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

theory CCv_Eiger_Port_modified_Invariants
  imports CCv_Eiger_Port_modified
begin

— ‹Lemmas about simulation functions›

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_rtxn_added:
  assumes "txn_state (cls s cl) = Idle"
    and "txn_state (cls s' cl) = RtxnInProg keys kv_map"
    and "txn_sn (cls s' cl) = txn_sn (cls s cl)"
    and "∀cl'. cl' ≠ cl ⟶ cls s' cl' = cls s cl'"
  shows "Collect (pending_rtxn s') = insert (get_txn_cl s cl) (Collect (pending_rtxn s))"


lemma pending_rtxn_removed:
  assumes "txn_state (cls s cl) = RtxnInProg keys kv_map"
    and "txn_state (cls s' cl) = Idle"
    and "txn_sn (cls s' cl) = txn_sn (cls s cl)"
    and "∀cl'. cl' ≠ cl ⟶ cls s' cl' = cls s cl'"
  shows "Collect (pending_rtxn s') = Set.remove (get_txn_cl s cl) (Collect (pending_rtxn s))"


lemma pending_wtxn_cl_ev_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 pending_wtxn_svr_ev_inv:
  assumes "cls s' = cls s"
  shows "pending_wtxn s' t = pending_wtxn s t"


lemma pending_wtxn_added:
  assumes "txn_state (cls s cl) = Idle"
    and "txn_state (cls s' cl) = WtxnPrep kv_map"
    and "txn_sn (cls s' cl) = txn_sn (cls s cl)"
    and "∀cl'. cl' ≠ cl ⟶ cls s' cl' = cls s cl'"
  shows "Collect (pending_wtxn s') = insert (Tn (get_txn_cl s cl)) (Collect (pending_wtxn s))"


lemma pending_wtxn_removed:
  assumes "txn_state (cls s cl) = WtxnPrep kv_map"
    and "txn_state (cls s' cl) = WtxnCommit gts cts kv_map"
    and "txn_sn (cls s' cl) = txn_sn (cls s cl)"
    and "∀cl'. cl' ≠ cl ⟶ cls s' cl' = cls s cl'"
  shows "Collect (pending_wtxn s') = Set.remove (Tn (get_txn_cl s cl)) (Collect (pending_wtxn s))"


lemma indices_map_get_ver_committed_rd [simp]:
  "indices_map (map (get_ver_committed_rd s) vl) i = indices_map vl i"


lemma dom_indices_map:
  "dom (indices_map vl i) = v_writer ` set (vl)"


lemma insert_in_vl_ver_features:
  "f ` set (insert_in_vl vl (Some ver)) = insert (f ver) (f ` set vl)"


lemma commit_all_in_vl_length:
  "length (commit_all_in_vl s vl1 vl2) = length vl1 + length vl2"


lemma commit_all_in_vl_writers:
  "v_writer ` set (commit_all_in_vl s vl1 vl2) = v_writer ` set vl1 ∪ v_writer ` set vl2"


lemma commit_all_in_vl_readersets:
  "v_readerset ` (set (commit_all_in_vl s vl1 vl2)) = v_readerset ` set vl1 ∪ v_readerset ` set vl2"


lemma commit_all_in_vl_append:
  "commit_all_in_vl s vl_c (vl @ [ver]) =
  insert_in_vl (commit_all_in_vl s vl_c vl) (Some (committed_ver ver (get_glts s ver) 0))"


lemma get_vl_pre_committed_writers:
  "v_writer ` set (get_vl_pre_committed s vl) = v_writer ` {x ∈ set vl. ¬v_is_pending x ∨ ¬ pending_wtxn s (v_writer x)}"


lemma get_vl_pre_committed_readersets:
  "v_readerset ` (set (get_vl_pre_committed s vl)) ⊆ v_readerset ` (set vl)"


lemma pending_wtxns_empty:
  "pending_wtxns s k = {} ⟷ (∀t. wtxn_state (svrs s k) t ∈ {Ready, Commit})"
```

```
lemma pending_wtxns_non_empty:
  assumes "wtxn_state (svrs s k) t ≠ Ready"
    and "wtxn_state (svrs s k) t ≠ Commit"
  shows "pending_wtxns s k ≠ {}"


— ‹Lemmas for unchanged elements in svrs›

lemma DS_eq_all_k:
  assumes "DS (svrs s' k) = DS (svrs s k)"
    and "other_insts_unchanged k (svrs s) (svrs s')"
  shows "∀k. DS (svrs s' k) = DS (svrs s k)"


lemma eq_for_all_k:
  assumes "f (svrs s' k) = f (svrs s k)"
    and "∀k'. k' ≠ k ⟶ svrs s' k' = svrs s k'"
  shows "∀k. f (svrs s' k) = f (svrs s k)"


lemma eq_for_all_k_t:
  assumes "f (svrs s' k) t = f (svrs s k) t"
    and "∀k'. k' ≠ k ⟶ svrs s' k' = svrs s k'"
    and "∀t'. t' ≠ t ⟶ f (svrs s' k) t' = f (svrs s k) t'"
  shows "∀k. f (svrs s' k) = f (svrs s k)"


lemma eq_for_all_cl:
  assumes "f (cls s' cl) = f (cls s cl)"
    and "∀cl'. cl' ≠ cl ⟶ cls s' cl' = cls s cl'"
  shows "∀cl. f (cls s' cl) = f (cls s cl)"


subsection ‹Monotonic lemmas and inequality of timestamps invariants›

lemma glts_monotonic:
  assumes "state_trans s e s'"
  shows "global_time s' ≥ global_time s"


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


lemma cl_clock_monotonic:
  assumes "state_trans s e s'"
  shows "cl_clock (cls s' cl) ≥ cl_clock (cls s cl)"


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

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

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

definition PendingWtxnsLB where
  "PendingWtxnsLB s svr ⟷ (∀ts ∈ pending_wtxns s svr. lst (svrs s svr) ≤ ts)"

lemma min_pending_wtxns_monotonic:
  assumes "state_trans s e s'"
    and "pending_wtxns s k ≠ {}"
    and "pending_wtxns s' k ≠ {}"
    and "PendingWtxnsUB s k" and "FinitePendingInv s k"
  shows "Min (pending_wtxns s k) ≤ Min (pending_wtxns s' k)"


lemma lst_monotonic:
  assumes "state_trans s e s'"
    and "ClockLstInv s" and "FinitePendingInv s svr"
    and "PendingWtxnsLB s svr" and "PendingWtxnsUB s svr"
  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)"


— ‹Invariants about kvs, global ts and init version v0›

definition KVSNonEmp where
  "KVSNonEmp s ⟷ (∀k. DS (svrs s k) ≠ [])"

definition GltsNotZero where
  "GltsNotZero s ⟷ global_time s > 0"

definition CommitGltsNotZero where
  "CommitGltsNotZero s cl ⟷ (∀gts cts kv_map. txn_state (cls s cl) = WtxnCommit gts cts kv_map ⟶ gts > 0)"

definition InitVerInv where
  "InitVerInv s k ⟷ v_writer (DS (svrs s k) ! 0) = T0 ∧ v_glts (DS (svrs s k) ! 0) = 0 ∧
    ¬v_is_pending (DS (svrs s k) ! 0)"
```

```
definition KVSNotAllPending where
  "KVSNotAllPending s k ⟷  ¬v_is_pending (DS (svrs s k) ! 0)"

lemma get_vl_committed_length_inv:
  assumes "KVSNonEmp s"
    and "KVSNotAllPending s k"
  shows "length (get_vl_committed_wr (DS (svrs s k))) > 0"


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

— ‹To make sure get_glts works›
definition ReadyToCommitVer where (*Not yet proven*)
  "ReadyToCommitVer s k ⟷
    (∀cl v n. v ∈ set (get_vl_ready_to_commit_wr s (DS (svrs s k)))∧ v_writer v = Tn (Tn_cl n cl) ⟶
    (∃glts cts kv_map. txn_state (cls s cl) =  WtxnCommit glts cts kv_map))"


— ‹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 ReadOnlyTxn where
  "ReadOnlyTxn s ⟷ (∀cl svr ks vs. txn_state (cls s cl) ∈ {Idle, RtxnInProg ks vs}
    ⟶ wtxn_state (svrs s svr) (get_txn_cl s cl) = Ready)"

definition WriteTxnIdleSvr where
  "WriteTxnIdleSvr s ⟷
    (∀cl k gts cts kv_map. txn_state (cls s cl) ∈ {WtxnPrep kv_map, WtxnCommit gts cts kv_map}
        ∧ kv_map k = None ⟶ wtxn_state (svrs s k) (get_txn_cl s 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}"


definition FutureTidRdDS where (* Not yet proven *)
  "FutureTidRdDS s cl ⟷ (∀n k. ∀ver ∈ set (DS (svrs s k)). n > txn_sn (cls s cl) ⟶ Tn_cl n cl ∉ v_readerset ver)"

definition FutureTidWrDS where
  "FutureTidWrDS s cl ⟷ (∀n k. n > txn_sn (cls s cl) ⟶ Tn (Tn_cl n cl) ∉ v_writer ` set (DS (svrs s k)))"

— ‹t is not in the v_readerset in the beginning of the transaction›
definition FreshReadTxnInv where (* Not yet proven *)
  "FreshReadTxnInv s cl ⟷ (txn_state (cls s cl) = Idle
    ⟶ (∀k. get_txn_cl s cl ∉ ⋃ (v_readerset ` set (DS (svrs s k)))))"

definition FreshWriteTxnInv where
  "FreshWriteTxnInv s cl ⟷
    (∀keys kv_map k. txn_state (cls s cl) ∈ {Idle, RtxnInProg keys kv_map} ⟶
      Tn (get_txn_cl s cl) ∉ v_writer ` set (DS (svrs s k)))"

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

lemma kvs_of_s_inv: (* Not yet proven *)
  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 writers_inv_not_commit_write:
  assumes "state_trans s e s'"
    and "⋀cl kv_map cts sn u. ¬write_commit cl kv_map cts sn u s s'"
  shows "v_writer ` set (get_vl_pre_committed s' (DS (svrs s' svr))) =
  v_writer ` set (get_vl_pre_committed s (DS (svrs s svr)))"



definition NoPendingInView where (* Not yet proven *)
  "NoPendingInView s ⟷ (∀cl k. cl_view (cls s cl) k ⊆ v_writer ` set (get_vl_pre_committed s (DS (svrs s k))))"

lemma in_view_index_not_none:
  assumes "x ∈ cl_view (cls s cl) k"
    and "NoPendingInView s"
  shows "x ∈ dom (get_indices_map (kvs_of_s s k))"


lemma map_extend_subset:
  assumes "k ∉ dom m1"
    and "m2 = [k ↦ v] ++ m1"
  shows "m1 ⊆m m2"


lemma prefix_update_get_indices_map:
  shows "indices_map (vl1 @ [ver]) i = [v_writer ver ↦ (i + length vl1)] ++ indices_map vl1 i"


lemma prefix_subset_indices_map:
  assumes "v_writer ver ∉ v_writer ` set vl1"
  shows "indices_map vl1 i ⊆m indices_map (vl1 @ [ver]) i"
```

```
lemma read_commit_indices_map_grows: (* Not yet proven *)
  assumes "read_done cl kv_map sn u s s'"
  shows "get_indices_map (kvs_of_s s k) ⊆m get_indices_map (kvs_of_s s' k)"


definition OnlyPendingVer where (* Not yet proven *)
  "OnlyPendingVer s cl k ⟷
  (∀t. ∀ver ∈ set (DS (svrs s k)). v_is_pending ver ∧ is_txn_writer t ver ⟶ t = Tn (get_txn_cl s cl))"

definition CurrentVerPending where (* Not yet proven *)
  "CurrentVerPending s cl k ⟷
    (∀kvm keys ver. txn_state (cls s cl) ∈ {Idle, WtxnPrep kvm, RtxnInProg keys kvm} ∧
    find (is_txn_writer (Tn (get_txn_cl s cl))) (DS (svrs s k)) = Some ver ⟶ v_is_pending ver)"

lemma write_commit_not_add_to_ready:
  assumes "find (is_txn_writer (Tn (get_txn_cl s cl))) (DS (svrs s k)) = None"
    and "txn_sn (cls s' cl) = txn_sn (cls s cl)"
    and "other_insts_unchanged cl (cls s) (cls s')"
    and "svrs s' = svrs s"
  shows "get_vl_ready_to_commit_wr s' (DS (svrs s' k)) = get_vl_ready_to_commit_wr s (DS (svrs s k))"


lemma write_commit_adds_one_to_ready:
  assumes "find (is_txn_writer (Tn (get_txn_cl s cl))) (DS (svrs s k)) = Some ver"
    and "txn_state (cls s cl) = WtxnPrep kv_map"
    and "txn_state (cls s' cl) = WtxnCommit (global_time s) cts kv_map"
    and "txn_sn (cls s' cl) = txn_sn (cls s cl)"
    and "other_insts_unchanged cl (cls s) (cls s')"
    and "svrs s' = svrs s"
  shows "∃ver ∈ set (DS (svrs s' k)). get_vl_ready_to_commit_wr s' (DS (svrs s' k)) =
                                     get_vl_ready_to_commit_wr s (DS (svrs s k)) @ [ver]"


lemma assumes "ver ∈ set (get_vl_ready_to_commit_wr s (DS (svrs s k)))"
    and "find (is_txn_writer (Tn (get_txn_cl s cl))) (DS (svrs s k)) = None"
    and "txn_state (cls s cl) = WtxnPrep kv_map"
    and "txn_state (cls s' cl) = WtxnCommit (global_time s) cts kv_map"
    and "txn_sn (cls s' cl) = txn_sn (cls s cl)"
    and "other_insts_unchanged cl (cls s) (cls s')"
    and "svrs s' = svrs s"
  shows "get_glts s' ver = get_glts s ver"


lemma write_commit_indices_map_grows:
  assumes "write_commit cl kv_map cts sn u s s'"
  shows "get_indices_map (kvs_of_s s k) ⊆m get_indices_map (kvs_of_s s' k)"


subsection‹View invariants›

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


lemma views_of_s_inv:
  assumes "state_trans s e s'"
    and "invariant_list_kvs s"
    and "not_committing_ev e"
  shows "views_of_s s' cl = views_of_s s cl"


lemma read_commit_views_of_s_other_cl_inv:
  assumes "read_done cl kv_map sn u s s'"
    and "NoPendingInView s"
    and "cl' ≠ cl"
  shows "views_of_s s' cl' = views_of_s s cl'"


lemma write_commit_views_of_s_other_cl_inv:
  assumes "write_commit cl kv_map cts sn u s s'"
    and "NoPendingInView s"
    and "cl' ≠ cl"
  shows "views_of_s s' cl' = views_of_s s cl'"


end
```