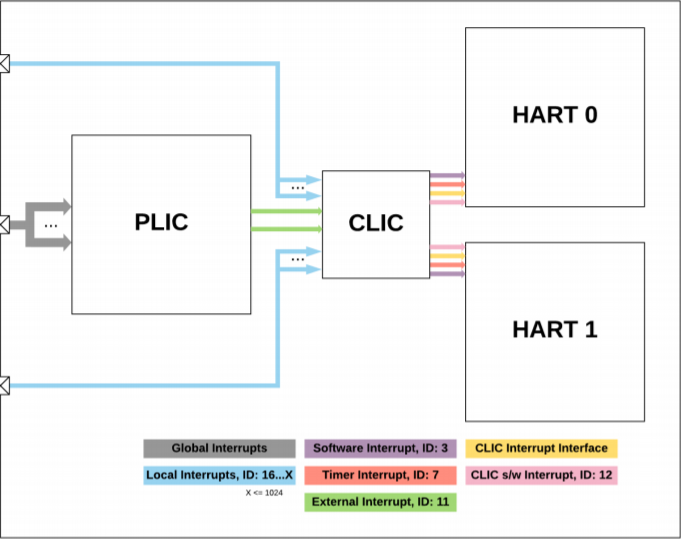
* Set the priority of each interrupt (Write to CSR)
* Enable the interrupts and set the priority threshold for the core (Write to CSR)
* Send the interrupts
* Wait until the interrupt is received at the PLIC, then read the interrupt ID (claim)
* Write the interrupt ID that was received (completion and clear?)

Diagram

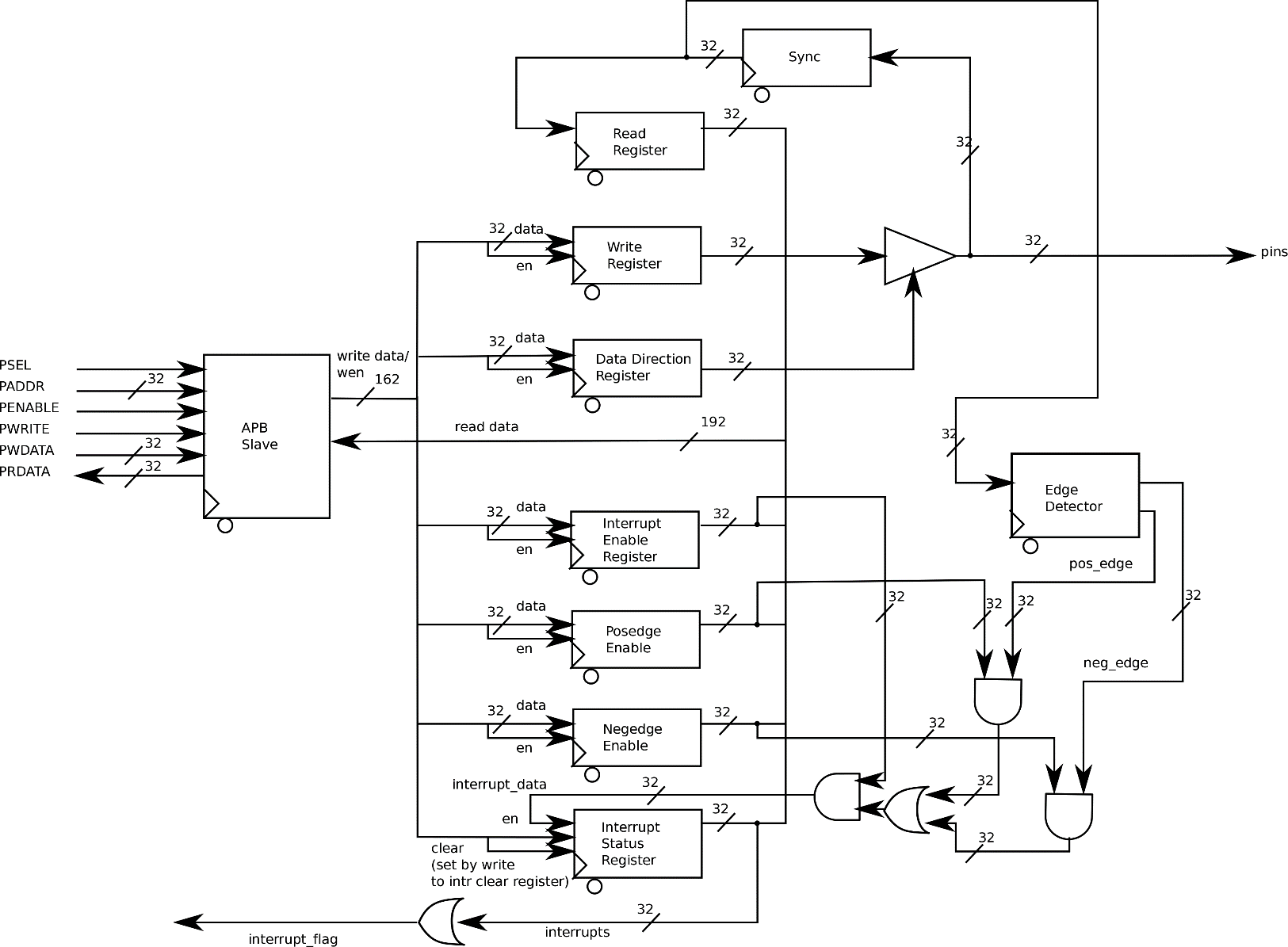
Description automatically generated

Transactions/Test Cases (Draft-1)

* Propagation of Single Interrupt
  + Config regs – config phase (Randomize)
  + HW Interrupt reg req
* Cole
  + PLIC parameterized for diff num of interrupts
    - Mult of 32
    - Non-multiples of 32
* Interrupts coming from multiple sources
  + Different priority levels
  + Same priority levels – Resolution through Interrupt ID
  + Use - Masking, Interrupt Enables, Interrupt\_below\_val (config)
* Interrupt types (pos, neg) - X
  + edge triggered (counters)
  + level triggered - X
* Preemption - New interrupt while an interrupt is being serviced (with random delays)
  + NOT SUPPORTED
  + Preemption at HW level – Multiple pulses of interrupt req
  + Multiple interrupts from the same source
  + Higher priority interrupt
  + Higher privilege mode – X (only M mode)
  + Clint - Local and Global/external interrupt (timer, SW execp, external interrupt at the same time)
* Interrupt request cleared/withdrawn from interrupt source (not likely in edge triggered?),
  + NOT SUPPORTED
  + Before interrupt is claimed
  + After interrupt is claimed (during ISR)
  + Expected behaviour on the IP interrupt interface
* CLINT ( and PLIC?) – Direct and Vectored modes
  + HANDLED BY THE CORE
  + CLINT/CLIC name in uvm\_clic branch
  + Top level module in TB captures CLINT too?
  + CSR – Inside Core
  + Rest (relevant to PLIC) in PLIC
* Timers – time\_cmp XFeature
* Interrupt Acknowledge
* TBD and discussed
  + Address generation in vectored mode (PLIC, CLINT)
  + CLIC,
    - Documentation (Only RV GitHub doc as of now)
    - Lot more complex. Offers lots of features and programmability
  + Verification of individual modules (link)
  + I/O rdata, wdata transactions during service routines (model without core?)
  + Interacting with the DUT, GPIO, Timer
  + Cycle level details for simulation without core?
  + Interleaving of Interrupts - UART,USB,GPIO, etc. Need to save additional states? Handler
  + What happens when ISR causes an exception? (Unlikely)



GPIO



Transactions/Test Cases (Draft-2)

* Propagation of Single Interrupt
  + Config regs – config phase (Randomize)
  + HW Interrupt reg req
* Cole
  + PLIC parameterized for diff num of interrupt sources
    - Mult of 32
    - Non-multiples of 32
* Interrupts coming from multiple sources
  + Different priority levels
  + Same priority levels – Resolution through Interrupt/Source ID
    - Not sure if this is supported
  + Use these features,
    - Masking
    - Interrupt Enables
    - Interrupt\_below\_val (config)
* Timers – time\_cmp XFeature
* AHB side verification?

Steps to follow for testing the DUT,

State machine inside scoreboard?

* When interrupt comes in, the ID of the interrupt will be reflected in interrupt Claim/Complete register
  + Check-1 : Predict resolution, reset checks
  + Reg Model : Might have to access,
    - * Separate sequence for configuration
      * Phasing – run : reset – config - main
    - Mask reg
    - mip reg
    - mie reg
    - threshold reg
    - below\_val reg
    - priority reg
    - control register - .get. Store in tb variables
    - status - .read and not .get - cluelogic
* When this register is read from (via AHB), an interrupt claim is performed. PLIC will internally clear the corresponding interrupt pending bit from mip reg
  + Check-2 : Reg Model clears the corresponding bit from mip reg
* Interrupt service is completed when a write (dummy write via AHB) is performed to the interrupt Claim/Complete register.
  + Predict the next interrupt to be serviced?
* PLIC moves on to the next interrupt source only after interrupt completion. So PLIC will send a new interrupt request pulse if,
  + There’s a new interrupt or,
  + If an interrupt has been serviced/completed and there’s another pending interrupt.
    - Check-3 : Check resolution correctness for the next interrupt to be serviced

General Master Process (Test bench):

Can use the wrappers for PLIC and CLINT (integrated with AHB)

* Set the priority of each interrupt (Write to CSR)
* Enable the interrupts and set the priority threshold for the core (Write to CSR)
* Send the interrupts
* Wait until the interrupt is received at the PLIC, then read the interrupt ID (claim)
* Write the interrupt ID that was received (completion and clear?)

Agents

* Active – AHB, Interrupt Sources/Vector, PLIC & CLINT (DUT?)
* Passive – Core(only receives interrupt request)? Can core be parametrized to initiate read/write transactions through AHB?

Registers for Reg Model

Checker – Priority Resolution algorithm + Register Model

Edge Cases

* Interrupt request sent to the core. Core does not claim the request.
* Interrupt request claimed but never serviced by the core.
* Interrupt never requested, but service completion signaled (via dummy write to Claim reg).
* Interrupt requested and service completion signaled before interrupt is claimed.
* Claim complete write doesn’t look like a dummy write. Write data should be interrupt ID
* Change config (enable, threshold) before claim, or after claim but before claim completion

GPIO – How to drive inputs?

PLIC

Diagram

Description automatically generated

Chart

Description automatically generated with medium confidence