|  |
| --- |
| **Internal Specification** |

|  |
| --- |
| **Development of ADC VMON Secondary Error Generator (AVSEG) model**  (v1.1) |

|  |
| --- |
| **Summary:** |
| This document describes the Detail Design Specification of ADC VMON Secondary Error Generator (AVSEG) model. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Reference Manuals** | | | | |
| **No.** | **Title name** | **Document number** | **Description** | **Path** |
| 1 | SC-HEAP\_E3 common requirement (v1.0) | - | The common requirement  (***File***: Common\_Requirement\_RVC.pdf) | **DMS:**  Documents/1. General Documents/010\_ENG/140\_FrontEnd/Project/01\_SLD/2\_SLD\_Project/Model\_Documents/02\_MCS\_Project/From\_MCS |
| 2 | SC-HEAP\_E3 Modeling guideline (Rev. 4.00) | IDF-14-010278-01 | This document describes the Guideline for peripheral macro  development which is connected to SC-HEAP\_E3 simulator  (***File***: SC-HEAP\_E3\_Modeling\_Guideline.pdf) |
| 3 | SC-HEAP\_E3 BUS I/F outline | LLWEB-00010925  ZSG-F31-12-0029-01 | The document describes the outline of bus I/F applied to SC-HEAP\_E3  (***File***: scheap\_e3\_bus\_if\_outline\_E.pdf) |
| 4 | SC-HEAP\_E3 PYTHON I/F function specification (v2.0) | LLWEB-00105192  MSS-SG-12-0062-02 | The document describes how to use python interface  (***File***: SC-HEAP\_E3 Python IF\_t.pdf) |
| 5 | M40PF common requirement (Rev1.10) | REQ-SLD-12-010 | The common requirement for M40PF models  (***File***: REQ-SLD-12010\_M40PF\_Common.ppt) | **DMS:**  Documents/1. General Documents/010\_ENG/140\_FrontEnd/Project/01\_SLD/2\_SLD\_Project/Model\_Documents/01\_Project\_Document\_Management/REQ/2012 |
| 6 | RH850 E2x/ADCH model development: Requirement Specification (Rev1.0) | REQ-SLD-16004 | Detail requirement of ADCH models  (***File***: REQ-SLD-16004\_E2x\_ADCH\_models.pptx) | **DMS:**  Documents/1. General Documents/010\_ENG/140\_FrontEnd/Project/01\_SLD/2\_SLD\_Project/Model\_Documents/01\_Project\_Document\_Management/REQ/2016 |
| 7 | RH850/E2x-FCC1 user’s manual (Rev.0.10) | - | User’s manual for RH850/E2x-FCC1  (***File***: NoSecurity\_r01uh0641ej0010\_rh850e2x-fcc1.pdf) | **Server:** /shsv/sld/ipp/From\_RT/20160610\_SCHEAP/ |
| 8 | AVSEG’s implemented classes: detail specification | - | Detail specification of AVSEG’s implemented classes  (***File***: INT-SLD-16004\_refman.pdf) | **DMS:**  Documents/1. General Documents/010\_ENG/140\_FrontEnd/Project/01\_SLD/2\_SLD\_Project/Model\_Documents/01\_Project\_Document\_Management/INT/2016/PDF |

Table of Contents

[1. Model summary 6](#_Toc461011335)

[2. Supported features 6](#_Toc461011336)

[3. Block diagram 7](#_Toc461011337)

[4. List of implemented registers 8](#_Toc461011338)

[5. Port behavior 11](#_Toc461011339)

[5.1. List of implemented ports 11](#_Toc461011340)

[5.2. Clock 12](#_Toc461011341)

[5.3. Reset 12](#_Toc461011342)

[5.4. Inputs to noise filter and outputs upper/lower error pulse 13](#_Toc461011343)

[6. Direction for users 15](#_Toc461011344)

[6.1. File structures 15](#_Toc461011345)

[6.2. Input/Output file 16](#_Toc461011346)

[6.3. How to connect Verification Environment 16](#_Toc461011347)

[6.4. Commands and parameters 16](#_Toc461011348)

[6.5. Message style 19](#_Toc461011349)

[6.6. Defined macro and template 22](#_Toc461011350)

[7. Flow diagram 23](#_Toc461011351)

[7.1. Sequence flow 23](#_Toc461011352)

[7.2. State diagram 24](#_Toc461011353)

[7.3. Reset flow 25](#_Toc461011354)

[7.4. Command/parameter configuration operation flow 26](#_Toc461011355)

[7.5. Noise filter and upper/lower error pulse output flow 28](#_Toc461011356)

[8. Functions description 30](#_Toc461011357)

[9. Limitation 31](#_Toc461011358)

**Index of Figures**

[Figure 3.1: General block diagram 7](#_Toc461011359)

[Figure 5.1: Clock 12](#_Toc461011360)

[Figure 5.2: Inputs to noise filter and outputs upper/lower error pulse 13](#_Toc461011361)

[Figure 6.1: File structure 15](#_Toc461011362)

[Figure 6.2: An example of python interface usage 17](#_Toc461011363)

[Figure 6.3: An example of command interface usage 18](#_Toc461011364)

[Figure 7.1: Sequence flow 23](#_Toc461011365)

[Figure 7.2: State diagram 24](#_Toc461011366)

[Figure 7.3: Reset flow 25](#_Toc461011367)

[Figure 7.4: Relationship between software reset and hardware reset 25](#_Toc461011368)

[Figure 7.5: Command/parameter configuration operation flow (1/4) 26](#_Toc461011369)

[Figure 7.6: Command/parameter configuration operation flow (2/4) 26](#_Toc461011370)

[Figure 7.7: Command/parameter configuration operation flow (3/4) 27](#_Toc461011371)

[Figure 7.8: Command/parameter configuration operation flow (4/4) 27](#_Toc461011372)

[Figure 7.9: Noise filter and upper/lower error pulse output flow 28](#_Toc461011373)

**Index of Tables**

[Table 2.1: Feature of AVSEG model 6](#_Toc461011374)

[Table 4.1: List of implemented registers 8](#_Toc461011375)

[Table 5.1: List of implemented ports 11](#_Toc461011376)

[Table 6.1: File description 15](#_Toc461011377)

[Table 6.2: List of parameters 16](#_Toc461011378)

[Table 6.3: List of commands 16](#_Toc461011379)

[Table 6.4: Dumping model command help message description 19](#_Toc461011380)

[Table 6.5: Dumping register help message description 19](#_Toc461011381)

[Table 6.6: Dumping target help message description 19](#_Toc461011382)

[Table 6.7: Dumping Register RW message description 20](#_Toc461011383)

[Table 6.8: Error and debugging message description 20](#_Toc461011384)

[Table 6.9: Error and debug messages 20](#_Toc461011385)

[Table 8.1: Functions of Cavseg class 30](#_Toc461011386)

[Table 8.2: Functions of Cavseg\_core class 31](#_Toc461011387)

# Model summary

* ADC VMON Secondary Error Generator (AVSEG) is equivalent to the function which notify an upper error pulse (secondary HDET) and a lower error pulse (Secondary LDET) of each power supply (VCC, EVCC and VDD) to ECM in the secondary power supply voltage monitor. The input signals are passed through a filter to reduce noise.
* This model is developed for RH850/E2x.
* Registers of model can be accessed to read/write via target socket (of TLM target interface).
* Both loosely time mode (LT) and approximately time (AT) mode are supported.
* This model supports little endian mode as the endian of APB bus interface.

***Note****: Hereafter, ADC VMON Secondary Error Generator model is simply called “AVSEG”.*

# Supported features

Table 2.1: Feature of AVSEG model

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Description** | | **HWM chapter** |
| **Hardware** | **Model** |
| Max frequency of APB bus clock (CLK\_LSB clock) | 40 MHz | Unlimited frequency. There is no setting condition. | *15.2/Table 15.4 (ref[7])* |
| Read/Write registers | Use bus interface | Use TLM target socket | *-* |
| Reset | Hardware reset (assert/deassert reset signal) | Hardware reset (assert/deassert reset signal)  Software reset (set by command AssertReset) (\*) | *-* |
| Main function | Upper Error Pulse Control:  Notify an upper error pulse whether or not an upper bound of each power supply (VCC, EVCC and VDD) is being exceeded. | <- | *34.9 (ref[7])* |
| Lower Error Pulse Control:  Notify a lower error pulse whether or not a lower bound of each power supply (VCC, EVCC and VDD) is being exceeded. | <- |
| Noise Filter: The upper and lower error pulses are generated by signals passed through a filter to reduce noise | <- |

***Notes****:*

*- (\*) This command is described in Chapter 6.4*

*- The symbol “<-” means that these features are supported as description in the hardware manual.*

*- All features described in HWM (ref[7]/Chapter 34.9) are supported in model.*

# Block diagram

Figure 3.1: General block diagram

**AVSEG**

**Command IF**

Commands & Parameters

CLK\_LSB

**Clock handler**

**Input handler**

vculmo

vcllmo

vcend

40

**Reset handler**

PRESETn

**AVSEG\_Core**

**Register**

m\_tgt\_sockets[0]

**Output handler**

evccshdet

evccsldet

vccshdet

vccsldet

vddshdet

vddsldet

TLM socket

external port

API call/internal port

***Explanation***:

* AVSEG model has a target socket used for read/write accessing model’s registers. Data from bus is transferred to this model via TLM target interface (m\_tgt\_sockets[0]).
* AVSEG includes 6 blocks:
* “Clock handler” block receives external input clock (CLK\_LSB) and provides this clock to other blocks.
* “AVSEG\_Core” block stores registers and handles read/write accessing to registers. Besides, this block controls model’s operation (e.g: upper/lower error pulse control and noise filter).
* “Input handler” block receives external input signals and transfer them to “AVSEG\_Core” block.
* “Reset Handler” block handles reset signal (PRESETn) and correlative reset commands.
* “Command IF” block handles commands and parameters which are input from users.
* After input signals are received in the “Input handler” block, “AVSEG\_Core” block will process operation and notify operation result to “Output handler” block. The “Output handler” block will output signals to other model.

# List of implemented registers

Table 4.1: List of implemented registers

| **Register name** (1\*) | **Address offset** | **Initial value** | **Size (byte)** | **Write Access size (bit)** | **Read Access size (bit)** | **R/W** | **Bit position** | **Bit name** (1\*) | **Explanation** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **VCCCHSCR**  Input Channel Selector VCC | 0x00 | 0x0 | 4 | 32 | 8|16|32 | R|W | 7:0  2:0 | CHS | **Input Channel Selector**  These bits select the channel used for generating error of VCC in the secondary power supply voltage monitor.  + 0x00: Selects ADCH0 virtual channel 0  :  + 0x27: Selects ADCH0 virtual channel 39  + 0x28 to 0xFF: Reserved (3\*) |
| **VCCCNTCR**  Filter Counter Control VCC | 0x04 | 0x101 | 4 | 32 | 8|16|32 | R|W | 28 | ENB | **Filter Enable**  This bit enable AVSEG for VCC functions  + 0: DISABLED  + 1: Enables the filter |
| 11:8 | NRMCNT (2\*) | **Recovery Counter Settings**  These bits control the number of counts until the signal is considered recovered.  Once the signal is considered out of bounds, it will not be recovered until the set number of consecutive non-error pulses is detected.  The counter value is reset when the VCCCNTCR register is written to.  + 0x1: Recovery after the signal is within the boundaries for 1 count  :  + 0xF: Recovery after the signal is within the boundaries for 15 consecutive counts |
| 3:0 | ERRCNT (2\*) | **Error Counter Control**  These bits control the number of counts until the signal is considered out of bounds.  The signal is considered out of bounds once the set number of consecutive error pulses is detected.  The counter value is reset when the VCCCNTCR register is written to.  + 0x1: Out of bounds when the signal violates the boundaries for 1 count  :  + 0xF: Out of bounds when the signal violates the boundaries for 15 consecutive counts |
| **EVCCCHSCR**  Input Channel Selector EVCC | 0x10 | 0x0 | 4 | 32 | 8|16|32 | R|W | 7:0  2:0 | CHS | **Input Channel Selector**  These bits select the channel used for generating error of EVCC in the secondary power supply voltage monitor.  + 0x00: Selects ADCH0 virtual channel 0  :  + 0x27: Selects ADCH0 virtual channel 39  + 0x28 to 0xFF: Reserved (3\*) |
| **EVCCCNTCR**  Filter Counter Control EVCC | 0x14 | 0x101 | 4 | 32 | 8|16|32 | R|W | 28 | ENB | **Filter Enable**  This bit enable AVSEG for EVCC functions  + 0: DISABLED  + 1: Enables the filter |
| 11:8 | NRMCNT (2\*) | **Recovery Counter Settings**  These bits control the number of counts until the signal is considered recovered.  Once the signal is considered out of bounds, it will not be recovered until the set number of consecutive non-error pulses is detected.  The counter value is reset when the EVCCCNTCR register is written to.  + 0x1: Recovery after the signal is within the boundaries for 1 count  :  + 0xF: Recovery after the signal is within the boundaries for 15 consecutive counts |
| 3:0 | ERRCNT (2\*) | **Error Counter Control**  These bits control the number of counts until the signal is considered out of bounds.  The signal is considered out of bounds once the set number of consecutive error pulses is detected.  The counter value is reset when the EVCCCNTCR register is written to.  + 0x1: Out of bounds when the signal violates the boundaries for 1 count  :  + 0xF: Out of bounds when the signal violates the boundaries for 15 consecutive counts |
| **VDDCHSCR**  Input Channel Selector VDD | 0x20 | 0x0 | 4 | 32 | 8|16|32 | R|W | 7:0  2:0 | CHS | **Input Channel Selector**  These bits select the channel used for generating error of VDD in the secondary power supply voltage monitor.  + 0x00: Selects ADCH0 virtual channel 0  :  + 0x27: Selects ADCH0 virtual channel 39  + 0x28 to 0xFF: Reserved (3\*) |
| **VDDCNTCR**  Filter Counter Control VDD | 0x24 | 0x101 | 4 | 32 | 8|16|32 | R|W | 28 | ENB | **Filter Enable**  This bit enable AVSEG for VDD functions  + 0: DISABLED  + 1: Enables the filter |
| 11:8 | NRMCNT (2\*) | **Recovery Counter Settings**  These bits control the number of counts until the signal is considered recovered.  Once the signal is considered out of bounds, it will not be recovered until the set number of consecutive non-error pulses is detected.  The counter value is reset when the VDDCNTCR register is written to.  + 0x1: Recovery after the signal is within the boundaries for 1 count  :  + 0xF: Recovery after the signal is within the boundaries for 15 consecutive counts |
| 3:0 | ERRCNT (2\*) | **Error Counter Control**  These bits control the number of counts until the signal is considered out of bounds.  The signal is considered out of bounds once the set number of consecutive error pulses is detected.  The counter value is reset when the VDDCNTCR register is written to.  + 0x1: Out of bounds when the signal violates the boundaries for 1 count  :  + 0xF: Out of bounds when the signal violates the boundaries for 15 consecutive counts |

***Notes***: *- (1\*) Register name/Bit name is name used in model. It may be not exactly same as the name used in hardware ref[7]/Chapter 34.9.2.*

*- (2\*) Write “0” to this bit is prohibited. A warning message is dumped and previous value is kept.*

*- (3\*) When users write reserved value to this bit, there is no input channel selected. Model’s operation is not processed.*

*- All registers described in HWM (ref[7]/Chapter 34.9.2) are supported in model.*

# Port behavior

## List of implemented ports

Table 5.1: List of implemented ports

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Port name** | **I/O** | **Type** | **Initial** | **Active** | **Synchronous clock** | **Description** |
| ***Clock and Reset*** | | | | | | |
| CLK\_LSB | In | sc\_in<sc\_dt::uint64> | - | - | - | Bus clock |
| PRESETn | In | sc\_in<bool> | - | (1\*) | CLK\_LSB | Bus reset |
| ***Peripheral bus*** | | | | | | |
| m\_tgt\_sockets[0] | In/Out | tlm::tlm\_target\_socket | - | - | CLK\_LSB | TLM target socket |
| ***Input and Output ports*** | | | | | | |
| vcend[39:0] | In | sc\_in<bool> \* | - | High | CLK\_LSB | Virtual Channel (0 -> 39) A/D Conversion End Notice of ADCH0 unit (SAR-ADC) |
| vculmo | In | sc\_in<bool> | - | High | CLK\_LSB | Virtual Channel Upper Limit Exceeds Notice of ADCH0 unit (SAR-ADC) |
| vcllmo | In | sc\_in<bool> | - | High | CLK\_LSB | Virtual Channel Lower Limit Exceeds Notice of ADCH0 unit (SAR-ADC) |
| evccshdet | Out | sc\_out<bool> | False | High | CLK\_LSB | EVCC Secondary HDET |
| evccsldet | Out | sc\_out<bool> | False | High | CLK\_LSB | EVCC Secondary LDET |
| vccshdet | Out | sc\_out<bool> | False | High | CLK\_LSB | VCC Secondary HDET |
| vccsldet | Out | sc\_out<bool> | False | High | CLK\_LSB | VCC Secondary LDET |
| vddshdet | Out | sc\_out<bool> | False | High | CLK\_LSB | VDD Secondary HDET |
| vddsldet | Out | sc\_out<bool> | False | High | CLK\_LSB | VDD Secondary LDET |

***Note***: *- (1\*) Active level of PRESETn depend on defining macro IS\_RESET\_ACTIVE\_LOW (refer to Chapter 6.6).*

## Clock

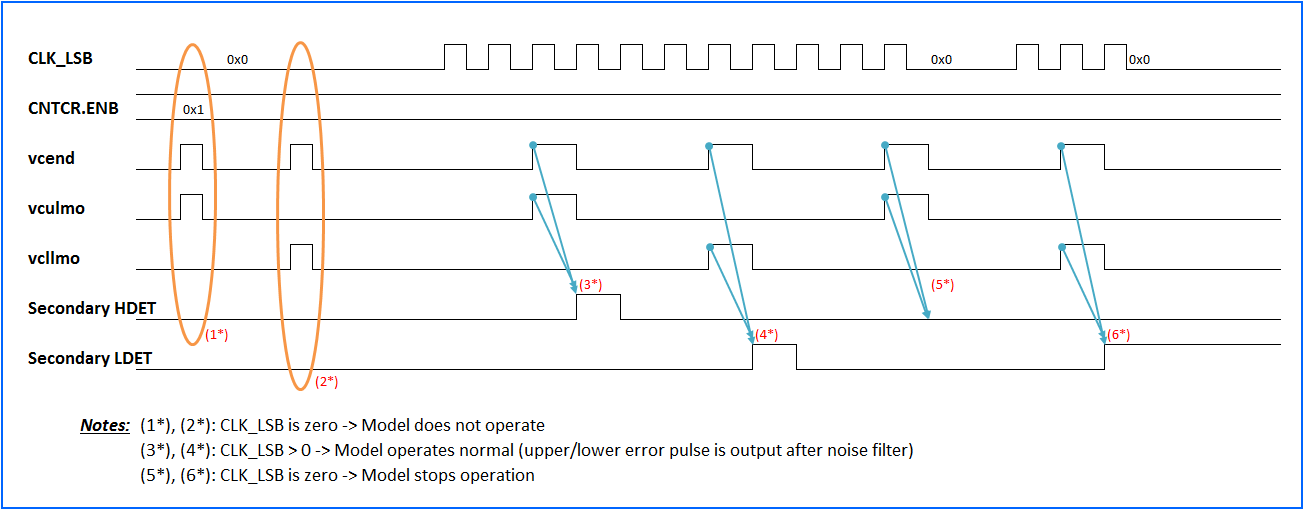


Figure 5.1: Clock

***Explanation***:

* AVSEG model has 1 clock port (CLK\_LSB) used to receive clock frequency signal (Hz) which is used to calculate the time required for transaction (register accessing) and internal behavior processing (noise filter and error pulse output).
* When CLK\_LSB clock frequency is zero:
  + Model does not start operation: Upper/lower error pulse control and noise filter operation are not processed when receive input signals vcend, vculmo, vcllmo although the filter is enabled and input channel is valid. ((1\*) and (2\*))
  + Model stops operation: Cancel all the process without any data member initialization. ((5\*) and (6\*))
  + No error message dumped. If any input ports which notify the internal process are active, a warning message is dumped.
  + A core dump error may be occurred when access to register (AT mode) due to CLK\_LSB is register access clock.
  + Reset operation is executed immediately when PRESETn is activated.

## Reset

* This model has 1 reset port (PRESETn) used to reset registers and operation of model. Parameters in Table 6.2 are not effected by reset operation (refer to chapter 7.3 for detail of reset operation).
* PRESETn port is synchronized with CLK\_LSB clock when CLK\_LSB frequency is different from zero value.
* If CLK\_LSB frequency is zero value, reset operation is executed immediately when PRESETn is activated.
* Active level of PRESETn depend on defining the macro IS\_RESET\_ACTIVE\_LOW (refer to Chapter 6.6).

## Inputs to noise filter and outputs upper/lower error pulse

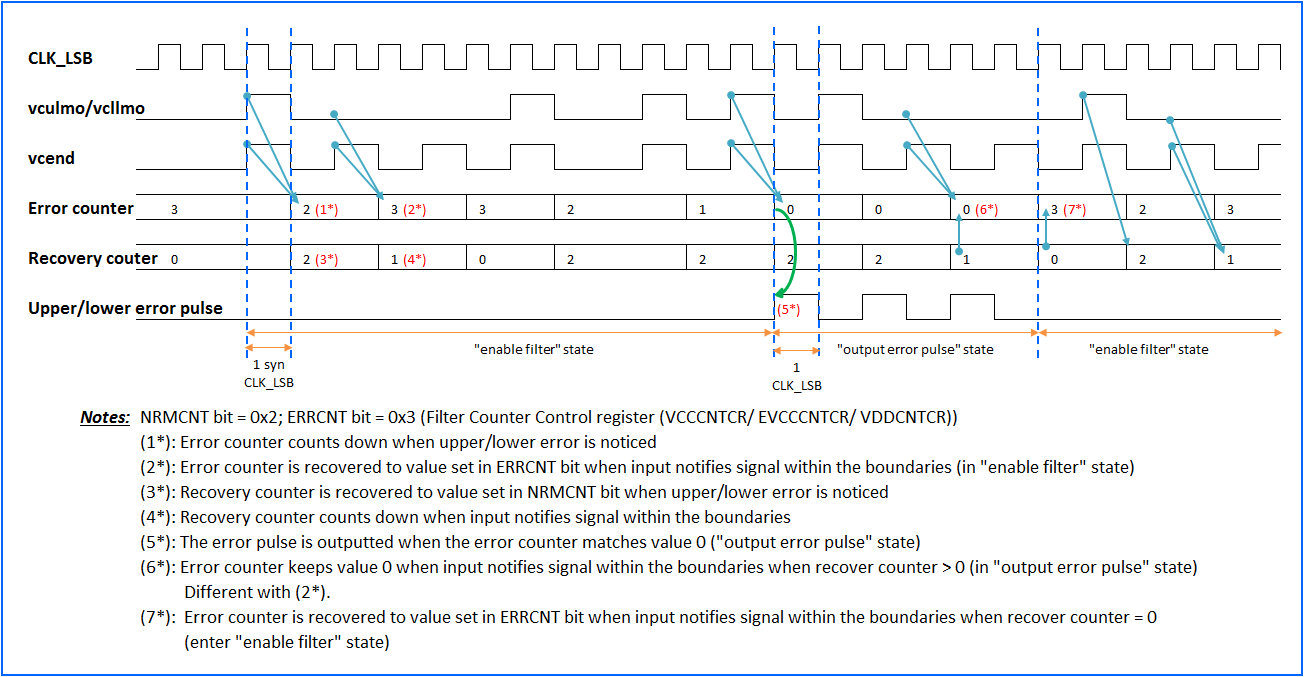


Figure 5.2: Inputs to noise filter and outputs upper/lower error pulse

***Explanation***:

* AVSEG model receive an A/D conversion result via input ports vculmo, vcllmo and vcend.
  + The virtual channel (0->39) conversion end notice signals are notified to vcend.
  + When the A/D conversion result has increased above the upper limit value in each virtual channel, a notice signal is notified to vculmo.
  + When the A/D conversion result has decreased below the lower limit in each virtual channel, a notice signal is notified to vcllmo.
* Noise is filtered and reduced by counting the number of times the ADC boundary values are exceeded using the input error pulses (vcend, vculmo and vculmo).
  + The Filter Counter Control register (VCCCNTCR/ EVCCCNTCR/ VDDCNTCR) controls the noise filter operation. When ENB bit in this register is 0, all AVSEG for correlative power supply (VCC, EVCC, VDD) functions are disabled (Noise filter is disabled and upper/lower error pulse is not outputted). When ENB bit is set to 1, the noise filter for correlative power supply (VCC/ EVCC/ VDD) is enabled.
  + When upper/lower error is noticed, at the next raising edge of CLK\_LSB, the error counter counts down (discount 1) (1\*) and the recovery counter is recovered to value set in NRMCNT bit (of Filter Counter Control register) (3\*).
  + If the signal violates the boundaries for consecutive counts which specified by ERRCNT bit (in Filter Counter Control register), the error pulse can be outputted (5\*). That is mean, when the error counter matches value 0, model enters “output error pulse” state and the error pulse is outputted (1 pulse = 1 CLK\_LSB).
  + If model receives the signal within boundaries, at the next raising edge of CLK\_LSB, the recovery counter counts down (discount 1) (4\*). And, the error counter is recovered to value set in ERRCNT bit, when model in “enable filter” state (2\*).
  + When model in “output error pulse” state, if the signal within the boundaries for consecutive counts which specified by NRMCNT bit, the error counter is recovered to value set in ERRCNT bit. That is mean, when the recovery counter matches value 0 in “output error pulse” state; the error counter equals value set in ERRCNT bit and model enters “enable filter” state (7\*). Otherwise (the recovery counter > 0), the error counter keeps value 0 (6\*).
* After noise is filtered, upper/lower error pulse is outputted via secondary HDET/LDET output ports (evccshdet/evccsldet, vccshdet/vccsldet, vddshdet/vddsldet).
  + According setting in CHS bit in Input Channel Selector register (VCCCHSCR/ EVCCCHSCR/ VDDCHSCR), the upper/lower error pulse of selected channel is detected.

***Notes:*** - *“The input of both upper and lower pulses into the same module at the same time is prohibited”.*

*- The range of error counter is [ERRCTN value:0]. The error counter keeps value 0 when its down counting already reached 0 and (1\*) occurs (minimum value).*

*- The range of recovery counter is [NRMCNT value:0] . The recovery counter keeps value 0 when its down counting already reached 0 and (4\*) occurs (minimum value).*

*- Refer to Chapter 7.5 for more detail about Noise filter and upper/lower error pulse outputs operation.*

# Direction for users

## File structures

***Legend:***

A

B

File A includes file B

A

B

The function were declared in file A is defined in file B

PY\_AVSEG.h

Python.h

PY\_AVSEG.cpp

avseg.h

avseg.cpp

avseg\_core.h

avseg.h

avseg\_cmdif.h

tlm\_tgt\_if.h

tlm\_if.h

tlm\_tgt\_if\_unit.h

tlm\_if\_command.h

re\_security.h

re\_security.cpp

re\_define.h

avseg\_core.h

avseg\_regif.h

re\_register.h

avseg\_regif.cpp

reg\_super.h

re\_register.cpp

avseg\_core.cpp

avseg.h

Figure 6.1: File structure

Table 6.1: File description

|  |  |  |  |
| --- | --- | --- | --- |
| **File name** | **CVS tag** | **Implementation** | **Description** |
| avseg.h | - | Developed | Header file of AVSEG wrapper |
| avseg.cpp | - | Developed | Implementation file of AVSEG wrapper |
| avseg\_core.h | - | Developed | Header file of AVSEG function |
| avseg\_core.cpp | - | Developed | Implementation file of AVSEG function |
| avseg\_regif.h | - | Generated (1\*) | Header file of AVSEG register interface |
| avseg\_regif.cpp | - | Generated (1\*) | Implementation file of AVSEG register interface |
| avseg\_cmdif.h | - | Generated (1\*) | Implementation file of command interface and re\_printf function |
| PY\_AVSEG.h | - | Generated (1\*) | Header file of AVSEG Python interface |
| PY\_AVSEG.cpp | - | Generated (1\*) | Implementation file of AVSEG Python interface |
| Python.h | - | Reused | Header file of python library |
| re\_register.h | v2013\_05\_28 | Reused | Header file of the re\_register class |
| re\_register.cpp | Reused | Implement the attributes and the operations of common register class |
| reg\_super.h | Reused | General class for models to access to the memory array |
| tlm\_tgt\_if.h | v2016\_08\_11\_b\_frm\_v2014\_04\_02 | Reused | Header file of the tlm\_tgt\_if class |
| tlm\_if.h | Reused | Header file of the tlm\_if class |
| tlm\_tgt\_if\_unit.h | Reused | Header file of the tlm\_tgt\_if\_unit class |
| tlm\_if\_command.h | Reused | Header file of the tlm\_if\_command class |
| re\_security.h | v100419 | Reused | Additional file of tlm\_ini\_if class and tlm\_tgt\_if class |
| re\_security.cpp | Reused |
| re\_define.h | v1.2  (2012/01/30) | Reused | Define common define macro, enum and so on |

***Note***: *- (1\*) Files avseg\_regif.h/.cpp are generated from Register IF Generator (v2014\_12\_01). Files avseg\_cmdif.h and PY\_AVSEG.h/.cpp are generated from Command IF Generator (v2015\_02\_12). After that they are modified for suitableness of the model.*

## Input/Output file

There is no input or output file.

## How to connect Verification Environment

There are 3 basic steps to connect AVSEG model to a verification environment.

* Step 1: Declare an instance of AVSEG class.
* Step 2: Bind the TLM target socket m\_tgt\_sockets[0].
* Step 3: Bind the input/output ports.

## Commands and parameters

Table 6.2: List of parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | **Parameter** | **Default** | **Description** |
| command | MessageLevel <fatal|error|warning|info> | fatal|error | Select debug message level (“fatal”, “error”, “warning” and “info”) (\*)  One or more than levels can be connected by vertical bar (Example “fatal|error”) |
| reg | MessageLevel <fatal|error|warning|info> |
| reg | <register\_name> MessageLevel <fatal|error|warning|info> |
| reg | DumpRegisterRW <true/false> | false | Enable/disable dumping access register (\*\*)  + false: Not dump register access information  + true: Dump register access information |

Table 6.3: List of commands

|  |  |  |
| --- | --- | --- |
| **Category** | **Command** | **Description** |
| command | AssertReset <rst\_name> <start\_time> <period> | Assert and deassert reset signal  + <rst\_name>: name of reset signal  + <start-time>: the time until asserting reset signal from current time. The unit is “ns”  + <period>: the time from asserting reset signal to de-assert it. The unit is “ns” |
| command | SetCLKfreq <clk\_name> <clk\_freq> <unit> | Set clock frequency value  + <clk\_name>: name of clock signal  + < clk\_freq>: clock frequency  + <unit>: frequency unit (“Hz”, “KHz”, “MHz” or “GHz”) |
| command | GetCLKfreq <clk\_name> | Get clock frequency value  + <clk\_name>: name of clock signal |
| reg | <register\_name> force <value> | Force register with setting value  + <reg\_name>: name of register  + <value>: value which is set to register |
| reg | <register\_name> release | Release register from force value  + <reg\_name>: name of register |
| reg | <register\_name> <value> | Write a value to register  + <reg\_name>: name of register  + <value>: value which is set to register |
| reg | <register\_name> | Read value from register  + <reg\_name>: name of register |
| reg | - | Dump register names of model |
| tgt | set\_param <term> <value> | Set simulation information about access to target  + <term>: m\_bus\_clk | m\_bus\_gnt | m\_bus\_rgnt | m\_buf\_size| m\_wr\_latency | m\_rd\_latency | m\_phase\_mode  | m\_p\_log\_file | m\_wr\_log | m\_rd\_log | m\_msg\_out\_lvl  | m\_wr\_req\_latency | m\_rd\_req\_latency | m\_fw\_req\_phase  | m\_info\_displayed  + <value> : Please see tlm\_common\_class spec sheet |
| tgt | get\_param <term> | Get simulation information about access to target |
| tgt | init\_param | Initialize simulation information |
| command | help | Dump the direction how to use parameters and commands |
| reg | help |
| tgt | help |

***Notes:***

*- (\*) The setting value MessageLevel is not effected when REGIF\_SC\_REPORT macro is defined.*

*- (\*\*) The message belong to dumping register information is not only effected by setting of MessageLevel parameter but also DumpRegisterRW parameter.*

***How to use***: Below example describes how to use commands/parameters

* Python interface (setting in .py file).

|  |
| --- |
| SCHEAP.setFreq(100,”MHz”)  **#########MessageLevel#########**  SCHEAP.AVSEG\_MessageLevel(“RH850.avseg”) #Get message level  SCHEAP.AVSEG\_MessageLevel(“RH850.avseg”,“info|error|warning|fatal”)#Set message level  SCHEAP.AVSEG\_reg(“RH850.avseg”,“MessageLevel”) #Get message level  SCHEAP.AVSEG\_reg(“RH850.avseg”,“MessageLevel info|error”) #Set message level  SCHEAP.AVSEG\_reg(“RH850.avseg”,“VCCCHSCR MessageLevel”) #Get message level  SCHEAP.AVSEG\_reg(“RH850.avseg”,“VCCCHSCR MessageLevel info|error”) #Set message level  **######### DumpRegisterRW#######**  SCHEAP.AVSEG\_reg(“RH850.avseg”,“DumpRegisterRW”) #Get setting value  SCHEAP.AVSEG\_reg(“RH850.avseg”,“DumpRegisterRW true”) #Set value  **#########AssertReset##########**  SCHEAP.AVSEG\_AssertReset(“RH850.avseg”,“PRESETn 100 20”)  **#########SetCLKfreq###########**  SCHEAP.AVSEG\_SetCLKfreq(“RH850.avseg”,“CLK\_LSB 250 Hz”)  **#########GetCLKfreq###########**  SCHEAP.AVSEG\_GetCLKfreq(“RH850.avseg”,“CLK\_LSB”)  **#########ForceRegister########**  SCHEAP.AVSEG\_reg(“RH850.avseg”,“VCCCHSCR force 0xFF”)  **#########ReleaseRegister######**  SCHEAP.AVSEG\_reg(“RH850.avseg”,“VCCCHSCR release”)  **#########WriteRegister########**  SCHEAP.AVSEG\_reg(“RH850.avseg”,“VCCCHSCR 0xFF”)  **#########ReadRegister#########**  SCHEAP.AVSEG\_reg(“RH850.avseg”,“VCCCHSCR”)  **#########ListRegister#########**  SCHEAP.AVSEG\_reg(“RH850.avseg”,“”)  **#########set\_param############**  SCHEAP.AVSEG\_tgt(“RH850.avseg”,“set\_param m\_wr\_latency=100,SC\_NS”)  **#########get\_param############**  SCHEAP.AVSEG\_tgt(“RH850.avseg”,“get\_param m\_bus\_clk”)  **#########init\_param############**  SCHEAP.AVSEG\_tgt(“RH850.avseg”,“init\_param”)  **#########Help#################**  SCHEAP.AVSEG\_help(“RH850.avseg”) #Model command help message  SCHEAP.AVSEG\_reg(“RH850.avseg”,“help”) #Register I/F help message  SCHEAP.AVSEG\_tgt(“RH850.avseg”,“help”) #Target I/F help message  SCHEAP.sc\_start(1000) |

Figure 6.2: An example of python interface usage

* Command interface (setting in .cmd file).
  + Instance name of model is necessary to put in front of the each keyword.
  + If the target is model, it is necessary to put the keyword "command" in front of each command.
  + If the target is register I/F, it is necessary to put the keyword "reg" in front of each command.
  + And if the target is TLM target I/F, it is necessary to put the keyword "tgt" in front of each command.
  + If a command is required to be broadcast to all targets, the keyword should be empty “”.

|  |
| --- |
| **#Instance keyword command/parameter**  **#########MessageLevel#########**  reslx.avseg MessageLevel #Get general message level  reslx.avseg command MessageLevel #Get message level  reslx.avseg command MessageLevel info|error|warning|fatal #Set message level  reslx.avseg reg MessageLevel #Get message level  reslx.avseg reg MessageLevel fatal|error|warning|info #Set message level  reslx.avseg reg VCCCHSCR MessageLevel #Get message level  reslx.avseg reg VCCCHSCR MessageLevel info|error #Set message level  **######### DumpRegisterRW#######**  reslx.avseg reg DumpRegisterRW #Get setting value  reslx.avseg reg DumpRegisterRW true #Set value  **#########AssertReset##########**  reslx.avseg command AssertReset PRESETn 100 20  **#########SetCLKfreq###########**  reslx.avseg command SetCLKfreq CLK\_LSB 250 Hz  **#########GetCLKfreq###########**  reslx.avseg command GetCLKfreq CLK\_LSB  **#########ForceRegister########**  reslx.avseg reg VCCCHSCR force 0xFF  **#########ReleaseRegister######**  reslx.avseg reg VCCCHSCR release  **#########WriteRegister########**  reslx.avseg reg VCCCHSCR 0xFF  **#########ReadRegister#########**  reslx.avseg reg VCCCHSCR  **#########ListRegister#########**  reslx.avseg reg  **#########set\_param############**  reslx.avseg tgt set\_param m\_wr\_latency=100,SC\_NS  **#########get\_param############**  reslx.avseg tgt get\_param m\_bus\_clk  **#########init\_param############**  reslx.avseg tgt init\_param  **#########Help#################**  reslx.avseg help #General help message  reslx.avseg command help #Model command help message  reslx.avseg reg help #Register I/F help message  reslx.avseg tgt help #Target I/F help message |

Figure 6.3: An example of command interface usage

## Message style

### Help messages

* + - 1. ***Model command help messages***

Table 6.4: Dumping model command help message description

|  |  |
| --- | --- |
| **Condition** | This message is dumped out when model command “help” is called. |
| **Output** | This message's kind is printed to standard output (console). |
| --- command ---  help Show direction  MessageLevel <fatal|error|warning|info> Select debug message level (Default: fatal,error)  AssertReset <rst\_name> <start\_time> <period> Assert and deassert reset signal to a target model  SetCLKfreq <clk\_name> <clk\_freq> <unit> Set clock frequency  GetCLKfreq <clk\_name> Get clock frequency | |

* + - 1. ***Register I/F help messages***

Table 6.5: Dumping register help message description

|  |  |
| --- | --- |
| **Condition** | This message is dumped out when register I/F “help” is called. |
| **Output** | This message's kind is printed to standard output (console). |
| --- reg ---  reg MessageLevel <fatal|error|warning|info> Select debug message level (Default: fatal,error)  reg DumpRegisterRW <true/false> Select dump register access information (Default: false)  reg <register\_name> MessageLevel <fatal|error|warning|info>  Select debug message level for register (Default: fatal,error)  reg <register\_name> force <value> Force register with setting value  reg <register\_name> release Release register from force value  reg <register\_name> <value> Write a value into register  reg <register\_name> Read value of register  reg help Show a direction | |

* + - 1. ***TLM Target I/F help messages***

Table 6.6: Dumping target help message description

|  |  |
| --- | --- |
| **Condition** | This message is dumped out when TLM target I/F “help” is called. |
| **Output** | This message's kind is printed to standard output (console). |
| <model’s instance> has the following target I/F commands.  Command Description  -----------------------------------------------------------------------------------------  set\_param <term> <value> : Set simulation information about access to target.  <term> : m\_bus\_clk | m\_bus\_gnt | m\_bus\_rgnt | m\_buf\_size  | m\_wr\_latency | m\_rd\_latency | m\_phase\_mode  | m\_p\_log\_file | m\_wr\_log | m\_rd\_log | m\_msg\_out\_lvl  | m\_wr\_req\_latency | m\_rd\_req\_latency | m\_fw\_req\_phase  | m\_info\_displayed  <value> : Please see tlm\_common\_class spec sheet.  get\_param <term> : Get simulation information about access to target.  init\_param : Initialize simulation information. | |

### Register RW messages

Table 6.7: Dumping Register RW message description

|  |  |
| --- | --- |
| **Condition** | This message is outputted when register of model is accessed and parameter DumpRegisterRW is set “true”. |
| **Output** | This message's kind is printed to standard output (console). |
| **Format**: Info: <hier\_instance\_name>: [<time>ps] REG [<reg\_name>] R Size = <size> Addr = <reg\_address> Data = <reg\_value>  Info: <hier\_instance\_name>: [<time>ps] REG [<reg\_name>] W Size = <size> Addr = <reg\_address> Data = <reg\_value> : <old\_value> => <new\_value> | |
| **Tag name** | **Description** |
| hier\_instance\_name | Hierarchy instance name of model is being used. |
| time | Simulation time |
| reg\_name | Name of accessed register. |
| size | Access size. |
| reg\_address | Register’s address. |
| reg\_value | Register’s value. |
| old\_value | Register's value before writing. |
| new\_value | Register's value after writing. |

### Error and debugging messages

#### Error and debugging messages style

Table 6.8: Error and debugging message description

|  |  |
| --- | --- |
| **Condition** | This message's kind is output when error occurs or some important events occur. It's depended on setting of parameter MessageLevel.  Detailed conditions are described in the “Description” column of Table 6.9. |
| **Output** | This message's kind is printed to standard output (console). |
| **Format**: <severity>: <hier\_instance\_name>: [<time><unit>] <Message content> | |
| **Tag name** | **Description** |
| severity | Kind of message's severity. |
| hier\_instance\_name | Hierarchy instance name of model is being used. |
| time | Simulation time. |
| unit | Simulation time's unit. |
| Message content | Message content (message list is described in Table 6.9). |

#### List of error and debugging messages

Table 6.9: Error and debug messages

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Level** | **Type** | **Message** | **Description** |
| 1 | error | user | Invalid access address <address> | Users access the model’s register with invalid address. |
| 2 | error | user | Invalid access address <address> with access size <size> bytes | Users access the model’s register with invalid address and size. |
| 3 | error | user | Invalid access size: <size> bytes | Users access the model’s register with invalid size. |
| 4 | error | user | Writing access size to <register\_name> at address <address> is wrong: <size> byte(s). | Users write the value to register with unsupported size. |
| 5 | error | user | Reading access size to <register\_name> at address <address> is wrong: <size> byte(s). | Users read the value in register with unsupported size. |
| 6 | error | user | command name “<input\_command>” is invalid (\*) | Users call command with invalid command. |
| 7 | error | user | <command name> has too much arguments (\*) | Users call command with number of argument is wrong. |
| 8 | error | user | <command name> command needs an argument [true/false] (\*) | Users call command with invalid arguments. |
| 9 | error | user | Register name <register\_name> is invalid (\*) | Users call command of register I/F with register name is wrong. The register names are list in Table 4.1. |
| 10 | error | user | Invalid force value (\*) | Users call force command of register I/F with invalid arguments. |
| 11 | error | user | Invalid write value (\*) | Users call release command of register I/F with invalid arguments. |
| 12 | error | user | Wrong command : (<input\_command> (\*) | Users call command of register I/F which is unsupported. |
| 13 | error | user | wrong argument: <argument (s)> (<input\_command> ) : Type <model\_name> help (\*) | Users call command with invalid arguments. |
| 14 | error | user | wrong argument (<input\_command> ) : Type <model\_name> help (\*) |
| 15 | error | user | wrong number of arguments ( <input\_command> ) : Type <model\_name> help (\*) | Users call command with invalid number arguments. |
| 16 | - | user | Wrong number of argument for <command name> command. | Users call command (via python I/F) with invalid number arguments. |
| 17 | warning | user | Clock name (<clock\_name>) is invalid. | Users call SetCLKfreq or GetCLKfreq command with clock name is wrong. |
| 18 | warning | user | The name of reset signal is wrong. It should be <reset\_name(s)>. | Users call AssertReset command with reset name is wrong. |
| 19 | warning | user | <register\_name> is blocked writing from Bus I/F. | Users try to write to forced register. |
| 20 | warning | user | Cannot launch call-back function during reset period | Users read the register during reset period. |
| 21 | warning | user | Cannot write 1 to reserved bit | Users write value to reserved bit of register. |
| 22 | warning | user | Cannot write during reset period | Users write value to register during reset operation. |
| 23 | warning | user | Should read all bit in a register | Users read the register with read access size if smaller than register size. |
| 24 | warning | user | Frequency unit (<unit>) is wrong; frequency unit (Hz) is set as default. | Users call SetCLKfreq with frequency unit is wrong. The frequency unit must be Hz, KHz, MHz, GHz. |
| 25 | warning | user | The AssertReset command of <reset\_name> is called in the reset operation (<reset\_name>) of the model. So this calling is ignored. | Users call AssertReset command during reset operation. |
| 26 | warning | user | Setting prohibited: Cannot write 0 to <bit\_name> bit. | Users write a prohibited value to register. |
| 27 | warning | user | The input of both upper and lower pulses into the same module at the same time is prohibited. | Model receives input of both upper and lower pulses into the same module at the same time. |
| 28 | info | user | There are <number> same input channel selectors (CHS bit = <channel\_number>). | Users select same input virtual channel for more than 1 power supply. |
| 29 | info | user | <clock\_name> frequency is <frequency> <unit>. | Users call GetCLKFreq with valid clock name. |
| 30 | info | user | The model will be reset (<reset\_name>) for <period\_time> ns after <delay\_time> ns. | Users call AssertReset command with start reset time and reset period. |
| 31 | info | internal | The model is reset by AssertReset command of <reset\_name>. | Reset by AssertReset command is active. |
| 32 | info | user | The reset signal of <reset\_name> is asserted. | Users activate reset signal. |
| 33 | info | user | The reset signal of <reset\_name> is negated. | Users deactivate reset signal. |
| 34 | info | internal | Reset period of <reset\_name> is over. | Reset period of reset which is set by AssertReset is over. |
| 35 | info | internal | Initialize <register\_name> (<value>) | Registers are initialized by reset signal(s) |
| 36 | info | user | <clock\_name> frequency is zero. | Users do an action when clock frequency is zero value. |
| 37 | info | user | There is no input channel is selected for generating error of <power\_name>. | Users set reserved value to CHS bit. |
| 38 | info | internal | ADCH0 virtual channel <channel\_number> is selected for generating error of <power\_name>. | Users set valid value to CHS bit. |

***Note:***

*- (\*) The dumping message is not depend on setting value of MessageLevel parameter. This message is returned to Command Handler (commandHandler.h)/Python Handler (PY\_AVSEG.cpp). Command/Python Handler will decide the message content will be dumped.*

## Defined macro and template

* There is no template in model.
* There are three macros IS\_RESET\_ACTIVE\_LOW, REGIF\_SC\_REPORT, and REGIF\_NOT\_USE\_SYSTEMC in the model
* When users define the macro IS\_RESET\_ACTIVE\_LOW, reset active Low level. If not, reset actives High level.
* If users define the macro REGIF\_SC\_REPORT, the SC\_REPORT function is used. Otherwise, the “printf” function is used. This macro should be not defined if users defined REGIF\_NOT\_USE\_SYSTEMC.
* Users can define macro REGIF\_NOT\_USE\_SYSTEMC to remove SystemC part.

# Flow diagram

## Sequence flow

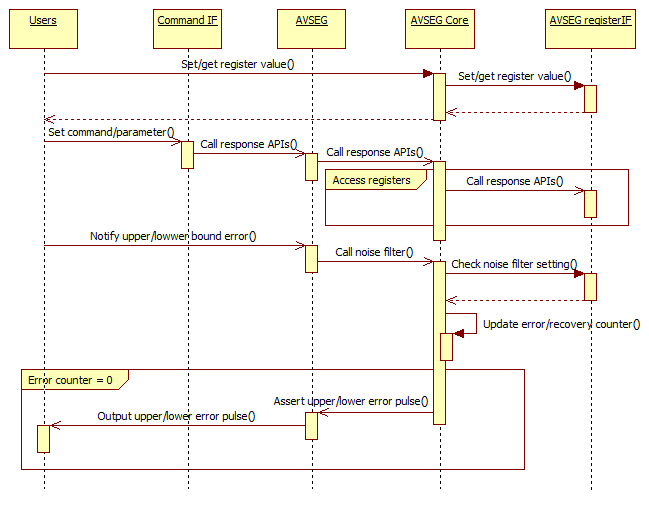


Figure 7.1: Sequence flow

***Explanation***:

* Users interact with parameters/commands of the model via Command I/F. (The commands and parameters are described in Table 6.2 and Table 6.3).
* Users access the registers of model via the TLM target socket m\_tgt\_sockets[0].
* When model receives an upper/lower bound error, the noise filter is called. If this error match with the noise filter setting, the error counter and recovery counter are updated (refer to Chapter 7.5 for more detail).
* When the error counter reaches value 0, the error pulse is outputted.

## State diagram

(7)

(3)

(4)

**RESET**

**ENABLE FILTER**

**DISABLE FILTER**

(1)

(2)

(5)

**OUTPUT ERROR PULSE**

(6)

(8)

Figure 7.2: State diagram

***Explanation***:

* RESET: Transition of RESET state to DISABLE FILTER state is described as below:
* RESET → DISABLE FILTER: The state of the model is changed from RESET state to DISABLE FILTER state after reset is negated. (1)
* DISABLE FILTER: Transition of this state to other states is described as below:
  + DISABLE FILTER → RESET: If the reset is asserted (by hardware reset signal or user reset command), the state of the model is changed to RESET state. (2)
  + DISABLE FILTER → ENABLE FILTER: If users write 1 to ENB bit in Filter Counter Control register (VCCCNTCR/ EVCCCNTCR/ VDDCNTCR), the noise filter of correlative power supply (VCC, EVCC, VDD) is enabled. (3)
* ENABLE FILTER: Transition of this state to other states is described as below:
* ENABLE FILTER → DISABLE FILTER: When users write 0 to ENB bit in Filter Counter Control register (VCCCNTCR/ EVCCCNTCR/ VDDCNTCR), the AVSEG for correlative power supply functions are disabled. (4)
* ENABLE FILTER → OUTPUT ERROR PULSE: When the error counter reaches value 0, the upper/lower error pulse can be outputted. (5)
* ENABLE FILTER → RESET: If the reset is asserted (by hardware reset signal or python command), the state of the model is changed to RESET state. (7)
* OUTPUT ERROR PULSE: Transition of this state to other states is described as below:
* OUTPUT ERROR PULSE → ENABLE FILTER: When recovery counter reaches value 0 and model receives a signal within boundaries, the state of the model is changed from OUTPUT ERROR PULSE state to ENABLE FILTER state. (6)
* OUTPUT ERROR PULSE → RESET: If the reset is asserted (by hardware reset signal or python command), the state of the model is changed to RESET state. (8)

## Reset flow

Initialize variables and output signals

Cancel all operation events

Cancel events related to AssertReset

Initialize all registers

[PRESETn active]

[else]

Figure 7.3: Reset flow

***Explanation***:

* The AVSEG model has 1 hardware reset and 1 corresponding software reset of PRESETn, users can reset model by software reset via Command IF or by reset signal via PRESETn port. The Figure 7.4 shows the relationship between software reset and hardware reset.
* If the PRESETn is active, the model operates as following:
* Cancel all operation events and events related to the AssertReset.
* Initialize all registers.
* Initialize all internal variables and output signals.

***Note****:*

*- All registers cannot be accessed (read/write) during reset PRESETn period.*

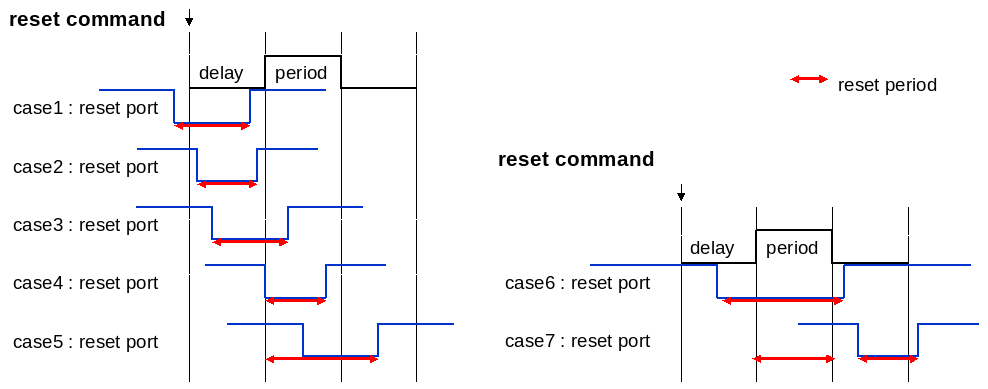


Figure 7.4: Relationship between software reset and hardware reset

## Command/parameter configuration operation flow

* Users set/call the parameters/commands of model via the Command I/F. Setting/calling operation is described as Figure 7.5 -> Figure 7.8.
* The function of the parameters and commands is described in Table 6.2 and Table 6.3.

MessageLevel

parameter?

Users set parameters or commands via Command IF

Select debug message level

DumpRegisterRW

parameter?

Dump register access info

when register is accessed

Yes

Yes

No

No

Figure 7.5: Command/parameter configuration operation flow (1/4)

ForceRegister

command?

Force register with setting value

Is valid register name?

ReleaseRegister

command?

Release register with forced value

Is valid register name?

ListRegister

command?

Dump register name list

Yes

Yes

Yes

No

No

No

No

No

Yes

Yes

Figure 7.6: Command/parameter configuration operation flow (2/4)

No

Help

command?

No

No

WriteRegister

command?

Set value to register

Is valid register name?

Set value to register

Yes

Yes

Yes

No

Reset operation

Is valid reset name?

Yes

Yes

Get value from register

Yes

Yes

ReadRegister

command?

Is valid register name?

No

AssertRreset

command?

No

No

Figure 7.7: Command/parameter configuration operation flow (3/4)

SetCLKFreq

command?

Set value to clocks

Is valid clock name?

GetCLKFreq

command?

Get frequency value of clocks

Is valid clock name?

Yes

Yes

Yes

Yes

No

No

No

No

Figure 7.8: Command/parameter configuration operation flow (4/4)

## Noise filter and upper/lower error pulse output flow

Error counter counts down (2\*)

[else]

[Recovery counter <= 1 || Error counter > 0]

[Recovery counter > 0]

[Lower bound error]

[Error status = lower error]

[selected channel = chan\_num]

[vcend[chan\_num] asserts]

[Noise filter enable]

[Upper bound error ||

Lower bound error]

[Error counter > 0]

[Error counter == 0]

[Upper bound error]

[Error status = upper error]

Wait next raising edge of CLK\_LSB

Recovery counter = recovery count number (1\*)

Output upper error pulse (1 CLK\_LSB)

Error status = upper error

Output lower error pulse (1 CLK\_LSB)

Error status = lower error

Error counter = error count number (1\*)

Error status = no error

Recovery counter counts down (2\*)

[else]

[else]

[else]

[else]

[else]

[else]

[else]

[else]

[else]

[else]

Figure 7.9: Noise filter and upper/lower error pulse output flow

***Explanation***:

* There are 2 counters for each correlative power supply (1 error counter, 1 recovery counter) used to control noise filter and output error pulse.
* When model receives an AD conversion end signal (vcend[chan\_num]) from an ADCH0 virtual channel “chan\_num”, model will confirm whether there is any noise filter enabled for this channel “chan\_num” or not by checking setting of CHS bit in Input Channel Selector registers (VCCCHSCR, EVCCCHSCR and VDDCHSCR) and setting of ENB bit in Filter Counter Control registers (VCCCNTCR, EVCCCNTCR and VDDCNTCR).
* When noise filter is enabled for virtual channel “chan\_num”:
  + If upper/lower bound error is noticed, the recovery counter is recovered to value set in NRMCNT bit of Filter Counter Control register at next raising edge of CLK\_LSB clock. Besides, if error counter > 0, the error counter counts down (discount 1). At this time, when error counter reaches value 0, error status is updated to upper/lower error.
  + Otherwise, model receive the signal within boundaries, if recovery counter <= 1 or error counter > 0, the error counter is recovered the value set in ERRCNT bit of Filter Counter Control register, and error status is set to “no error”. Besides, if recovery counter > 0, the recovery counter counts down (discount 1).
* After noise is filtered, upper/lower error pulse is outputted via secondary HDET/LDET output ports (evccshdet/evccsldet, vccshdet/vccsldet, vddshdet/vddsldet) if error status is uppler/lower error.

***Notes***:

*- “chan\_num” = 0 -> 39.*

*- There are 3 noise filters for correlative power supply (VCC, EVCC and VDD)*

*- (1\*) “error counter number” is set in ERRCNT bit of Filter Counter Control register and “recovery counter number” is set in NRMCNT bit of Filter Counter Control register*

*- (2\*) The error/recovery counter keeps value 0 when its value already reached 0 (minimum value)*

# Functions description

The detail specification of implemented classes is described in ref[8].

Table 8.1: Functions of Cavseg class

| **No.** | **Method** | **Level** | **Description** |
| --- | --- | --- | --- |
| 1 | Cavseg (sc\_module\_name name) | Public | Constructor of Cavseg class. |
| 2 | ~ Cavseg (void) | Destructor of Cavseg class. |
| 3 | void AssertReset (const std::string reset\_name, const double start\_time, const double period) | Reset the model. This method is called by Command IF. |
| 4 | void SetCLKfreq (const std::string clock\_name, const sc\_dt::uint64 freq, const std::string unit) | Set clock frequency to the model. This method is called by Command IF. |
| 5 | void GetCLKfreq (const std::string clock\_name) | Dump information for clock frequency from the model. This method is called by Command IF. |
| 6 | std::string RegisterHandleCommand(const std::vector<std::string>& args) | Set parameters or commands related to register I/F to AVSEG\_Core. This method is called by Command IF. |
| 7 | void NotifyErrPulseOut (const unsigned int chn\_sel, const unsigned int ex\_bound) | Notify Upper/Lower error pulse output. |
| 8 | unsigned int tgt\_acc\_dbg(unsigned int id, tlm::tlm\_generic\_payload &trans) | Private | Access memory in debug mode via TLM target socket. This is virtual function of tlm\_tgt\_if. |
| 9 | void tgt\_acc(unsigned int id, tlm::tlm\_generic\_payload &trans, sc\_time &t) | Access memory in normal mode via TLM target socket. This is virtual function of tlm\_tgt\_if. |
| 10 | void Initialize (void) | Initializes internal variables & output ports. |
| 11 | void CancelEvents (void) | Cancel all operation events. |
| 12 | void ResetOutputPort (void) | Reset output ports when reset is active. |
| 13 | void EnableReset (const bool is\_active, const unsigned int reset\_id) | Process reset function. |
| 14 | void HandleCLK\_LSBSignalMethod (void) | Handle the change of CLK\_LSB clock signal. |
| 15 | void HandlePRESETnSignalMethod (void) | Handle the change of PRESETn signal. |
| 16 | void HandleResetHardMethod (const unsigned int reset\_id) | Process reset function when reset port is changed. |
| 17 | void HandleResetCmdMethod (const unsigned int reset\_id) | Process reset function when reset command is called. |
| 18 | void CancelResetCmdMethod (const unsigned int reset\_id) | Cancel reset function when reset command is processed completely. |
| 19 | void HandleADEndInputMethod (const unsigned int chan\_num) | Handle the change in vcend[chan\_num] input port (chan\_num = 0 -> 39). |
| 20 | void NotifyFilterControlMethod (const unsigned int chn\_sel) | Notify to Filter Control after receive AD end pulse. |
| 21 | void NotifyErrPulseDoneMethod (const unsigned int chn\_sel) | Notify to error pulse output is done. |
| 22 | void WriteErrPulseMethod (const unsigned int chn\_sel) | Write error pulse output. |
| 23 | bool CheckClockPeriod (const std::string clock\_name) | Check clock period. |
| 24 | double GetCurTime(void) | Get current simulation time. |
| 25 | double CalculateClockEdge (const double clock\_period, const bool is\_pos, const bool add\_period, const double time\_point, const double time\_change\_point) | Calculate clock synchronous time. |
| 26 | void ConvertClockFreq (sc\_dt::uint64 &freq\_out, std::string &unit\_out, sc\_dt::uint64 freq\_in, const std::string unit\_in) | Convert frequency value and frequency unit. |
| 27 | void GetTimeResolution (sc\_dt::uint64 &resolution\_value, sc\_time\_unit &resolution\_unit) | Get simulation time resolution. |
| 28 | void SetLatency\_TLM (const bool is\_constructor) | Set read/write latency and clock for TLM |

Table 8.2: Functions of Cavseg\_core class

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Method** | **Level** | **Description** |
| 1 | Cavseg\_core(std::string name, Cavseg \*AVSEGPtr) | Public | Constructor of Cavseg\_core class. |
| 2 | ~Cavseg\_core (void) | Destructor of Cavseg\_core class. |
| 3 | bool read (const bool is\_rd\_dbg, unsigned int addr, unsigned char \*p\_data, unsigned int size) | Access to read value of register. |
| 4 | bool write (const bool is\_wr\_dbg, unsigned int addr, unsigned char \*p\_data, unsigned int size) | Access to write value to register. |
| 5 | bool GetWriteCondition (const unsigned int addr) | Private | Check writing condition to register. |
| 6 | void PreReadCall (const unsigned int addr) | Handle register before reading. |
|  | void EnableReset (const bool is\_active, const unsigned int reset\_id) | Reset registers. |
| 7 | void Initialize (void) | Initialize internal variables. |
| 8 | std::string RegisterHandler (const std::vector<std::string> &args) | Set parameters or commands to Cavseg\_regif class. |
| 9 | unsigned int GetChannelSelect (const unsigned int chn\_sel) | Get selected input channel. |
| 10 | void NotifyFilterControl(const unsigned int chn\_sel, const bool is\_upper, const bool is\_lower) | Notify Noise Filter. |
| 11 | void cb\_VCCCHSCR\_CHS(RegCBstr str) | Callback functions when register is written. |
| 12 | void cb\_VCCCNTCR\_ENB(RegCBstr str) |
| 13 | void cb\_EVCCCHSCR\_CHS(RegCBstr str) |
| 14 | void cb\_EVCCCNTCR\_ENB(RegCBstr str) |
| 15 | void cb\_VDDCHSCR\_CHS(RegCBstr str) |
| 16 | void cb\_VDDCNTCR\_ENB(RegCBstr str) |
| 17 | bool strmatch(const char \*ptn, const char \*str) | Compare 2 strings. |

# Limitation

* None

**Revision History**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Version** | **Modified points** | **Agreement** | **Approver** | **Checker** | **Author** |
| 1.0 | - Created new | - | Yen Nguyen  08/03/2016 | Thanh Phan  08/03/2016 | Ngan Tran  07/27/2016 |
| 1.1 | - Updated document number “D-SLD-M40-0053-01” -> “D-SLD-M40-0053-02”  - Added reference document No.8  - Added 1 more note in Chapter 2 “All features described in HWM (ref[7]/Chapter 34.9) are supported in model”  - Corrected block name “Python IF” into “Command IF” in Chapters 3, 7.1, 7.4, 8  - Added 1 more note in Chapter 4 “All registers described in HWM (ref[7]/Chapter 34.9.2) are supported in model”  - Added TLM target sockets in Table 5.1  - Added referent about macro to set active level for reset in Table 5.1, Chapters 5.3, 6.6  - Modified description in Chapter 5.4 (e.g: add “at the next raising edge of CLK\_LSB” to make clear the timing of operation; …)  - Updated CVS tag for TLM common class, re\_register in Table 6.1, and change version of Register IF Generator into v2014\_12\_01  - Updated Table 6.1 and Figure 6.1 to include re\_define.h  - Updated Chapter 6.4 to modify command/parameter description and using direction  - Updated Chapter 6.5  + Swap order of chapter 6.5.1 and chapter 6.5.2  + Modify more clearly about “help” command  + Update Table 6.9 (e.g: add model’s messages; update No.16 to remove message level due to PY\_<model\_name>.cpp dumps this message by printf without message level)  - Updated Chapter 7.5 to support 2 counters instead of 4 counters for each power supply  - Updated Chapter 8 according final source code |  | Yen Nguyen  09/08/2016 | Thanh Phan  09/08/2016 | Ngan Tran  08/05/2016 |