```
section ‹Eiger Port+ Refinement Proof Invariants (and important lemmas)›

theory CCv_Eiger_Port_modified_Invariants
  imports CCv_Eiger_Port_modified
begin

— ‹Invariants about kv store›
definition KVSNonEmp where
  "KVSNonEmp s ⟷ (∀k. DS (svrs s k) ≠ [])"

definition KVSNotAllPending where
  "KVSNotAllPending s k ⟷ (∃i. i < length (DS (svrs s k)) ∧ ¬v_is_pending (DS (svrs s k) ! i))"

definition KVSSNonEmp where
  "KVSSNonEmp s ⟷ (∀k. kvs_of_s s k ≠ [])"

— ‹Invariant about future and past transactions svrs›

definition FutureTIDInv where
  "FutureTIDInv s cl ⟷ (∀n k. n > txn_sn (cls s cl) ⟶ wtxn_state (svrs s k) (Tn_cl n cl) = Ready)"

definition PastTIDInv where
  "PastTIDInv s cl ⟷ (∀n k. n < txn_sn (cls s cl) ⟶ wtxn_state (svrs s k) (Tn_cl n cl) ∈ {Ready, Commit})"

lemma other_sn_idle:
  assumes "FutureTIDInv s cl" and "PastTIDInv s cl"
    and "get_cl_txn t = cl" and "get_sn_txn t ≠ txn_sn (cls s cl)"
  shows "⋀k. wtxn_state (svrs s k) t ∈ {Ready, Commit}"


abbreviation not_committing_ev where
  "not_committing_ev e ≡ ∀cl kv_map cts sn u. e ≠ RDone cl kv_map sn u ∧ e ≠ WCommit cl kv_map cts sn u"

abbreviation invariant_list_kvs where
  "invariant_list_kvs s ≡ ∀cl k. FutureTIDInv s cl ∧ PastTIDInv s cl ∧ KVSNonEmp s ∧ KVSNotAllPending s k"

subsection ‹Refinement Proof›

lemma pending_rtxn_inv:
  assumes "∀keys kv_map. txn_state (cls s cl) ≠ RtxnInProg keys kv_map"
    and "∀keys kv_map. txn_state (cls s' cl) ≠ RtxnInProg keys kv_map"
    and "∀cl'. cl' ≠ cl ⟶ cls s' cl' = cls s cl'"
  shows "pending_rtxn s' t = pending_rtxn s t"


lemma pending_wtxn_inv:
  assumes "∀kv_map. txn_state (cls s cl) ≠ WtxnPrep kv_map"
    and "∀kv_map. txn_state (cls s' cl) ≠ WtxnPrep kv_map"
    and "∀cl'. cl' ≠ cl ⟶ cls s' cl' = cls s cl'"
  shows "pending_wtxn s' t = pending_wtxn s t"


lemma kvs_of_s_inv:
  assumes "state_trans s e s'"
    and "invariant_list_kvs s"
    and "not_committing_ev e"
  shows "kvs_of_s s' = kvs_of_s s"


lemma finite_pending_wtxns:
  assumes "pending_wtxns (svrs s' k) t = Some x"
    and "∀k'. finite (ran (pending_wtxns (svrs s k')))"
    and "∀k'. k' ≠ k ⟶ pending_wtxns (svrs s' k') = pending_wtxns (svrs s k')"
    and "∀t'. t' ≠ t ⟶ pending_wtxns (svrs s' k) t' = pending_wtxns (svrs s k) t'"
  shows "∀k. finite (ran (pending_wtxns (svrs s' k)))"


definition FinitePendingInv where
  "FinitePendingInv s svr ⟷ finite (ran (pending_wtxns (svrs s svr)))"

lemma clock_monotonic:
  assumes "state_trans s e s'"
  shows "clock (svrs s' svr) ≥ clock (svrs s svr)"


definition PendingWtsInv where
  "PendingWtsInv s ⟷ (∀svr. ∀ts ∈ ran (pending_wtxns (svrs s svr)). ts ≤ clock (svrs s svr))"

definition ClockLstInv where
  "ClockLstInv s ⟷ (∀svr. lst (svrs s svr) ≤ clock (svrs s svr))"
```

```
lemma lst_monotonic:
  assumes "state_trans s e s'"
  shows "lst (svrs s' svr) ≥ lst (svrs s svr)"


lemma gst_monotonic:
  assumes "state_trans s e s'"
  shows "gst (cls s' cl) ≥ gst (cls s cl)"


lemma tm_view_inv:
  assumes "state_trans s e s'"
    and "not_committing_ev e"
  shows "cl_view (cls s' cl) = cl_view (cls s cl)"


end
```