RootVoter technical description (draft)

Author: Ilya Tuzov (Universitat Politècnica de València)
Date/version: Feb. 2022 / v.1.0

Abstract: This document describes the architecture and a bare-metal API of the SELENE RootVoter v.2 cell.

Architecture

RootVoter is a small-footprint configurable HW module that allows detection of data corruption and timeout/hang failures in multicore SoC. RootVoter implements any M out of N voting scheme (MooN) configured at the software level at runtime. Its top-level architecture is depicted in Fig.1.

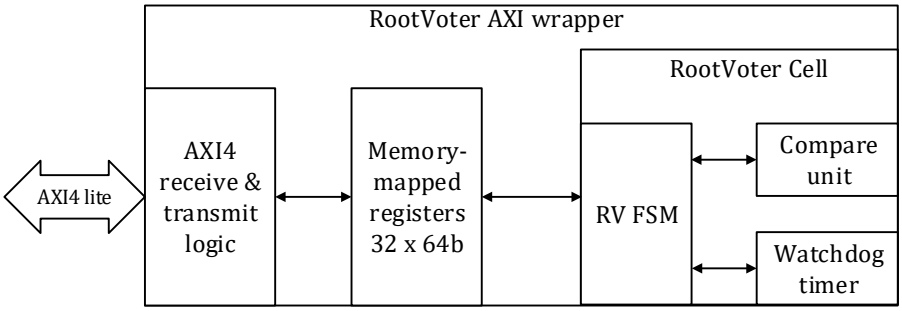

Figure 1 – Top-level architecture of the RootVoter cell

A core of RootVoter cell comprises:

- Configurable compare unit (CompareUnit.sv) that is in charge of detecting matches/mismatches between each pair of input datasets, as well as counting the number of such matches for each dataset.
- Configurable watchdog timer (down-counter) used for the detection of timeout failures;
- RV FSM that controls the voting process attending to the diagram depicted in Fig. 2.

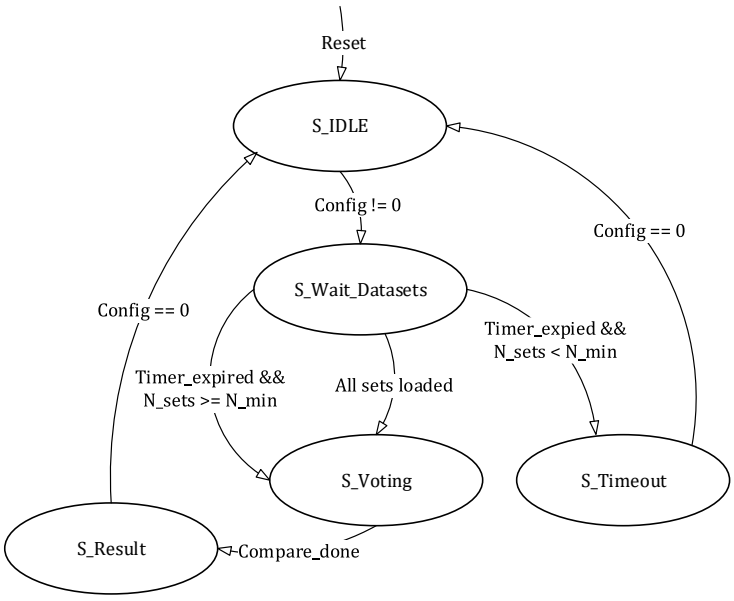

Figure 2 – RootVoter FSM logic

The accelerators_axi4 branch of the SELENE SoC includes four RootVoter cells, mapped onto the following base addresses (see rootvoter_v2/sw/RootVoter.h):

```
#define RVC0_BASE 0xfffc0100
#define RVC1_BASE 0xfffc0200
#define RVC2_BASE 0xfffc0300
#define RVC3_BASE 0xfffc0400
```

Memory-mapped registers

RootVoter is connected to the SoC via the AXI4-lite interface, and communicates with the SoC by means of 32 memory-mapped registers, detailed in Table 1. Please, note that base address of each RootVoter is aligned by 256 bytes, i.e. BASE_ADR = AXI_ADR & 12'hF00, LOCAL_ADR = AXI_ADR & 12'h0FF.

TABLE 1 – Description of RootVoter memory-mapped registers

	Offset (bytes)	(uint64* ptr)	Register	Details
BASE_ADR	+0	(+0)	config	(slv_reg[0] : write only)
	+8	(+1)	set[0] (A)	(slv_reg[1] : write only)
	+16	(+2)	set[1] (B)	(slv_reg[2] : write only)
	+24	(+3)	set[2] (C)	(slv_reg[3] : write only)
	+32	(+4)	set[3] (D)	(slv_reg[4] : write only)
	+40	(+5)	set[4] (E)	(slv_reg[5] : write only)
	+48	(+6)	set[5] (F)	(slv_reg[6] : write only)
	+56	(+7)	set[6] (G)	(slv_reg[7] : write only)
	+64	(+8)	set[7] (H)	(slv_reg[8] : write only)
	+72	(+9)	set[8] (I)	(slv_reg[9] : write only)
	+80	(+10)	set[9] (J)	(slv_reg[10] : write only)
	+88	(+11)	set[10] (K)	(slv_reg[11] : write only)
	+96	(+12)	set[11] (L)	(slv_reg[12] : write only)
	+104	(+13)	set[12] (M)	(slv_reg[13] : write only)
	+112	(+14)	set[13] (N)	(slv_reg[14] : write only)
	+120	(+15)	set[14] (O)	(slv_reg[15] : write only)
	+128	(+16)	set[15] (P)	(slv_reg[16] : write only)
	+136	(+17)	match_vector[63:0]	(slv_reg[17] : read only)
	+144	(+18)	match_vector[119:64]	(slv_reg[18] : read only)
	+152	(+19)	state	(slv_reg[19] : read only)
	+160	(+20)	status	(slv_reg[20] : read only)
	+168	(+21)	match_counters (16x4)	(slv_reg[21] : read only)
	+248	(+31)	reset control	(non-register: write only, 0xF to clear all registers)

1. Config register (Write-only):
   Voting mode is configured by software as "MooN", where:
   M = config[7:4]
   N = config[3:0]
   NOTE: RVC checks that (N <= MAX_DATASETS) and (M <= N) and (M >= 2),
   FSM passes to voting state only when these conditions are satisfied.

   The datasets should be loaded during "timeout" clock cycles after setting the config register, where
   timeout = config[39:8] (max clock cycles to wait for datasets)

Voting time depends on the number of used datasets N (config[3:0])

```
2 sets:   1 clk      10 sets:  45 clk
3 sets:   3 clk      11 sets:  55 clk
4 sets:   6 clk      12 sets:  66 clk
5 sets:  10 clk      13 sets:  78 clk
6 sets:  15 clk      14 sets:  91 clk
7 sets:  21 clk      15 sets: 105 clk
8 sets:  28 clk      16 sets: 120 clk
9 sets:  36 clk
```

2. Set registers (Write-only):
   16 datasets reserved in the memory map: slv_reg[1] to slv_reg[16]
   NOTE: actual number of set registers supported by RVC can be read from the state register
   (see HWInfo.MAX_DATASETS in p.5 below)

3. status register (read-only):

```
   [0:0]  ready flag (voting completed)
 [23:8]   timeout flags for each dataset (16 bits - one per dataset)
[39:24]   failure flags for each dataset (16 bits - one per dataset)
```

4. state register (read-only):

```
   [4:0]  FSM state: 1: idle, 2: wait_datasets, 4: voting, 8: timeout, 16: result
     [5]  compare_done
     [6]  compare_en
     [7]  compare_reset
  [11:8]  HWInfo.ID
 [16:12]  HWInfo.MAX_DATASETS
    [17]  HWInfo.LIST_FAILURES
    [18]  HWInfo.LIST_MATCHES
    [19]  HWInfo.COUNT_MATCHES
 [23:20]  HWInfo.RV_VERSION
 [31:24]  RESERVED (not used)
 [63:32]  SYNC WORD (0x55555555)
```

5. match_counters register (read-only):
Indicates the number of matches of i-th dataset with the rest of datasets (4 bits), where i = [0 to N]

```
  [3:0]  match_count for dataset[0]
  [7:4]  match_count for dataset[1]
    ...  …
[63:60]  match_count for dataset[15]
```

6. Match vector (read-only):
slv_reg[17] and slv_reg[18]
Comparison flags for each pair of datasets

Mapping of match vector bits for different N=[2:10]:

matchvec bit	pair N= 2	pair N= 3	pair N= 4	pair N= 5	pair N= 7	pair N= 8	pair N= 9	pair N=10
0	0: 1	1: 2	2: 3	3: 4	5: 6	6: 7	7: 8	8: 9
1		0: 2	1: 3	2: 4	4: 6	5: 7	6: 8	7: 9
2		0: 1	1: 2	2: 3	4: 5	5: 6	6: 7	7: 8
3			0: 3	1: 4	3: 6	4: 7	5: 8	6: 9
4			0: 2	1: 3	3: 5	4: 6	5: 7	6: 8
5			0: 1	1: 2	3: 4	4: 5	5: 6	6: 7
6				0: 4	2: 6	3: 7	4: 8	5: 9
7				0: 3	2: 5	3: 6	4: 7	5: 8
8				0: 2	2: 4	3: 5	4: 6	5: 7
9				0: 1	2: 3	3: 4	4: 5	5: 6
10					1: 6	2: 7	3: 8	4: 9
11					1: 5	2: 6	3: 7	4: 8
12					1: 4	2: 5	3: 6	4: 7
13					1: 3	2: 4	3: 5	4: 6
14					1: 2	2: 3	3: 4	4: 5
15					0: 6	1: 7	2: 8	3: 9
16					0: 5	1: 6	2: 7	3: 8
17					0: 4	1: 5	2: 6	3: 7
18					0: 3	1: 4	2: 5	3: 6
19					0: 2	1: 3	2: 4	3: 5
20					0: 1	1: 2	2: 3	3: 4
21						0: 7	1: 8	2: 9
22						0: 6	1: 7	2: 8
23						0: 5	1: 6	2: 7
24						0: 4	1: 5	2: 6
25						0: 3	1: 4	2: 5
26						0: 2	1: 3	2: 4
27						0: 1	1: 2	2: 3
28							0: 8	1: 9
29							0: 7	1: 8

matchvec bit	pair N= 2	pair N= 3	pair N= 4	pair N= 5	pair N= 7	pair N= 8	pair N= 9	pair N=10
30							0: 6	1: 7
31							0: 5	1: 6
32							0: 4	1: 5
33							0: 3	1: 4
34							0: 2	1: 3
35							0: 1	1: 2
36								0: 9
37								0: 8
38								0: 7
39								0: 6
40								0: 5
41								0: 4
42								0: 3
43								0: 2
44								0: 1

HW configuration parameters

Each RootVoter cell is instantiated in the SoC (selene_core.vhd) attending to the following template:

```
rootvoter_0 : rv_wrapper
generic map (
  RVC_ID              => 1,
  MAX_DATASETS        => RVC_0_MAX_DATASETS,
  COUNT_MATCHES       => RVC_0_COUNT_MATCHES,
  LIST_MATCHES        => RVC_0_LIST_MATCHES,
  LIST_FAILURES       => RVC_0_LIST_FAILURES
)
port map (
  clk       => clkm,
  rst_n     => rstn,
  axi_in    => accel_l_aximo(1),
  axi_out   => accel_l_aximi(1),
  interrupt => rv_interrupt(0) -- currently not used
);
```

RootVoter cell accepts several parameters, that can be adjusted for reducing area/utilization, and/or for extending voting statistics collected by the RootVoter:

- MAX_DATASETS – max. number of dataset registers supported by RV cell (2 to 16);
- RVC_0_LIST_FAILURES – enable/disable detection of failures for each dataset (failure flags in the status[39:24] (reg[20]));
- COUNT_MATCHES - (1/0): enable/disable counting of matches for each dataset in the match_counters register;
- LIST_MATCHES – enable/disable tracking of comparison flags for each pair of datasets in the match_vector register (reg[17] and reg[18]).

FPGA Utilization statistics

Utilization of FPGA (Virtex7) logic resources corresponding to different configurations of RootVoter is listed in Table 2.

Table 2 – Utilization of LUTs/FFs logic resources (Virtex7)

RVC size*	Top (RV with AXI wrapper)	RootVoter Cell	Compare Unit	Timer
Moo3 (max 3 sets)	389 / 355	320 / 66	246 / 21	55 / 32
Moo6 (max 6 sets)	616 / 601	503 / 81	411 / 33	53 / 32
Moo7 (max 7 sets)	689 / 679	517 / 86	429 / 37	51 / 32
Moo9 (max 9 sets)	769 / 838	618 / 96	517 / 45	54 / 32
Moo16 (max 16 sets)	1241 / 1416	943 / 143	811 / 86	54 / 32

* Under the default configuration of voting statistics: COUNT_MATCHES = 1, LIST_MATCHES = 0, LIST_FAILURES =1

Bare-metal API

A tiny API for baremetal applications is defined in the "safety/rootvoter_v2/sw/RootVoter.h". It includes three data-structures and five functions defined as follows:

```
//Hardware configuration of RV cell - read from the state register slv_reg[19]
typedef struct {
    uint8_t        id;            //RVC identifier (index)
    uint8_t        max_sets;      //Read-back attribute: Supported Number of set registers
    uint8_t        count_matches; //Read-back attribute: 1/0 RVC counts matches for each dataset in the reg[21]
    uint8_t        list_matches;  //Read-back attribute: 1/0 RVC provides comparison flags for each pair of datasets
    uint8_t        list_failures; //Read-back attribute: 1/0 RVC provides failure flags for each dataset in the reg[20]
    uint8_t        version;       //Read-back attribute: RVC version (equals 2 for the current version)
} RootVoterHWInf



// Software + Hardware configuration of RV cell (top descriptor of RV cell)
typedef struct {
    uint64_t*         base_adr;      // RVC base address
    RootVoterHWInf    inf;           // HW features supported by RVC (Read-only)
    uint8_t           mode;          // MooN voting scheme:
                                     //  mode[7:4] = M (min number of matching sets required for the agreement),
                                     //  mode[3:0] = N (total number of sets to vote)
    uint64_t          timeout;       // Max amount of CLK cycles to load all datasets
} RootVoterDescriptor


// Vote result for the last voting transaction
typedef struct {
    uint8_t        mode;         // Used MooN voting scheme:
    uint16_t       failvec;      // Agreement/failure status of each dataset (one bit per dataset):
                                 //  0 - pass (at least M-1 matches),
                                 //  1 - fail (less than M-1 matches)
    uint16_t       timeout;      // Timeout detection status (one bit per dataset)
    uint64_t       match_cnt;    // Match counters (64 bits - 4 bits per each dataset)
    uint8_t        agreement;    // Overall agreement status: 1 (reached) / 0 (not reached)
    uint64_t       matchvec_p1;  //raw (unparsed) comparison flags (low   part:  64 bits)
    uint64_t       matchvec_p2;  //raw (unparsed) comparison flags (upper part:  41 bits)
} VoteResult


//Synchronizes with RV cell at address BASEADR
//Retrieves RV cell features (HWInfo) from the state register
//Returns: 0 if sync successful, 1 otherwise
int RVC_sync(RootVoterDescriptor* RVC, uint64_t BASEADR);


//Clears RV cell registers
//Sets RV configuration (voting mode and timeout)
//after execution RV cell remains in wait_datasets state
//Returns: 0 if reset successful, 1 otherwise
int RVC_reset(RootVoterDescriptor* RVC, uint8_t mode, uint64_t max_wait_time);


//Loads a dataset to the set[dataset_id] register of RVC
int RVC_load_dataset(RootVoterDescriptor* RVC, uint8_t dataset_id, uint64_t dataset);


//Waits for completion of RVC voting (polling mode)
//Returns voting statistics in form of VoteResult structure
VoteResult RVC_vote(RootVoterDescriptor* RVC);


//Logs VoteResult to std out
void print_vote_result(VoteResult* v);
```

The aforementioned functions are implemented in the "safety/rootvoter_v2/sw/RootVoter.c".

API usage

Example baremetal applications that use this API are defined in:

- “safety/rootvoter_v2/sw/test_singlecore.c” – single core RV example
- “safety/rootvoter_v2/sw/test_multicore.c” – multicore core RV example

For instance, the following code fragment (listing 1):

- instantiates an RV descriptor;
- synchronizes it with RV cell at the base address RVC0_BASE (extracts HWInfo);
- configures the voting mode 2oo3, and a timeout of 1000 clock cycles;
- simulates loading of datasets to the set registers 0, 1, and 2;
- Initiates voting process and retrieves voting results in polling mode;
- logs the voting results to the console.

Listing 1. Example application that uses baremetal RootVoter API

```
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include "RootVoter.h"

int main() {

    RootVoterDescriptor RVC0;
    RVC_sync(&RVC0, RVC0_BASE);

    printf("Features of RootVoter Cell version %d:\n\t
            Base address %08x:\n\t
            id=%2d\n\tMaxDatasets=%2d\n\t
            count_matches=%2s\n\t
            list_matches=%2s\n\t
            list_failures=%2s\n",

            RVC0.inf.version,
            RVC0.base_adr,
            RVC0.inf.id,
            RVC0.inf.max_sets,
            RVC0.inf.count_matches==1?"YES":"NO",
            RVC0.inf.list_matches ==1?"YES":"NO",
            RVC0.inf.list_failures==1?"YES":"NO");

    VoteResult res;

    printf("\n\nTest-1: RVC (id=%2d) (2oo3) expected: pass/pass/pass \n", RVC0.inf.id);
    RVC_reset(&RVC0, 0x23, 1000);
    RVC_load_dataset(&RVC0, 0, 0xf1f2f3f4cafebabe);    //assume core-0, core-1 and core-2 have loaded valid datasets
    RVC_load_dataset(&RVC0, 1, 0xf1f2f3f4cafebabe);
    RVC_load_dataset(&RVC0, 2, 0xf1f2f3f4cafebabe);
    res = RVC_vote(&RVC0);
    print_vote_result(&res);


    printf("\n\nTest-2: RVC (id=%2d) (2oo3) expected: pass/pass/timeout \n", RVC0.inf.id);
    RVC_reset(&RVC0, 0x23, 1000);
    RVC_load_dataset(&RVC0, 0, 0xa1a2a3a4a5a6a7a8);    //assume only core-0 and core-1 have loaded their datasets
    RVC_load_dataset(&RVC0, 1, 0xa1a2a3a4a5a6a7a8);
    res = RVC_vote(&RVC0);
    print_vote_result(&res);


    printf("\n\nTest-3: RVC (id=%2d) (2oo3) expected:  timeout/fail/timeout \n", RVC0.inf.id);
    RVC_reset(&RVC0, 0x23, 1000);
    RVC_load_dataset(&RVC0, 1, 0xc1c2c3c4c5c6c7c8);    //assume only core-1 has loaded its dataset
    res = RVC_vote(&RVC0);
    print_vote_result(&res);


    printf("\n\nTest-4: RVC (id=%2d) (2oo3) expected: pass/fail/pass \n", RVC0.inf.id);
    RVC_reset(&RVC0, 0x23, 1000);
    RVC_load_dataset(&RVC0, 0, 0xf1f2f3f4f5f6f7f8);    //assume core-0 and core-2 have loaded valid datasets
    RVC_load_dataset(&RVC0, 1, 0x1112131415161718);    //assume core-1 loads invalid dataset
    RVC_load_dataset(&RVC0, 2, 0xf1f2f3f4f5f6f7f8);
    res = RVC_vote(&RVC0);
    print_vote_result(&res);

    return 0;
}
```

Simulation (QuestaSim)

1. Compile the design:

```
> make map_xilinx_7series_lib
> make selene-sim
```

2. Run the simulation:

```
> vsim -i -quiet work.glbl -L secureip -L unisims_ver -t 1ps “+notimingchecks” -voptargs=”+acc” \
    testbench -gdisas=0 -GSIM_COLLISION_CHECK=GENERATE_X_ONLY -gUSE_MIG_INTERFACE_MODEL=true \
   -G/testbench/cpu/cpu/core0/mem0/mig_gen/sim_ram_gen/axi_mem/rb/fname=../../safety/rootvoter_v2/sw/hello_singlecore.srec
```

3. Resulting log (transcript):

```
# NOELV/GRLIB VCU118 Demonstration design
# GRLIB Version 2021.2.0, build 4267
# Target technology: uscaleplus, memory library: uscaleplus
# ahbctrl: AHB arbiter/multiplexer rev 1
# ahbctrl: Common I/O area at 0xfff00000, 1 Mbyte
# ahbctrl: AHB masters: 11, AHB slaves: 16
... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...

... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...

# Features of RootVoter Cell version 2:
#   Base address fffc0100:
#   id= 1
#   MaxDatasets= 9
#   count_matches=YES
#   list_matches=NO
#   list_failures=YES
#
#
# Test-1: RVC (id= 1) (2oo3) expected: pass/pass/pass
## Waiting for datasets
## All datasets available before timeout: voting
## Voting completed
## State_result
# RVC Mode = 2/3, Timeout=0000, Failvec=0000, Agreement=YES
# set[ 0]: result = Pass, cnt =  2
# set[ 1]: result = Pass, cnt =  2
# set[ 2]: result = Pass, cnt =  2
#
#
# Test-2: RVC (id= 1) (2oo3) expected: pass/pass/timeout
## Waiting for datasets
## Enough datasets available after timeout: attempting to vote
## Voting completed
## State_result
# RVC Mode = 2/3, Timeout=0004, Failvec=0004, Agreement=YES
# set[ 0]: result = Pass, cnt =  1
# set[ 1]: result = Pass, cnt =  1
# set[ 2]: result = Timeout, cnt =  0
#
#
# Test-3: RVC (id= 1) (2oo3) expected:  timeout/fail/timeout
## State_result
## Waiting for datasets
## Not enough datasets available after timeout: voting not possible
## State_timeout
# RVC Mode = 2/3, Timeout=0005, Failvec=0007, Agreement=NO
# set[ 0]: result = Timeout, cnt =  0
# set[ 1]: result = Fail, cnt =  0
# set[ 2]: result = Timeout, cnt =  0
#
# Test-4: RVC (id= 1) (2oo3) expected: pass/fail/pass
## Waiting for datasets
## All datasets available before timeout: voting
## Voting completed
## State_result
# RVC Mode = 2/3, Timeout=0000, Failvec=0002, Agreement=YES
# set[ 0]: result = Pass, cnt =  1
# set[ 1]: result = Fail, cnt =  0
# set[ 2]: result = Pass, cnt =  1
# Hart 0 halted after successful RAM test binary execution.
```