Register IF Generator
User Manual

Renesas Design Vietnam Co., Ltd.
D-SLD-CMN-0080-01
USR-SLD-14011 Rev.1.2

OUTLINE

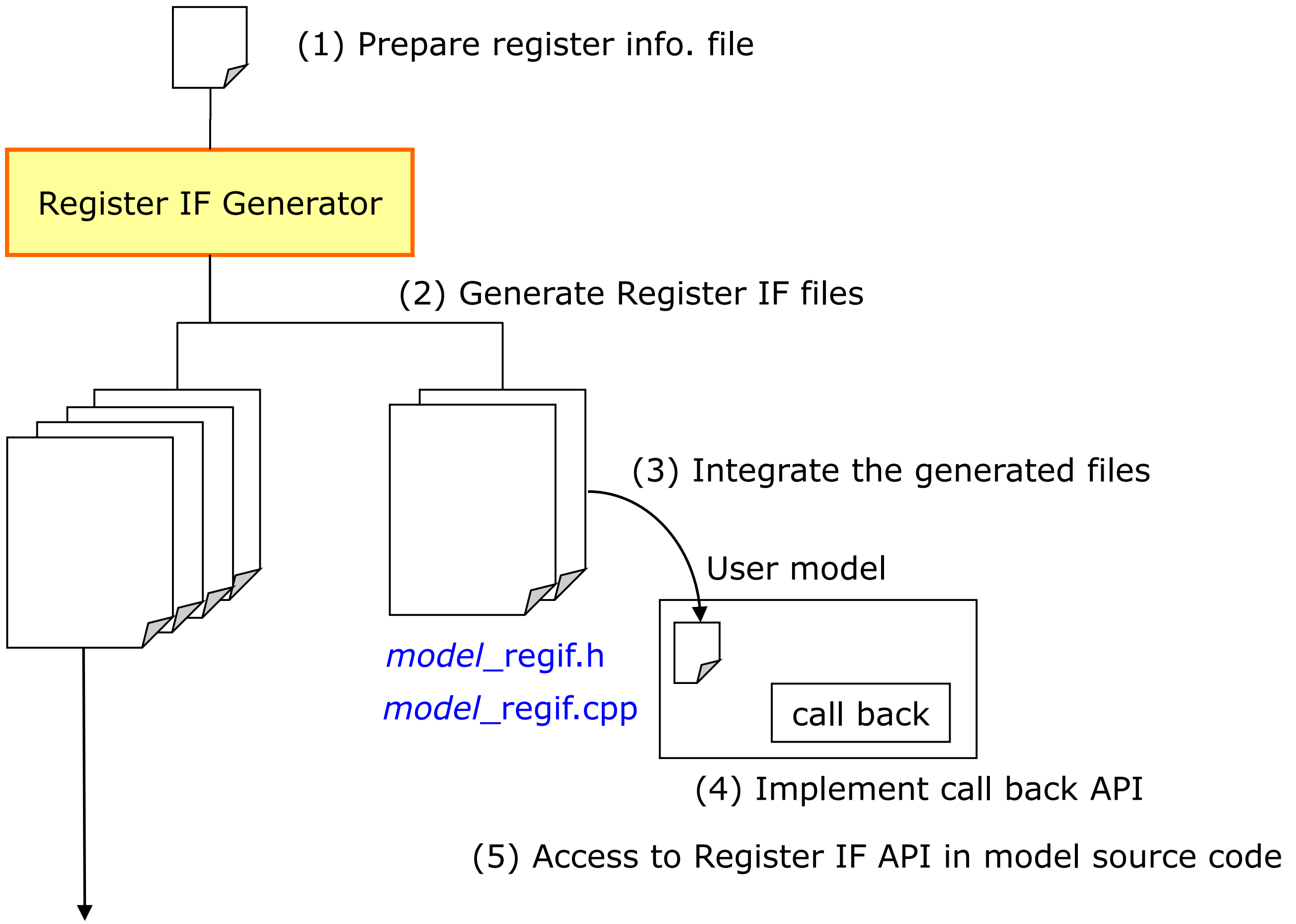
- Overview
  - Features
  - Design Flow
  - Block Diagram
- Usage & Features
  - Usage
  - Basic Features
  - Advance Features
  - Other Features
- Example of Usage

FEATURES

Register IF is one of model IF whose purposes are to declare model's register and control register access (read/write) operation. Register IF generator makes that C++ code from users defined register description file, hereafter “register info. file”.

- Generate C++ code
  - Generate register variables and functions for register access
  - Support register declaration for Synopsys Virtualizer
  - Support to access in APB mode
  - Support to access with flexible size
  - C++ class in two files or one file
- Register Info. File
  - Support two formats: flat and hierarchical
  - Enable to define multiple registers by factor index
  - Enable to specify writable value list
- Generate VLAB meta-data file
- Generate test pattern

DESIGN FLOW



(1) Prepare register info. file

Register IF Generator

(2) Generate Register IF files

*model*_regif.h

*model*_regif.cpp

(3) Integrate the generated files

User model

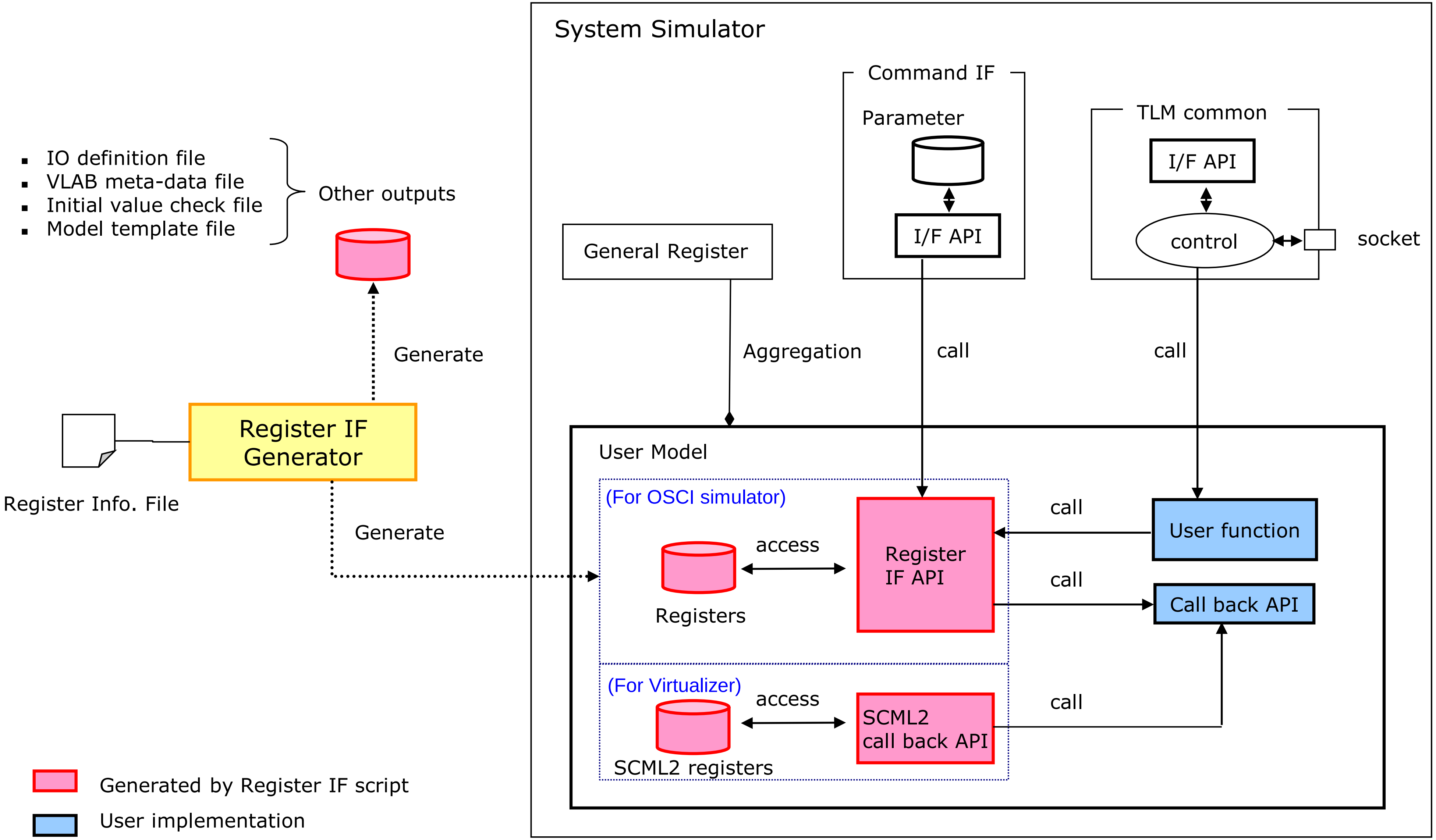
call back

(4) Implement call back API

(5) Access to Register IF API in model source code

iodefine_*model*.h: structure of all registers file, to declare the structure of all registers in the model
*model*_initregchk.c: test pattern file, to check the initial value of all registers in the model
model.h: model template file
*Cmodel*_metadata.py: VLAB metadata file, the register declaration file for simulation in ASTC's VLAB tool (optional)

BLOCK DIAGRAM



- IO definition file
- VLAB meta-data file
- Initial value check file
- Model template file

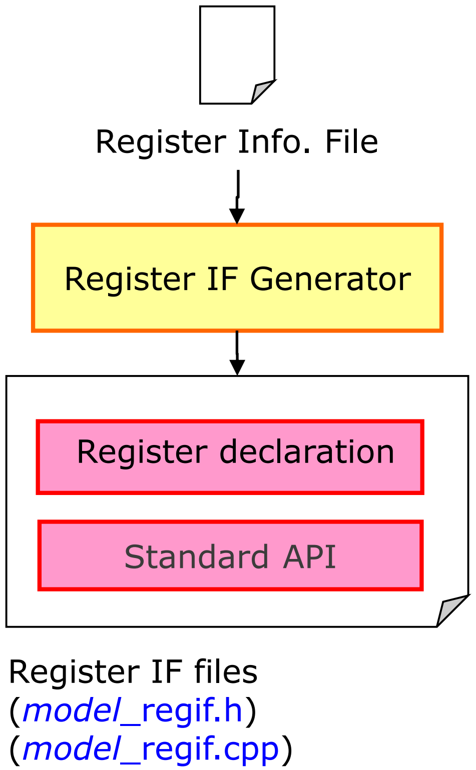
🟥 Generated by Register IF script

🟦 User implementation

OUTLINE

- Overview
  - Features
  - Design Flow
  - Block Diagram
- Usage & Features
  - Usage
  - Basic Features
  - Advance Features
  - Other Features
- Example of Usage

USAGE OF REGISTER IF GENERATOR



Register Info. File

Register IF Generator

Register declaration

Standard API

Register IF files
(*model*_regif.h)
(*model*_regif.cpp)

```
>> python3 gen_regif.py [option] <Register Info. File>
  [option]: [-h/--help] [--version] [-o/--onefile] [-m/--module]
            [-a/--metadata] [-i/--hier] [-k/--keyword]
```

- Register IF Generator works with Python version 3.
- Generate Register IF file from Register Info. file which includes at least registers structure description.
- Generated files name are composed of model-name + regif.h/regif.cpp
- Concatenate two output files (*model*_regif.h and *model*_regif.cpp) into one file when -o/--onefile is specified.
- Indicate each module name clearly when two or more module names are defined in Register Info. File by -m/--module option.
- Enable generating VLAB metadata file by -a/--metadata option.
- Indicate that input file contains register information in a hierarchy when -i/--hier is specified.
- Dump another desired file name by -k/--keyword option.
- Dump a script usage by -h/--help option and dump a generator version by --version option.

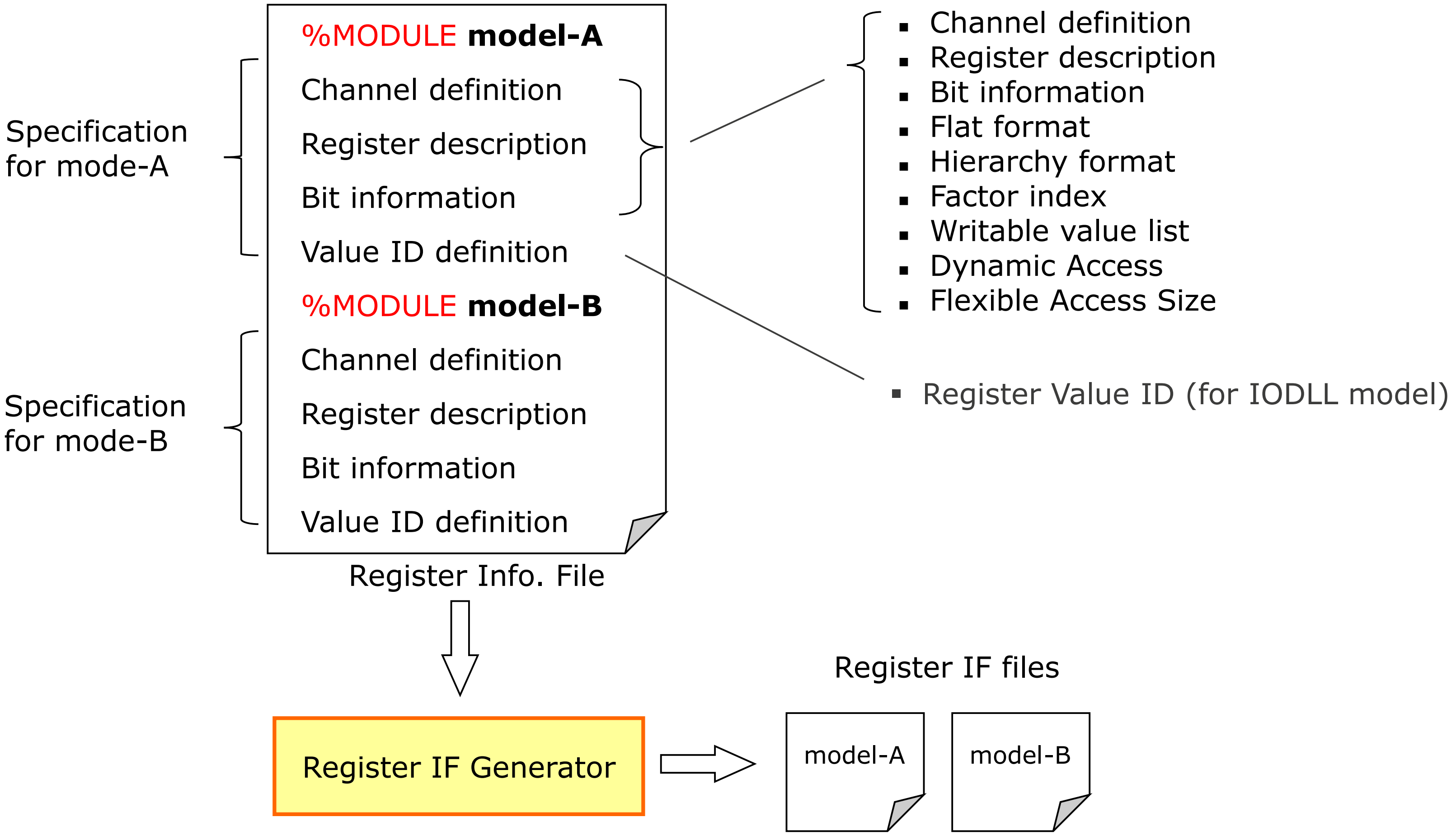
OUTLINE

- Overview
  - Features
  - Design Flow
  - Block Diagram
- Usage & Features
  - Usage
  - Basic Features
    - **Channel definition**
    - **Register description**
    - **Bit information**
    - **Flat format**
    - **Register IF handleCommand**
  - Advance Features
  - Other Features
- Example of Usage

FORMULA OF REGISTER INFO. FILE (1/9)

- The only input file of Register IF Generator is the Register Info. File. This is a text-format file which contains register information of a model.

- Model name is specified after %MODULE attribute.



Specification for mode-A

%MODULE **model-A**

Channel definition

Register description

Bit information

Value ID definition

Specification for mode-B

%MODULE **model-B**

Channel definition

Register description

Bit information

Value ID definition

Register Info. File

- Channel definition
- Register description
- Bit information
- Flat format
- Hierarchy format
- Factor index
- Writable value list
- Dynamic Access
- Flexible Access Size

- Register Value ID (for IODLL model)

Register IF Generator

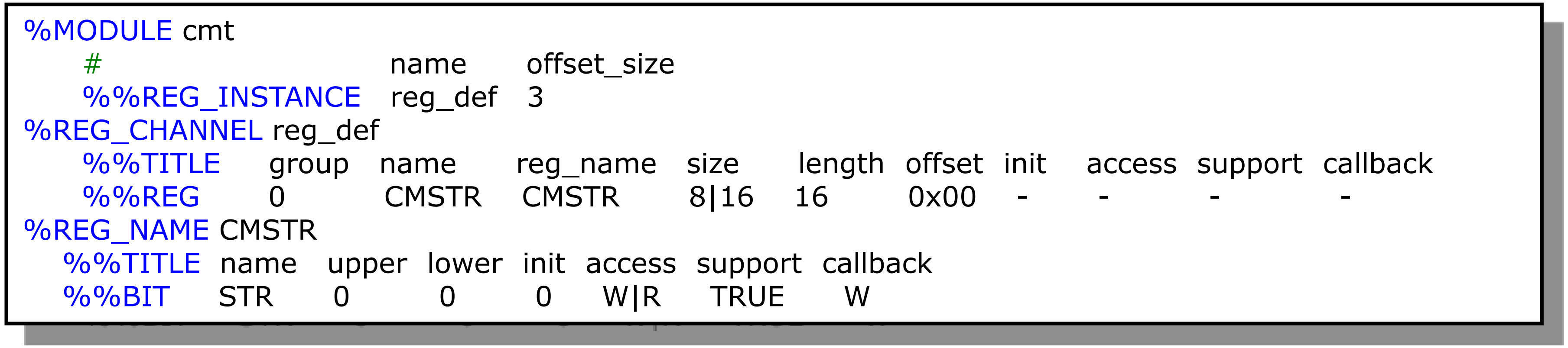
Register IF files

model-A

model-B

FORMULA OF REGISTER INFO. FILE (2/9)

Channel definition, register description & bit information



```
%MODULE cmt
      #                          name      offset_size
      %%REG_INSTANCE   reg_def   3
%REG_CHANNEL reg_def
      %%TITLE     group    name      reg_name   size      length  offset  init    access  support  callback
      %%REG       0        CMSTR     CMSTR      8|16      16      0x00    -       -       -        -
%REG_NAME CMSTR
    %%TITLE  name   upper  lower  init  access  support  callback
    %%BIT    STR    0      0      0     W|R     TRUE     W
```

- Summary
  - Define a model, channel, register description & bit information
- Description rule
  - A comment line begins with "#".
  - A description line begins with a control keyword. Control keyword starts with a number of "%". There are 2 levels of keywords: % and %%.
  - After the keyword, description lines contain arguments which describe register/bit setting. Determined by %%TITLE keyword, the order of arguments may change flexibly.
  - Arguments in a description line are separated by space(s) or tab(s).
  - Detailed specification is explained in the next slides

FORMULA OF REGISTER INFO. FILE (3/9)

Channel definition, register description & bit information

```
%MODULE cmt
    #                    name     offset_size
    %%REG_INSTANCE  reg_def    3
%REG_CHANNEL reg_def
    %%TITLE    group   name      reg_name   size     length  offset  init    access  support  callback
    %%REG      0       CMSTR     CMSTR      8|16     16      0x00    -       -       -        -
%REG_NAME CMSTR
   %%TITLE  name   upper  lower  init  access  support  callback
   %%BIT    STR    0      0      0     W|R     TRUE     W
```

- Generated code
  - Define model's register variables and functions
  - Declare call-back function

FORMULA OF REGISTER INFO. FILE (4/9)

Register description keywords

Keyword	Meaning	Explanation
%MODULE	Model definition	Define the model name(*1), contains: %%REG_INSTANCE for model registers declaration
%%REG_INSTANCE	Channel instance	Declare a channel of register: Require a corresponding %REG_CHANNEL afterward for detailed channel definition. - Flat type: only one %%REG_INSTANCE under %MODULE - Hierarchical type: one under %MODULE, and possibly more under %REG_CHANNEL(for multi-level register channels)
%REG_CHANNEL	Channel definition	Define register in a channel. Each %%REG_INSTANCE declaration requires a corresponding %REG_CHANNEL afterward for detailed channel definition. May contain: %%REG(for a register), or %%REG_INSTANCE(for a sub-channel of register)

(*1) Model name: the model name is specified by %MODULE keyword will be used in the Register IF class and file name. The script will add “C” prefix to model name to form class name (i.e, Cmodel, Cmodel_regif).

FORMULA OF REGISTER INFO. FILE (5/9)

Register description keywords (cont'd)

Keyword	Meaning	Explanation
%%REG	Register instance	Declare a register, under a %REG_CHANNEL Bit definition: - If only one bit name for the whole register: may define bit detail directly in %%REG - If multiple bit name in a register: may define in a corresponding %REG_NAME afterward.
%REG_NAME	Register definition	Define bits in a register, contains: %%BIT for bit definition
%%BIT	Bit definition	Define a bit under %REG_NAME
%%TITLE	Item definition	Arguments of this keyword determine the order of argument of succeeding %% in a same % *Example:* %REG_NAME CMSTR **%%TITLE name upper lower init access** %%BIT STR 0 0 0 W|R

FORMULA OF REGISTER INFO. FILE (6/9)

Channel declaration (%%REG_INSTANCE) argument

No	Argument	Must/optional	Default	Explanation	Enable value
1	name	must	-	Register channel name	String
2	offset_size	optional	16	Address offset bit width	1-32
3	offset_start	optional	0	Initial channel's address offset if there are some channels	Number
4	offset_times	optional	1	Number of channels	Number
5	offset_skip	optional	0x10000	Address size toward next channel if there are some channels	Number

These arguments (offset_start, offset_times, offset_skip) are used for hierarchical type

FORMULA OF REGISTER INFO. FILE (7/9)

Register definition (%%REG) argument

No	Argument	Must/optional	Default	Explanation	Enable value
1	name	must	-	Register name	String
2	reg_name	optional	“”	Register instance name (support flat type only)	String
3	offset	must	-	Address offset	Number
4	size	optional	32	Enable access size	8|16|32
5	wsize(*2)	optional	32	The accessible written size	8|16|32
6	rsize(*2)	optional	32	The accessible read size	8|16|32
7	length	optional	32	Register size	8|16|32
8	group	optional	“”	Group name	String

(*2) Dependent write/read access size: The accessible size will be supported to be separated or not based on users setting. Users can define two more fields to set to accessible size for each registerin writing and read operation. If the argument is not defined or defined with default value, the accessible size of writing and reading operation are same as the defined value of “size”.

FORMULA OF REGISTER INFO. FILE (8/9)

Register definition (%%REG) argument (cont'd)

No	Argument	Must/optional	Default	Explanation	Enable value
9	callback	optional	-	Need or not call back method	R or W or RW
10	init	optional	0x0	Initial value	Number
11	access	optional	W|R	Access limitation	String
12	support	optional	true	Implemented feature or not	true/false
13	factor_start(*3)	optional	""	Start number for multiple registers with continuous index	Number
14	factor_end(*3)	optional	""	End number for multiple registers with continuous index	Number
15	factor_index(*3)	optional	""	Index number for multiple registers with concrete index	Number
16	factor_step(*3)	optional	""	Step number of address for multiple registers	Number

(*3) factor_start, factor_end, factor_index, factor_step are used to define multiple register.
Refer slide "Factor index" for details.

callback, init, access and support: are only valid in case bit information (%%BIT) is not defined for that register

FORMULA OF REGISTER INFO. FILE (9/E)

Bit definition (%%BIT) argument

No	Argument	Must/optional	Default	Explanation	Enable value
1	name	must	-	Bit name	String
2	upper	must	-	Start point	0-31
3	lower	must	-	End point	0-31
4	init	optional	0	Initial value	Number
5	access(*4)	optional	W|R	Access limitation	String
6	support	optional	true	Implemented feature or not	true/false
7	callback(*5)	optional	-	Need or not call back method	R or W or RW
8	value(*6)	optional	-	Writable data	String

(*4) Access limitation: describe register access mode.
The format for this description is shown in "Access limitation" slide.

(*5) Call-back function: is the method that is called when a bit/register is accessed.
Refer to slide "Call back function" for detail.

(*6) Writable data list all data values which are allowed to write to register.
Refer to slide "Writable value list" for detail.

INPUT FORMAT

There are 2 types of input file:

- **Flat type**: description of all registers is listed independently one after another in the same level, without any hierarchical grouping.
- **Hierarchical type**: description of registers of the same unit/channel is grouped, and those groups of register description are listed hierarchically.

  Refer to “Hierarchical type” slide in Advanced Usage part.

FLAT FORMAT (1/3)

- Register list description

Register name	Symbol	Reset value	Offset Address	Access Size(bit)	Register length
CMT Start Register	CMSTR	0000h	0h	8|16	16
CMT Control Register	CMCR	0000h	2h	8|16	16
CMT Counter	CMCNT	0000h	4h	16	16
CMT Constant Register	CMCOR	FFFFh	6h	16	16

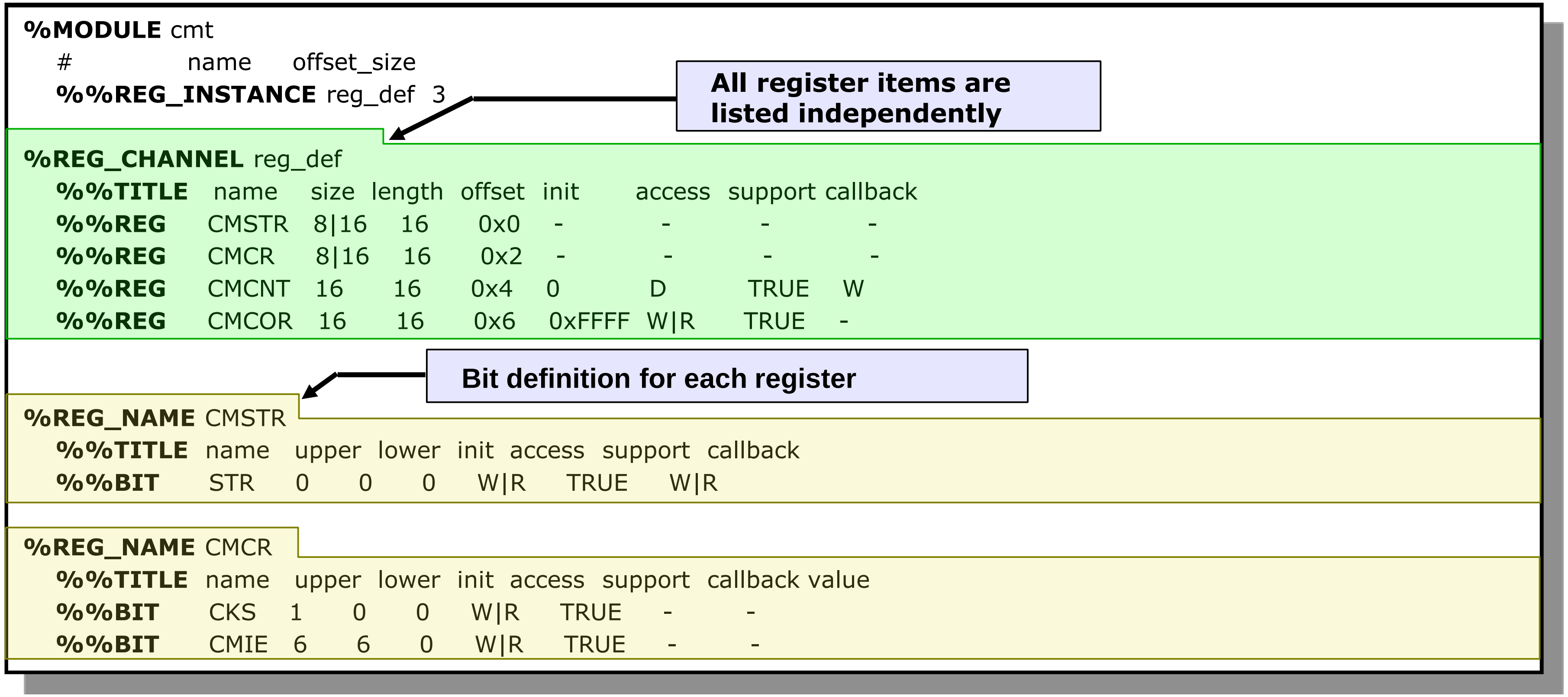
FLAT FORMAT (2/3)

- Bit description

Register	Bit	Bit symbol	Bit Name	Description	R|W
CMSTR	[15:1]	-	Reserved	These bits are always read as 0. The write value should always be 0.	R|W
	[0]	STR	Count Start	0: Counter is stopped 1: Counter is started	R|W
CMCR	[15:7] and [5:2]	-	Reserved	These bits are always read as 0. The write value should always be 0.	R|W
	[6]	CMIE	Compare Match Interrupt Enable	0: Compare match interrupt (CMI) disabled 1: Compare match interrupt (CMI) enabled	R|W
	[1:0]	CKS[1-0]	Clock Select	00:PCLK/8; 01: PCLK/32; 10: PCLK/128; 11: PCLK/512	R|W
CMCNT	[15:0]	CMCNT	Compare match counter	CMCNT is a readable/writable up-counter to generate interrupt request.	R|W
CMCOR	[15:0]	CMCOR	Compare match constant	CMCOR sets the interval up to a compare match with CMCNT	R|W

FLAT FORMAT (3/E)

- Register Info. File in flat type



```
%MODULE cmt
   #              name      offset_size
   %%REG_INSTANCE reg_def  3

%REG_CHANNEL reg_def
   %%TITLE   name    size  length  offset  init       access  support callback
   %%REG     CMSTR   8|16    16      0x0    -           -         -          -
   %%REG     CMCR    8|16    16      0x2    -           -         -          -
   %%REG     CMCNT   16      16      0x4    0           D        TRUE    W
   %%REG     CMCOR   16      16      0x6    0xFFFF  W|R       TRUE    -

%REG_NAME CMSTR
   %%TITLE  name   upper  lower  init  access  support  callback
   %%BIT     STR     0      0      0     W|R     TRUE     W|R

%REG_NAME CMCR
   %%TITLE  name   upper  lower  init  access  support  callback value
   %%BIT     CKS    1      0      0     W|R     TRUE     -         -
   %%BIT     CMIE   6      6      0     W|R     TRUE     -         -
```

All register items are listed independently

Bit definition for each register

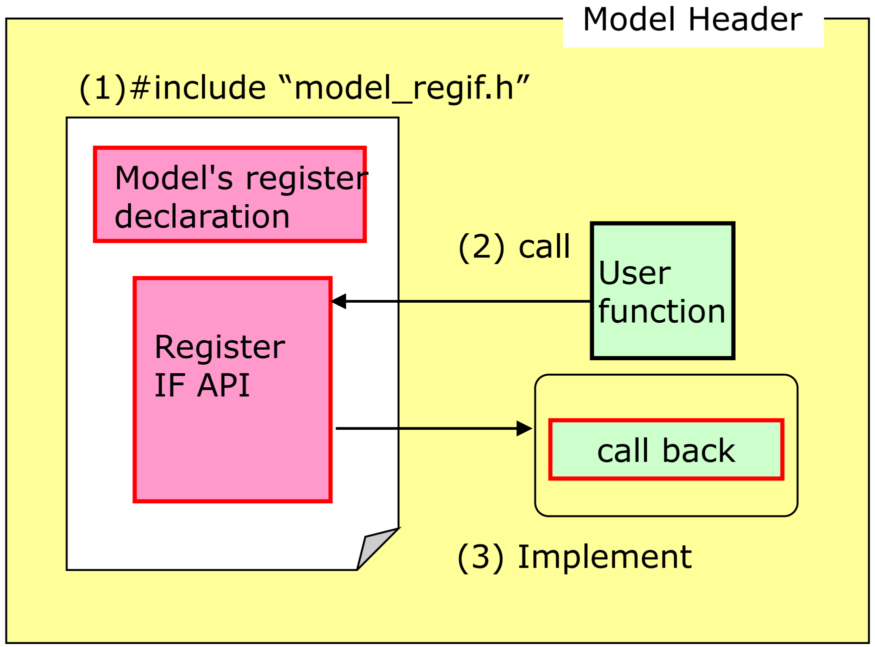
ACCESS LIMITATION

- Access limitation format

Notation	Meaning	Example
D	Decide access mode dynamically	"D": Decide access mode dynamically "W|R": enable to read/write "W": enable to write only "W0": enable to write 0 "W1": enable to write 1 "R-W0": To write 0 to register/bit, read action is required. "W0:1": If writing 0 to register/bit, updated value will be 1. This bit is write-only.
R	Enable to read	
W	Enable to write	
-	Combination access	
|	OR condition	
Number	Enable to write value. Only 1bit can be specified	
:Number	Update register's or bit's value. Only 1bit can be specified	

INTEGRATION INTO A USER CODE

1. Include the generated file in header definition
2. User function calls public APIs of Register IF which are described in next slides
3. Implement the call back functions which are explained in next slide.



```
#include <string>
#include <vector>
#include <map>
#include <cstdio>
#include “systemc.h”
#include "cmt_regif.h"
class Ccmt: public Ccmt_regif
{

// Implement call back function
private:
  void cb_CM01STR_STR(RegCBstr str);
  void cb_CM0CR_CKS(RegCBstr str);
```

PUBLIC API (1/2)

No	Explanation
1	**bool reg_wr(cuint addr, const unsigned char *p_data, cuint size);**
	This function is called to write data to an address
	argument 1 : the address need to write data argument 2 : the data need to write argument 3 : the size of data (byte)
2	**bool reg_rd(cuint addr, unsigned char *p_data, cuint size);**
	This function is called to read data from an address
	argument 1 : the address need to read argument 2 : the get data after read argument 3 : the size of data to read (byte)

PUBLIC API (2/E)

No	Explanation
1	**bool reg_wr_dbg(cuint addr, const unsigned char *p_data, cuint size);**
	This function is called to write data to an address in debug mode. In debug mode, users can write data into register easily to check user's design
	argument 1 : the address need to write data argument 2 : the data need to write argument 3 : the size of data (byte)
2	**bool reg_rd_dbg(cuint addr, unsigned char *p_data, cuint size);**
	This function is called to read data from an address in debug mode. In debug mode, users can read data from register easily to check user's design
	argument 1 : the address need to read argument 2 : the get data after read argument 3 : the size of data to read (byte)

CALL BACK FUNCTION

- Call-back function is the method that is called when a bit/register is accessed, for further process operation. It is a virtual method which is declared in Register IF class and defined in model class.

  Example: virtual void cb_CMSTR_STR(RegCBstr str) = 0

- Below is prototype of a call-back function, named ***cb_REGISTERNAME_BITNAME.*** *REGISTERNAME* is the name of the register and *BITNAME* is the name of the bit which requires call-back function. Registers without bit information are also treated as one bit, call-back function of that bit is also the call-back function of the register.

  Example: cb_CMSTR_STR, cb_CMCR_CKS

Thread/Normal	Untimed/Timed/Both	
Normal	Untimed	
Syntax	void cb_REGISTERNAME_BITNAME(RegCBstr str)	
Function	To process additional operation after the bit is written or read	
Argument	I/O	Meaning
RegCBstr str	I	Information of accessing register
Return value	Meaning	
None	-	
Explanation	These virtual methods are declared for each bit which requires call-back function. They are called from reg_wr_func() and reg_rd_func() of Register IF class and implemented in user model class	

REGISTER IF HANDLECOMMAND (1/2)

- There is a function called reg_handle_command() which used to process the handle command for Register IF.

Thread/Normal	Untimed/Timed/Both	
Normal	Untimed	
Syntax	std::string reg_handle_command (const std::vector <std::string>& args);	
Function	Process the handle command for Register IF	
Argument	I/O	Meaning
args	I	The vector of parameters and their value
Return value	Meaning	
std::string	The message output to the screen	
Explanation	This function is called to describe parameters and its arguments to support users in debugging in Register IF class	

REGISTER IF HANDLECOMMAND (2/E)

- The parameters and commands which reg_handle_command() support are as below

<table>
<tr><th>Syntax</th><th>Meaning</th></tr>
<tr><td>reg MessageLevel <fatal |error|warning|info></td><td>Select debug message level (default: fatal|error)</td></tr>
<tr><td>reg DumpRegisterRW <true/false></td><td>Select dump register access information(default: false)</td></tr>
<tr><td>reg APBAccessMode <true/false></td><td>Select for APB access mode when reading (default: false)</td></tr>
<tr><td>reg DumpBitInfo <true/false></td><td>Select for dump bit information (default: true)</td></tr>
<tr><td>reg <register_name> MessageLevel <fatal|error|warning|info></td><td>Select debug message level for register (default: fatal|error)</td></tr>
<tr><td>reg <register_name> force <value></td><td>Force register with setting value</td></tr>
<tr><td>reg <register_name> release</td><td>Release register from force value</td></tr>
<tr><td>reg <register_name> <value></td><td>Write a value into register</td></tr>
<tr><td>reg <register_name></td><td>Read value of register</td></tr>
<tr><td>reg help</td><td>Show a direction</td></tr>
</table>

Example:

```
reg APBAccessMode true
reg CMSTR 0x01
reg help
```

OUTLINE

- Overview
  - Features
  - Design Flow
  - Block Diagram
- Usage & Features
  - Usage
  - Basic Features
  - Advance Features
    - **Hierarchy format**
    - **Factor index**
    - **Local variables**
  - Other Features
- Example of Usage

HIERARCHY FORMAT (1/3)

- Register list description

Unit	Channel	Register name	Symbol	Reset value	Offset Address	Access Size(bit)	Register length
Unit 0	Common	CMT01 Start Register	CMSTR	0000h	00h	8|16	16
	CMT0	CMT0 Control Register	CMCR	0000h	04h	8|16	16
		CMT0 Counter	CMCNT	0000h	06h	16	16
		CMT0 Constant Register	CMCOR	FFFFh	08h	16	16
	CMT1	CMT1 Control Register	CMCR	0000h	0Ah	8|16	16
		CMT1 Counter	CMCNT	0000h	0Ch	16	16
		CMT1 Constant Register	CMCOR	FFFFh	0Eh	16	16
Unit 1	Common	CMT23 Start Register	CMSTR	0000h	10h	8|16	16
	CMT2	CMT2 Control Register	CMCR	0000h	14h	8|16	16
		CMT2 Counter	CMCNT	0000h	16h	16	16
		CMT2 Constant Register	CMCOR	FFFFh	18h	16	16
	CMT3	CMT3 Control Register	CMCR	0000h	1Ah	8|16	16
		CMT3 Counter	CMCNT	0000h	1Ch	16	16
		CMT3 Constant Register	CMCOR	FFFFh	1Eh	16	16

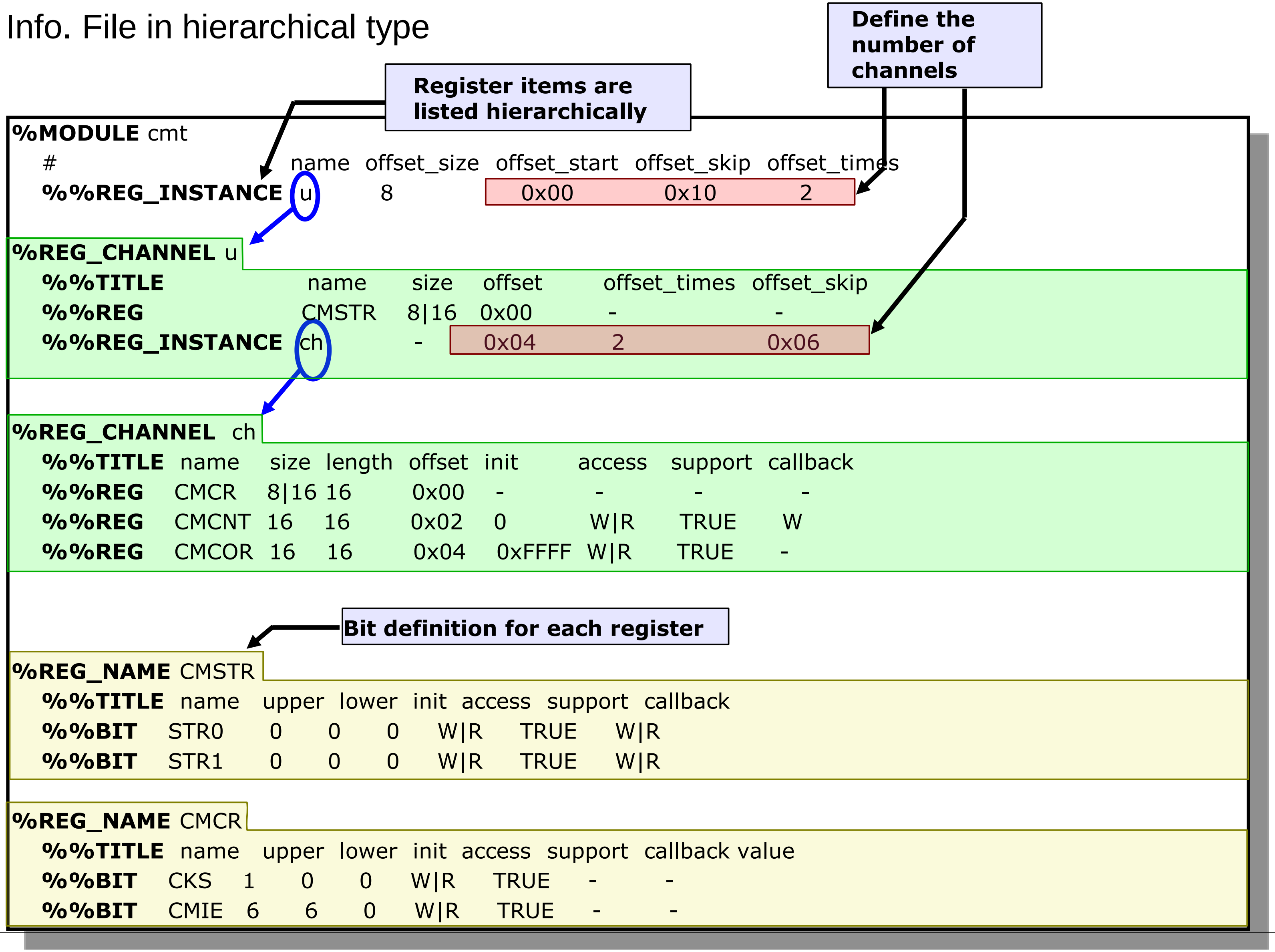
HIERARCHY FORMAT (2/3)

- Bit description

Register	Bit	Bit Symbol	Bit Name	Description	R|W
CMSTR	[15:1]	-	Reserved	These bits are always read as 0. The write value should always be 0.	R|W
	[1]	STR1	Count Start 1	0: Counter is stopped 1: Counter is started	R|W
	[0]	STR0	Count Start 0	0: Counter is stopped 1: Counter is started	R|W
CMCR	[15:7] and [5:2]	-	Reserved	These bits are always read as 0. The write value should always be 0.	R|W
	[6]	CMIE	Compare Match Interrupt Enable	0: Compare match interrupt (CMI) disabled 1: Compare match interrupt (CMI) enabled	R|W
	[1:0]	CKS[1-0]	Clock Select	00:PCLK/8; 01: PCLK/32; 10: PCLK/128; 11: PCLK/512	R|W
CMCNT	[15:0]	CMCNT	Compare match counter	CMCNT is a readable/writable up-counter to generate interrupt request.	R|W
CMCOR	[15:0]	CMCOR	Compare match constant	CMCOR sets the interval up to a compare match with CMCNT	R|W

HIERARCHY FORMAT (3/E)

- Register Info. File in hierarchical type



Register items are listed hierarchically

Define the number of channels

Bit definition for each register

```
%MODULE cmt
   #                       name  offset_size  offset_start  offset_skip  offset_times
   %%REG_INSTANCE u            8            0x00          0x10          2

%REG_CHANNEL u
   %%TITLE                  name      size   offset        offset_times  offset_skip
   %%REG                   CMSTR    8|16   0x00          -             -
   %%REG_INSTANCE ch               -       0x04          2             0x06

%REG_CHANNEL  ch
   %%TITLE  name    size  length  offset  init      access   support  callback
   %%REG    CMCR    8|16 16       0x00    -         -        -        -
   %%REG    CMCNT  16     16      0x02    0         W|R      TRUE     W
   %%REG    CMCOR  16     16      0x04    0xFFFF  W|R      TRUE     -

%REG_NAME CMSTR
   %%TITLE  name   upper  lower  init  access  support  callback
   %%BIT    STR0     0      0      0    W|R     TRUE     W|R
   %%BIT    STR1     0      0      0    W|R     TRUE     W|R

%REG_NAME CMCR
   %%TITLE  name   upper  lower  init  access  support  callback value
   %%BIT    CKS   1      0      0    W|R     TRUE     -        -
   %%BIT    CMIE   6      6      0    W|R     TRUE     -        -
```

FACTOR INDEX

“factor_start”, “factor_end”, “factor_index” and “factor_step” are used to describe the multiple register. There are 2 ways to define multiple register as below:

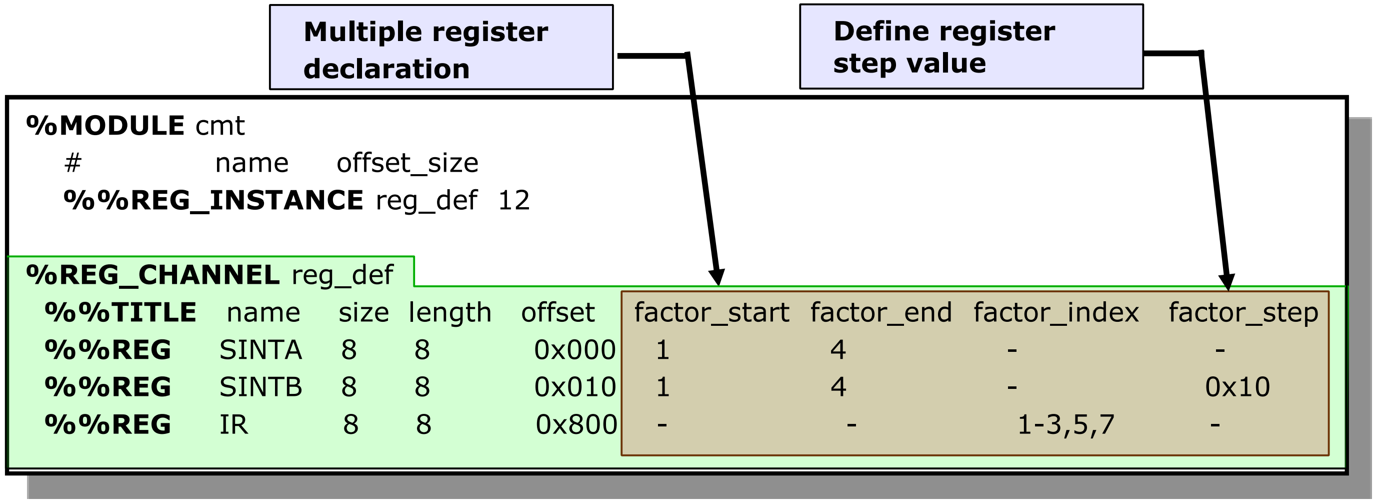
- Continuous index: users can use “factor_start”, “factor_end” to define the start index number and end index number of registers. The description for continuous index is SINTAx or SINTBy at below table.

Register name	Reset value	Offset Address	Access Size(bit)	Register length	Factor step
SINTAx (x=1..4)	0000h	000h	8	8	-
SINTBy (y=1..4)	0000h	010h	8	8	0x10
IRi (i= 1, 2, 3, 5, 7)	0000h	800h	8	8	-

- Concrete index: users can use “factor_index” to define the index of registers. The description for IRi in above table is concrete index.

FORMULA OF REGISTER INFO. FILE-2

Below is register info. file described with factor index



Multiple register declaration

Define register step value

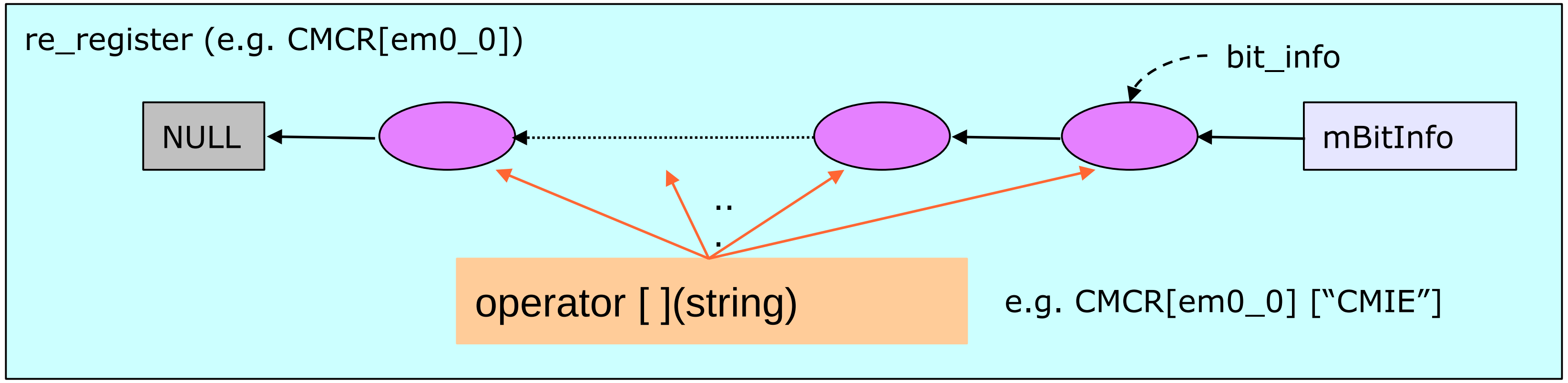
```
%MODULE cmt
   #            name     offset_size
   %%REG_INSTANCE reg_def  12

%REG_CHANNEL reg_def
  %%TITLE   name    size  length   offset  factor_start  factor_end  factor_index   factor_step
  %%REG     SINTA   8     8         0x000   1             4           -              -
  %%REG     SINTB   8     8         0x010   1             4           -              0x10
  %%REG     IR      8     8         0x800   -             -           1-3,5,7        -
```

factor_step: users can define the step number between each registers (default: factor_step equals register length)

Register IF Generator will generate definition for registers SINTA1(0x000) to SINTA4(0x003), SINTB1(0x010) to SINTB4(0x040), IR1, IR2, IR3, IR5, IR7.

LOCAL VARIABLES (1/4)

- General register (re_register class) has a list (represented by mBitInfo pointer) to store each bit information (instances of class bit_info). The designer accesses each bit by the name (string) via operation [] function. The register class searches for a pointer in the list whose name is same as specified name. It takes much time for this search.



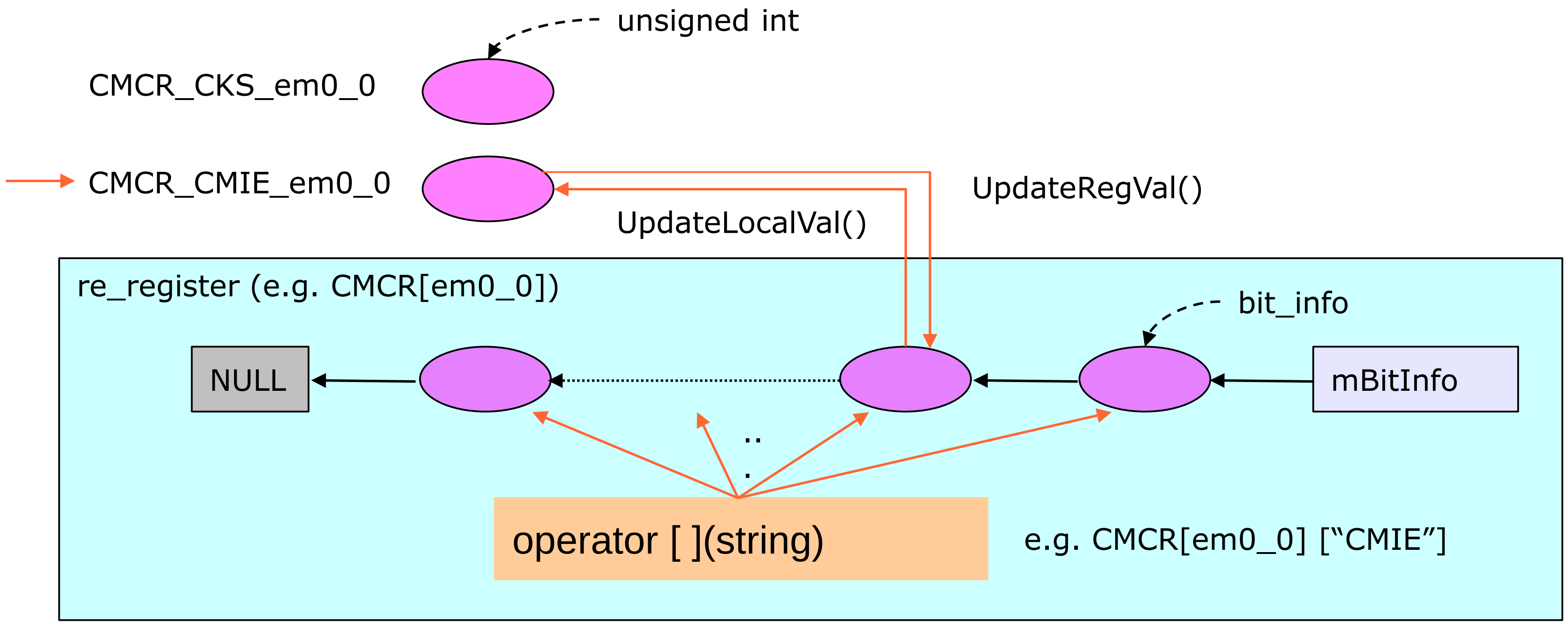
- To reduce access time, register IF supports "unsigned int" type variables for each bit of all register. Model can access to these variables directly instead of accessing to bits in re_register variable via operator[ ].

Example:

CMCR[em0_0]["CKS"] ↔ CMCR_CKS_em0_0
CMCR[em0_0]["CMIE"] ↔ CMCR_CMIE_em0_0

LOCAL VARIABLES (2/4)

- To synchronize the bit value from “unsigned int” variables to “re_register” variables and vice versa, Register IF supports UpdateLocalval() and UpdateRegVal() functions.



When value is written to re_register (via Bus IF or Command IF), UpdateLocalVal() is called by Register IF to copy the value to “unsigned int” variable too. That is the only case the synchronization is done automatically by Register IF.

LOCAL VARIABLES (3/4)

UpdateLocalVal()

Thread/Normal	Untimed/Timed/Both	
Normal	Untimed	
Syntax	void UpdateLocalVal(const unsigned int addr);	
Function	Update value from “re_register” to “unsigned int” variables	
Argument	I/O	Meaning
addr	I	Register address.
Return value	Meaning	
None	-	
Explanation	This function update the value from “re_register” to all “unsigned int” bits of specified address	

LOCAL VARIABLES (4/E)

UpdateRegVal()

Thread/Normal	Untimed/Timed/Both	
Normal	Untimed	
Syntax	void UpdateRegVal (const unsigned int addr);	
Function	Update value from “unsigned int” to “re_register” variables	
Argument	I/O	Meaning
addr	I	Register address
Return value	Meaning	
None	-	
Explanation	This function update the value from all “unsigned int” bits to re_register variable of specified address.	

OUTLINE

- Overview
  - Features
  - Design Flow
  - Block Diagram
- Usage & Features
  - Usage
  - Basic Features
  - Advance Features
  - Other Features
    - **One file generation**
    - **Writable value list**
    - **Dynamic Access**
    - **Flexible Access Size**
    - **APB Read Access Mode**
    - **VLAB meta-data file**
    - **Register Value ID**
- Example of Usage

ONE FILE GENERATION

- Register IF Generator can generate Register IF files with below mode.
  - **One file:** the Register IF class is described just in 1 file: *model*_regif.h

  This feature is enabled with following command:

  ```
  >> python3 gen_regif.py [-o/--onefile] <Register Info. File>
  ```

  Refer to slide “Usage of Register IF Generator” for detail.

WRITABLE VALUE LIST

Writable value list is a list of all data values which are allowed to write to register.

This description may include a comment/note string for each value.

Format for defining writable data is described below

Notation	Meaning	Example
Number	Support binary, decimal or hexadecimal format	“**b000, b001, b010**”: only 0, 1, 2 “**b00:PCLK_8, b01:PCLK_16, b10:PCLK_128, b11:PCLK_512**”: 4 values with string “**0x0000-0x0FFF**”: values from 0x0 to 0xFFF
:String	Specify string for each number. It will be an element of an enumeration type which users can use in their source code	
,	Specify some factors	
-	Specify a range of number values	

DYNAMIC ACCESS

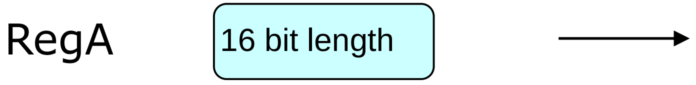
- When register or bit is set “D” at access part. It means that this register/bit can be allowed (or not allowed) to access. Refer to “Access limitation” slide for detail.
- There is a virtual function whose name is ChkAccess() is declared in general register to check access limitation dynamically. The description of ChkAccess() is shown as below, then it will be implemented in users' models.

Format		bool ChkAccess (const bool is_wr, const std::string channel_name, const std::string register_name, const std::string bit_name);
Argument	1	is_wr: write/read process
	2	channel_name: Channel name. If there is no hierarchy, it is “”
	3	register_name: Register name.
	4	bit_name: Bit name. If access to whole of register, it should be “”.
Return		true: enable to access false: forbid to access
Explanation		Return access limitation specified channel/register/bit about in argument.

FLEXIBLE ACCESS SIZE

- Register IF Generator also supports access size which different from register length
  - Access size is smaller than register length

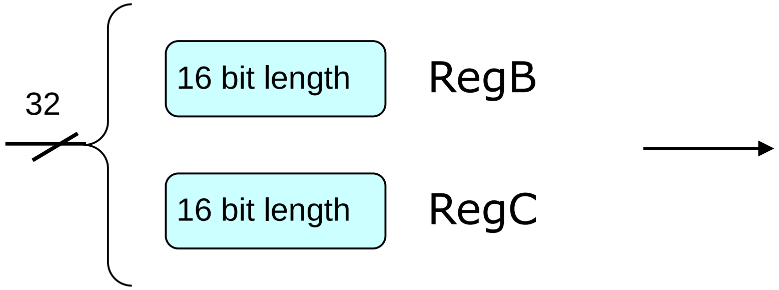
    Example: support to access 8 bits(1 byte) or 16 bit(2 bytes) of a 16-bit register by defining size is 8|16 in Register Info. File as below:



```
%%TITLE reg_name  size   length  offset
%%REG   RegA      8|16     16     0x00
```

  - Access size is greater than register length

    Example: support to access 32 bits(4 bytes) of two 16-bit registers by defining size is 16|32 in Register Info. File as below:

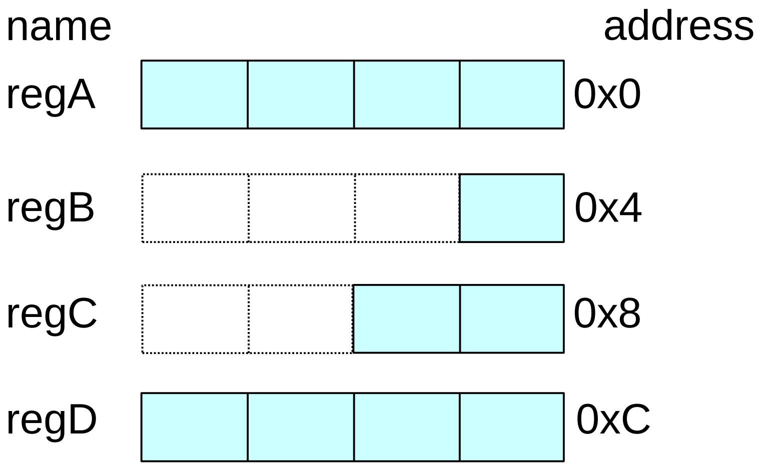


```
%%TITLE reg_name  size    length   offset
%%REG   RegB      16|32     16      0x04
%%REG   RegC      16        16      0x06
```

APB READ ACCESS MODE

- Register IF Generator supports to read 4 bytes in APB access mode by enable APBAccessMode parameter. Refer to “Register IF handleCommand” for detail.

Example: If access read 4 bytes to a register (described below) in APB access mode.
If regA or regD is accessed, the data will be returned all the bytes.
If regB is accessed, the target returns 4 bytes whose bit from 31 to 8 are all 0.
If regC is accessed, the target returns 4 bytes whose bit from 31 to 16 are all 0.



name	address
regA	0x0
regB	0x4
regC	0x8
regD	0xC

VLAB META-DATA FILE

- This file can be generated by using -a/--metadata option of Register IF Generator
- The output file name is Cmodel_metadata.py.
- The sample of a meta-data file is shown as below:

```
import vlab
bus = vlab.bus(name="m_tgt_sockets", dim=1, kind="target", width=32)
vlab.register("CMSTR"          , offset=0x0000, width=16, block=(bus,0))
vlab.register("CMCR"           , offset=0x0002, width=16, block=(bus,0))
vlab.register("CMCNT"          , offset=0x0004, width=16, block=(bus,0))
vlab.register("CMCOR"          , offset=0x0006, width=16, block=(bus,0))
```

REGISTER VALUE ID (1/4)

- Current consumption
  - The operation status of the model (e.g. idle, one-channel running, two-channel running) affects to its current consumption. Operation status is defined by values of a group of bits in the model.
  - To calculate the model's current consumption, when values of bits change, Register IF class should determine the operation status and send it to model class for calculation.
- Register Value ID feature
  - Register IF supports the "Register Value ID" feature to support current consumption calculation.
    - ➢ New keywords are supported in Register Info. file to define:
      - – the group of bits that defines operation status,
      - – value of each bit in the group, and
      - – the string assigned for each value combination (Value ID)
    - ➢ New functions and API are generated in Register IF class to check the current bit values and determine the corresponding Value ID (i.e. operation status) and send it to model class.
  - This feature is developed to be used in Web-simulator IODLL models (C++ models). In general, it can be used in any kind of model (e.g. SystemC models). To enable this feature:
    - – The keyword **%REG_VALUE_ID** should be defined in Register Info. File for code generation.
    - – The macro **USE_WEB_SIM** should be defined for compilation.

REGISTER VALUE ID (2/4)

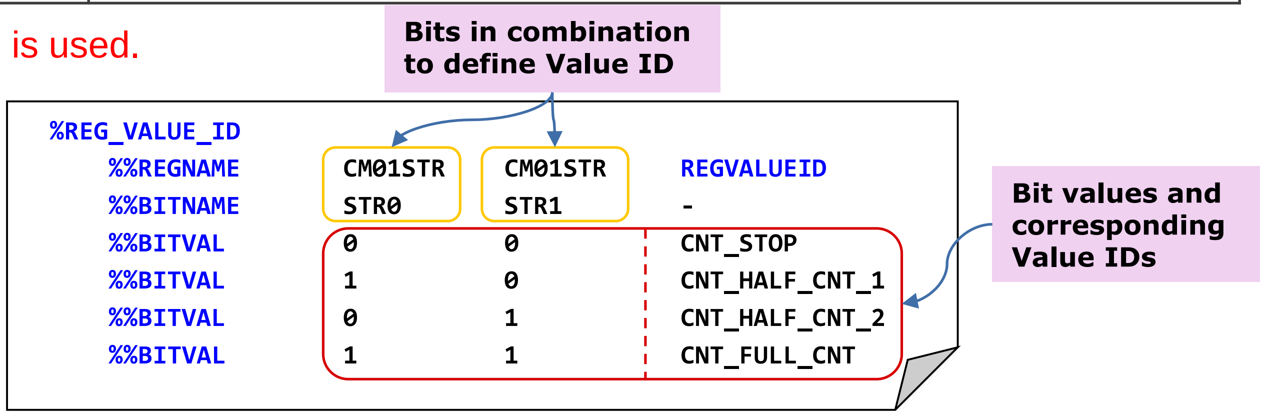
- Keywords in Register Info. File for Register Value ID definition:

Keyword	Mandatory/Optional	Meaning	Explanation
%REG_VALUE_ID	Optional	Start of register value ID part	It is used only when the feature "Register Value ID" is supported. This keyword is followed by no argument.
%%REGNAME	Mandatory (only once)	Name of registers	List of registers that contain bits in the group defining Value ID. %%REGNAME line should always finish with "REGVALUEID" keyword.
%%BITNAME	Mandatory (only once)	Name of bits	List of bits in the group defining Value ID. The number of bits in %%BITNAME should equals the number of registers in %%REGNAME.
%%BITVAL	Mandatory (at least once)	Bit value and ID definition	Combinational value of bits in the group and the assigned Value ID. Number of bit value in %%BITVAL should equals the number of registers, bits. The last argument of %%BITVAL is ID to assign for that value combination.

These keywords are mandatory only when %REG_VALUE_ID is used.

Example:

Value of CM01STR.STR0 bit and CM01STR.STR1 bit are used to define Value ID:



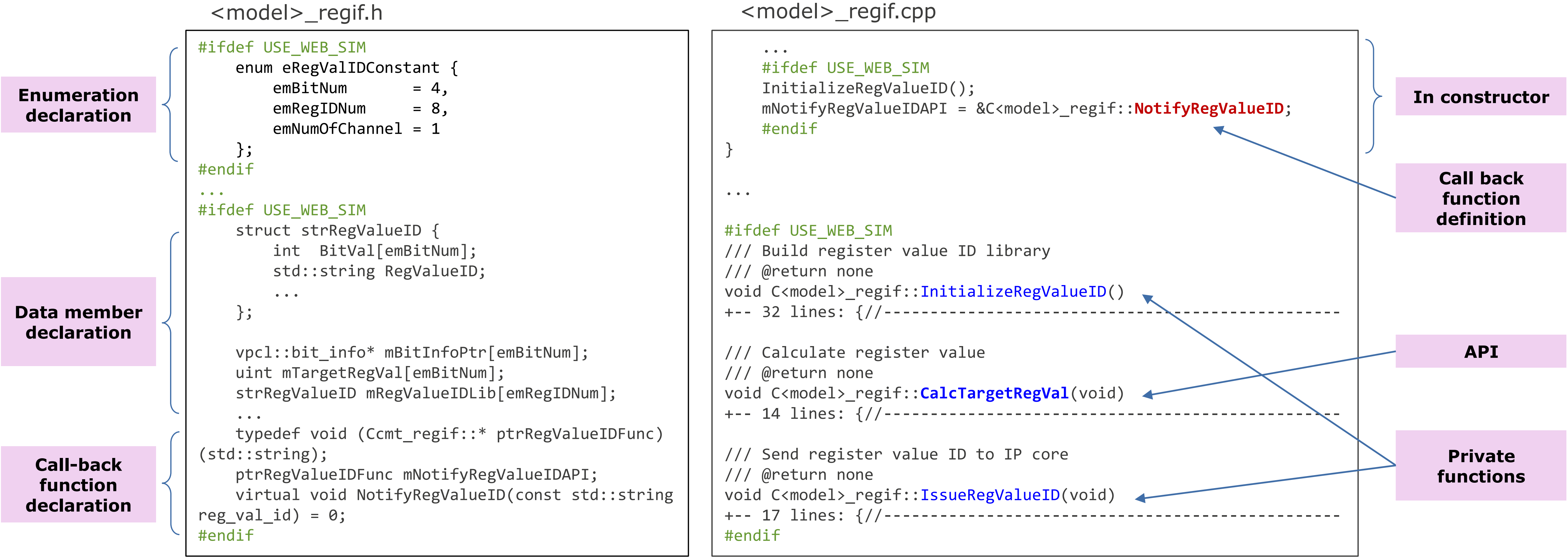
Bits in combination to define Value ID

```
%REG_VALUE_ID
    %%REGNAME    CM01STR    CM01STR    REGVALUEID
    %%BITNAME    STR0       STR1       -
    %%BITVAL     0          0          CNT_STOP
    %%BITVAL     1          0          CNT_HALF_CNT_1
    %%BITVAL     0          1          CNT_HALF_CNT_2
    %%BITVAL     1          1          CNT_FULL_CNT
```

Bit values and corresponding Value IDs

REGISTER VALUE ID (3/4)

- Code generated when Register Value ID feature is enabled:



<model>_regif.h

```
#ifdef USE_WEB_SIM
    enum eRegValIDConstant {
        emBitNum       = 4,
        emRegIDNum     = 8,
        emNumOfChannel = 1
    };
#endif
...
#ifdef USE_WEB_SIM
    struct strRegValueID {
        int  BitVal[emBitNum];
        std::string RegValueID;
        ...
    };

    vpcl::bit_info* mBitInfoPtr[emBitNum];
    uint mTargetRegVal[emBitNum];
    strRegValueID mRegValueIDLib[emRegIDNum];
    ...
    typedef void (Ccmt_regif::* ptrRegValueIDFunc)
(std::string);
    ptrRegValueIDFunc mNotifyRegValueIDAPI;
    virtual void NotifyRegValueID(const std::string
reg_val_id) = 0;
#endif
```

<model>_regif.cpp

```
    ...
    #ifdef USE_WEB_SIM
    InitializeRegValueID();
    mNotifyRegValueIDAPI = &C<model>_regif::NotifyRegValueID;
    #endif
}

...

#ifdef USE_WEB_SIM
/// Build register value ID library
/// @return none
void C<model>_regif::InitializeRegValueID()
+-- 32 lines: {//-----------------------------------------------

/// Calculate register value
/// @return none
void C<model>_regif::CalcTargetRegVal(void)
+-- 14 lines: {//-----------------------------------------------

/// Send register value ID to IP core
/// @return none
void C<model>_regif::IssueRegValueID(void)
+-- 17 lines: {//-----------------------------------------------
#endif
```

REGISTER VALUE ID (4/E)

- To check the register values for Value ID, CalcTargetRegVal() API should be called.

CalcTargetRegVal()

Thread/Normal	Untimed/Timed/Both	
Normal	Untimed	
Syntax	void **CalcTargetRegVal**(void);	
Function	Check the value of bits in defined combination and determine the corresponding Value ID.	
Argument	I/O	Meaning
None	-	-
Return value	Meaning	
None	-	
Explanation	This function determines the Value ID based on current register values.	

- CalcTargetRegVal() will call the call-back function NotifyRegValueID() of model class to inform the Value ID. NotifyRegValueID() is a pure virtual function in Register IF class and should be implemented in model class.

```
virtual void NotifyRegValueID(const std::string reg_val_id) = 0;
```

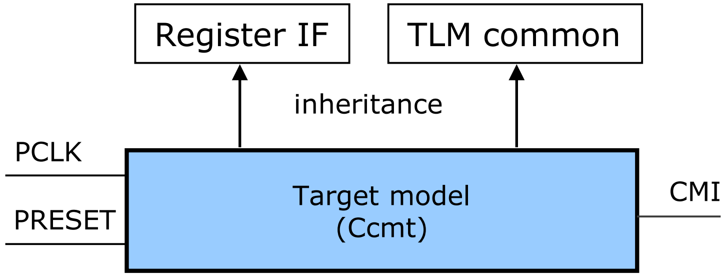
OUTLINE

- Overview
  - Features
  - Design Flow
  - Block Diagram
- Usage & Features
  - Usage
  - Basic Features
  - Advance Features
  - Other Features
- Example of Usage
  - **Specification & Execution**
  - **Sample of output files**

TARGET SPECIFICATION

Design a register IF for CMT model

1. Refer to "Flat format" slide for register, bit description



Register Info. file

```
%MODULE cmt
    #               name      offset_size
    %%REG_INSTANCE  reg_def  3
%REG_CHANNEL reg_def
    %%TITLE name   reg_name  size   length  offset  init    access   support  callback
    %%REG   CMSTR  CMSTR     8|16   16      0x0     -       -        -        -
    %%REG   CMCR   CMCR      8|16   16      0x2     -       -        -        -
    %%REG   CMCNT  CMCNT     16     16      0x4     0       W|R      TRUE     W|R
    %%REG   CMCOR  CMCOR     16     16      0x6     0xFFFF  W|R      TRUE     W
%REG_NAME CMSTR
    %%TITLE  name   upper  lower  init  access  support  callback
    %%BIT    STR    0      0      0     W|R     TRUE     W
%REG_NAME CMCR
    %%TITLE  name   upper  lower  init  access  support  callback  value
    %%BIT    CKS    1      0      0     W|R     TRUE     W|R       -
    %%BIT    CMIE   6      6      0     W|R     TRUE     W|R       -
```

Generate register IF file

```
python3 gen_regif.py cmt_regif_flat.txt

<INFO> Register description file type: Flat
<INFO> Module name found: cmt
Generate output files ...
Finish generating source files: iodefine_cmt.h
Finish generating source files: cmt_initregchk.c
Finish generating source files: cmt_regif.h
Finish generating source files: cmt_regif.cpp
Finish generating source files: cmt.h
```

SAMPLE OF OUTPUT FILES (1/7)

Sample of user code (subset) (*model*.h)

```
#ifndef __CMT_H__
#define __CMT_H__
#include "systemc.h"
#include "cmt_regif.h"

/// CMT model class
class Ccmt: public Ccmt_regif
{
public:
    SC_HAS_PROCESS(Ccmt);
    Ccmt(sc_module_name name);
    ~Ccmt();

    void cb_CMSTR_STR(RegCBstr str);
    void cb_CMCR_CKS(RegCBstr str);
    void cb_CMCR_CMIE(RegCBstr str);
    void cb_CMCNT_CMCNT(RegCBstr str);
    void cb_CMCOR_CMCOR(RegCBstr str);

};

#endif //__CMT_H__
```

SAMPLE OF OUTPUT FILES (2/7)

Sample of *model*_regif.h

```
#ifndef __CMT_REGIF_H__
#define __CMT_REGIF_H__
#include <string>
#include <map>
#include <list>
#include <cstdarg>
#include <cerrno>
#include <iomanip>
#include <sstream>
#include <cassert>
#ifndef REGIF_NOT_USE_SYSTEMC
#include "systemc.h"
#endif
#include "re_register.h"
#ifdef CWR_SYSTEMC
#include "scml2.h"
#endif
/// Register IF class of CMT model
class Ccmt_regif
: public vpcl::reg_super
{
protected:
    typedef const unsigned int cuint;
    typedef unsigned int uint;
#ifdef CWR_SYSTEMC
    typedef unsigned short REG_TYPE;
#endif
…
```

```
…
protected:
 vpcl::re_register *CMSTR [emNum_of_gr];
 vpcl::re_register *CMCR  [emNum_of_gr];
 vpcl::re_register *CMCNT [emNum_of_gr];
 vpcl::re_register *CMCOR [emNum_of_gr];

 #ifdef CWR_SYSTEMC
 scml2::memory<REG_TYPE> cwmem;
 #endif

 uint CMSTR_STR_em0;
 uint CMCR_CKS_em0;
 uint CMCR_CMIE_em0;
 uint CMCNT_CMCNT_em0;
 uint CMCOR_CMCOR_em0;

 void EnableReset(const bool is_active);
 uint bit_select(cuint val, cuint start, cuint end);
 bool reg_wr(cuint addr, const unsigned char *p_data, cuint size);
 bool reg_rd(cuint addr, unsigned char *p_data, cuint size);
 bool reg_wr_dbg(...);
 bool reg_rd_dbg(...);
 …
 virtual void cb_CMSTR_STR(RegCBstr str) = 0;
 virtual void cb_CMCR_CKS(RegCBstr str) = 0;
 …
```

Register declaration

Part of code for CoWare

Local variables

Access operation functions

Call-back functions

SAMPLE OF OUTPUT FILES (3/7)

Sample of *model*_regif.cpp

```
#include "cmt_regif.h"
#ifndef re_printf
#define re_printf get_fileline(__FILE__, __LINE__); _re_printf
#endif//re_printf

/// Constructor of Register IF class: define registers and bits
/// @return none
Ccmt_regif::Ccmt_regif(std::string name, uint buswidth)
    :vpcl::reg_super()
    #ifdef CWR_SYSTEMC
    , cwmem("register", 0x8)
    #endif
{
    CommandInit();
    CMSTR [em0] = new vpcl::re_register(0x0000, this, "CMSTR"      , name.c_str());
    CMCR  [em0] = new vpcl::re_register(0x0002, this, "CMCR"       , name.c_str());
    CMCNT [em0] = new vpcl::re_register(0x0004, this, "CMCNT"      , name.c_str());
    CMCOR [em0] = new vpcl::re_register(0x0006, this, "CMCOR"      , name.c_str());

    // Construct the register pointer list
    mCurReg = NULL;

    uint index = 0;
    mRegMap = new uint [1<<3];
    for (uint i = 0; i < (1<<3); i++) {
        mRegMap[i] = (1<<3);
    }
…
```

SAMPLE OF OUTPUT FILES (4/7)

Sample of *model*_regif.cpp (cont'd)

```
bool Cmodel_regif::reg_wr(cuint addr,                ///< [in] Writting address
                        const unsigned char *p_data, ///< [in] Writing data
                        cuint size)                  ///< [in] Data size (byte)
{
    if (size == 0) {
        re_printf("error", "Invalid access size: 0 byte\n");
        return false;
    }
    #ifdef IS_MODELED_ENDIAN_BIG
    if ((addr % mBusByteWidth) + size > mBusByteWidth) {
        re_printf("error", "Invalid access address 0x%08X with
                      access size %d bytes\n", addr, size);
        return false;
    }
    #endif

    bool ret_val = false;
    assert(p_data != NULL);
    ret_val = reg_wr_process (addr, p_data, size, false);
    return ret_val;
}
```

SAMPLE OF OUTPUT FILES (5/7)

Sample of *model*_regif.cpp (cont'd)

```
bool Cmodel_regif::reg_rd(cuint addr,           ///< [in]  Reading address
                        unsigned char *p_data, ///< [out] Reading data
                        cuint size)            ///< [in]  Data size (byte)
{
    if (size == 0) {
        re_printf("error", "Invalid access size: 0 byte\n");
        return false;
    }
    #ifdef IS_MODELED_ENDIAN_BIG
    if ((addr % mBusByteWidth) + size > mBusByteWidth) {
        re_printf("error", "Invalid access address 0x%08X with
                      access size %d bytes\n", addr, size);
        return false;
    }
    #endif

    bool ret_val = false;
    assert(p_data != NULL);
    ret_val = reg_rd_process (addr, p_data, size, false);
    return ret_val;
}
```

SAMPLE OF OUTPUT FILES (6/7)

Sample of *model*_regif.cpp (cont'd)

```
bool Cmodel_regif::reg_wr_dbg(cuint addr,                ///< [in] Writting address
                            const unsigned char *p_data, ///< [in] Writing data
                            cuint size)                  ///< [in] Data size (byte)
{
  …
}

bool Cmodel_regif::reg_rd_dbg(cuint addr,          ///< [in]  Reading address
                            unsigned char *p_data, ///< [out] Reading data
                            cuint size)            ///< [in]  Data size (byte)
{
 …
}
…
```

SAMPLE OF OUTPUT FILES (7/E)

Sample of iodefine_*model*.h

```
#ifndef __IODEFINE_CMT_H__
#define __IODEFINE_CMT_H__
struct st_cmt {
    union {
        unsigned short WORD;
        struct {
            unsigned char BYTE0;
            unsigned char BYTE1;
        } BYTES;
        struct {
            unsigned short : 15;
            unsigned short STR : 1;
        } BIT;
    } CMSTR;

     …
    union {
        unsigned short WORD;
        struct {
            unsigned short D:16;
        } BIT;
    } CMCOR;
};
#endif // __IODEFINE_CMT_H__

…
```

Sample of *model*_initregchk.c

```
#ifndef __CMT_INITREGCHK_C__
#define __CMT_INITREGCHK_C__

#include "iodefine_cmt.h"
#include "common.h"

// Please define the model base address in the line below
#define CMT_BASE 0x0
#define CMT (*(volatile struct st_cmt *)CMT_BASE)
int main() {
    cpu_setup();
    if (CMT.CMSTR.WORD != 0x0000) { fail_bp(); }
    if (CMT.CMCR.WORD  != 0x0000) { fail_bp(); }
    if (CMT.CMCNT.WORD != 0x0000) { fail_bp(); }
    if (CMT.CMCOR.WORD != 0xFFFF) { fail_bp(); }
    pass_bp();
}
#endif // __CMT_INITREGCHK_C__
```

REVISION HISTORY

Rev. No.	Contents	Agreed by customer	Approved by RVC	Checked by	Created by
ver1.0	• Copy content from User Manual phase 5 (USR-SLD-13029) • Add supported features in phase 6	M.Watanabe 2014/12/19	-	Duc Duong 2014/12/16	Thanh Phan 2014/12/16
ver1.1	• Update page 9 to add new feature "Value ID" • Add page 46~48 to describe new feature "Value ID"		Yen Nguyen 2017/04/20	Uyen Le 2017/04/20	Duc Duong 2017/04/19
ver1.2	• Page 47: Add "Mandatory/Optional" information • Add page 48 to describe sample code of new feature "Value ID"		Yen Nguyen 2017/05/05	Uyen Le 2017/05/05	Duc Duong 2017/05/04