Design and implementation

1.Design thought

- The Svnapot extension is essentially a base page that supports multiple sizes, this time a base page of 64KB is implemented on the NutShell as described in the manual.
- To implement this extension, we first need to modify the PTE encoding format to add N bits to the high bits.In the PTW logic, Walk to the last level and add additional logic to determine whether the PTE node is a Svnapot compliant node.And when you put it in the TLB, you need to specify whether this is a special 64KB PTE node.
- NutShell's ITLB and DTLB are fully connected structures. In order to minimize code modification, it was decided not to add new flag bits to the TLB entries, but to use a new mask to do this.The original TLB mask has three conditions, one is 0x00000 for 1GB, one is 0x3fe00 for 2MB, and the other is 0x3ffff for 4KB. For the Svnapot extension of 64KB base page implemented this time, we only need to add a mask of 0x3fff0.
- When the PTW Walk to the last level checks that the PTE node is a special 64KB PTE node, set it's mask to 0x3fff0 and put it into the TLB to achieve the effect of the special identification, and all the other logic of the TLB hit call does not require additional modification.

2.Specific code implementation

To achieve configurability, first add an entry in the configuration item whether the Svnapot extension is enabled or not

```
object DefaultSettings {
  def apply() = Map(
    "MemMapBase" -> 0x0000000000000000L,
    "MemMapRegionBits" -> 0,
    "MMIOBase" -> 0x0000000040000000L,
    "MMIOSize" -> 0x0000000040000000L,
    "ResetVector" -> 0x80000000L,
    "NrExtIntr" -> 1,
        ........
+   "Napot_on" -> true
  )
}
```

Set some Svnapot constants

```
  //for svnapot extension
  val napot_on = if (Settings.get("Napot_on")) true else false
  val napot_bits = 4
  val napot_patten = "b1000"
  val napot_mask : String = "h3fff0"
```

The encoding of PTE needs to add N bit, so it also needs to be modified

```
def pteBundle = new Bundle {
```

```
+   val n = if(napot_on) Bool() else null

+   val reserved  = if (napot_on) UInt((pteResLen - 1).W) else UInt(pteResLen.W)
    val ppn  = UInt(ppnLen.W)
    val rsw  = UInt(2.W)
    val flag = new Bundle {
      val d    = UInt(1.W)
      val a    = UInt(1.W)
      val g    = UInt(1.W)
      val u    = UInt(1.W)
      val x    = UInt(1.W)
      val w    = UInt(1.W)
      val r    = UInt(1.W)
      val v    = UInt(1.W)
    }
  }
```

napotCheck is a node that describes whether or not the Svnapot extension is violated

```
val napotCheck = if(napot_on)
            ((level =/= 1.U) || ((memRdata.n && memRdata.ppn(napot_bits-1,0) ===
napot_patten.U) || !memRdata.n))
                  else
             true.B
```

After the PTW finds the leaf, the permissions are checked, and the Svnapot extension is checked for violations. The napotCheck is mixed into Permexec, Permload, and PermStore

```
if(tlbname == "itlb") {
+ when (!(permExec && napotCheck)) { missIPF := true.B ; state := s_wait_resp}
  .otherwise {
    state := Mux(updateAD, s_write_pte, s_wait_resp)
    missMetaRefill := true.B
  }
}
if(tlbname == "dtlb") {
+  when( (!permLoad && req.isRead() || !napotCheck) || (!permStore  &&
req.isWrite()) || !napotCheck) {
    state := s_miss_slpf
    loadPF := req.isRead() && !isAMO
    storePF := req.isWrite() || isAMO
  }.otherwise {
    state := Mux(updateAD, s_write_pte, s_wait_resp)
    missMetaRefill := true.B
  }
}
```

Make a distinction between the mask that writes the TLB： PTW finds the PTE at the last level and the mask is set to 0x3fff0 if it is a PTE of Svnapot extension

```
if(napot_on)
    missMask := Mux(level===3.U, 0.U(maskLen.W), Mux(level===2.U,
"h3fe00".U(maskLen.W), Mux((memRdata.n && memRdata.ppn(napot_bits-1,0) ===
napot_patten.U),napot_mask.U(maskLen.W),"h3ffff".U(maskLen.W))))
else
    missMask := Mux(level===3.U, 0.U(maskLen.W), Mux(level===2.U,
"h3fe00".U(maskLen.W), "h3ffff".U(maskLen.W)))
```

In the case of write back to A/D bit, it is necessary to distinguish whether the PTE in TLB is the PTE of Svnapot extension, and use the mask to distinguish, if so, it is necessary to maintain the N bit when write back.

```
  //!!! PPN needs to be 44 bit-width
  val hitPTEStore = RegEnable(Cat(0.U(10.W) , 0.U( (44 - hitData.ppn.getWidth).W
) , hitData.ppn, 0.U(2.W), hitRefillFlag), hitWB)
  val hitNapotPTEStore = if (!napot_on) null else
RegEnable(Cat(1.U(1.W),0.U(9.W), 0.U( (44 - hitData.ppn.getWidth).W ) ,
hitData.ppn, 0.U(2.W), hitRefillFlag), hitWB)
  val hitWBStore =  if(napot_on)
                    Mux(hitMeta.mask ===
napot_mask.U,hitNapotPTEStore,hitPTEStore)
                  else
                    hitPTEStore
```

Fortunately, the rest of the TLB's judgment logic is compatible with Svnapot after these changes, because it's just a new condition for the mask in the TLB, so no other code changes are required.

3.Appendix

- Details of NutShell code changes
- Details of NEMU code changes