Internal Specification Guide

(Rev 1.0)

Summary

This document is to guide designers write Internal Specification

Related Document

SystemC model: Rule Model Development flow and delivery (ver1.0)

Contents

Index of Figures

Index of Tables

1. Model summary

This chapter should include:

- Short introduction of the model, usually described in Hardware manual or REQ
- The purpose of the developed model
- The product name which the model belongs to
- The main features of the model
- The advantage of using this model (Optional)

The order of the above items depends on the writers' style

Example: This is Model Summary of RCAN

RCAN (**R**enesas **C**ontroller **A**rea **N**etwork) model is the one of the models for RX610 processor**(1)**. The main operation of RCAN model is transmitting/receiving data to/from other models through CAN I/F**(2)**. The RCAN model supports**(3)**:

- Transmitting and receiving both formats of messages, namely the standard identifier(ID (11 bits) and extended ID (29 bits).
- Two kinds of frame: remote frame and data frame.

Beside that, users can use handleCommand function with parameters (DumpInterrupt, DebugLevel, ClkpPeriod, DumpRegisterRW and help) to control RCAN model **(4)**.

(1): The product name and the purpose

(2): The introduction of RCAN model in general

(3): The main features

(4): The advantages

2. List of implemented features[1]

This chapter is to describe whether or not all the features of the model are implemented and written as the following table:

No.	Feature	Implementation[2] (Yes/No)		Reason
		Un-timed	Timed	

Table 2.1: List of implemented features

- **Feature**: the names of the features which are listed in Hardware manual or REQ. The implemented registers should not be described in the part.
- **Un-Timed/Timed**: If there are any differences between two modes, please clarify them.
- **Reason**: In case the feature is not implemented, please write down its reason.

Note: The above items are mandatory. If the model has some special features, the additional columns could be made

Example 1: Normal case for RCAN

No.	Feature	Implementation (Yes/No)		Reason
		Un-timed	Timed	
1	CAN protocol provided by ISO	No	No	Send/Receive just frame, data size and data. Do not support the other additional information (ex. CRC)
2	Message Box	Yes	No	This is a un-timed model

Table 2.2: Example of List of implemented features (Normal Case)

Example 2: Special case for TPU/RX610

No.	Feature	Detailed feature	Channel	Implementation[3] (Yes/No)		Reason
				Un-timed	Timed	
1	Count clock	PCLK/1	All	No	Yes	Clock must be implemented in Un-timed mode
		PCLK/4	All	No	Yes	
		PCLK/16	All	No	Yes	
		PCLK/64	1, 3, 5, 7, 9, 11	No	Yes	

1 This title is usually “List of implemented functions”. It should be “List of implemented features”

2 The word is usually “Support” or “Supported”. It should be “Implemented” for correctness and suitability with the title of this part

3 The word is usually “Support” or “Supported”. It should be “Implemented” for correctness and suitability with the title of this part

		PCLK/256	2, 3, 4, 8, 9, 10	No	Yes	
		PCLK/1024	3, 9	No	Yes	
		PCLK/4096	0, 1, 2, 3, 4, 5	No	Yes	

Table 2.3: Example of List of implemented features (Special Case)

3. Block diagram

This chapter is to describe the model's block diagram with the other related models to understand the position of the model in the system. The **inside** components of the model and all **related outside** components should be described.

There are two parts:

- **The figure of block diagram** should include
  - Outside components
  - Inside components
  - Connections among them

Note: Avoid writing class names (file name) or method names of any class because readers cannot know which classes we use until this part. ***Write this part as simply as possible***

- **The explanation of block diagram** should include the followings
  - The inside components and their operations in general
  - How the model connects with the outside components

Example 1: A good way

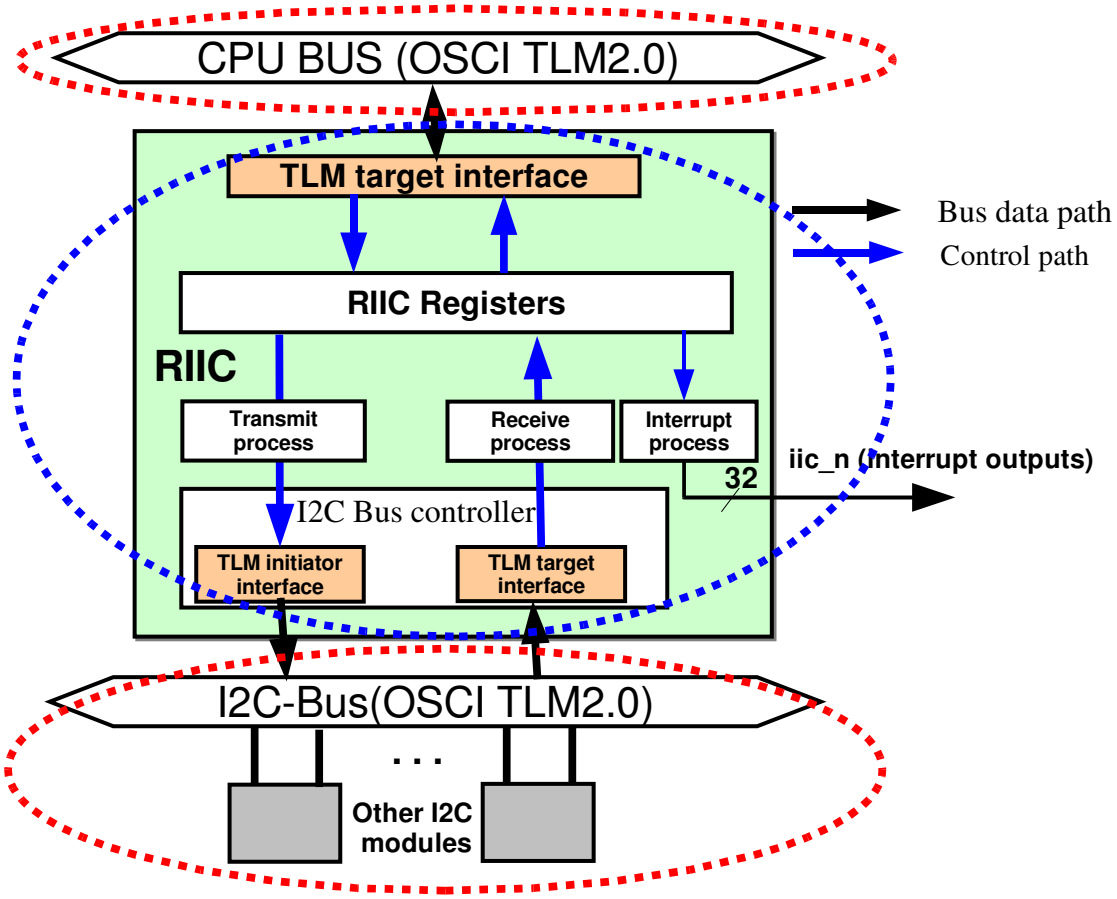
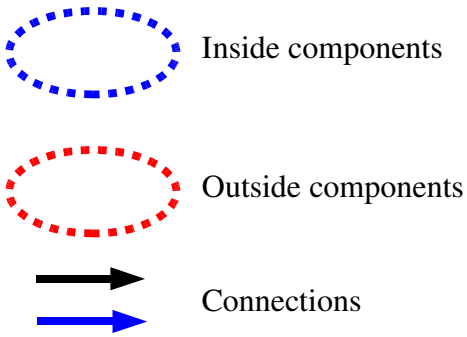


Figure 3.1: A good way: Block diagram of RX610 RIIC model



Explanation:

- RIIC communicates with CPU Bus and I2C Bus via TLM I/F components.
- RIIC includes TLM target interface responsible for receiving request (read/write) from CPU to access to registers.
- RIIC has I2C Bus controller which includes TLM initiator interface and TLM target interface responsible for sending and receiving request to/from I2C Bus.
- All interrupts signals are combined together and output as a 32-bit output port.

Example 2: A not-good way for block diagram

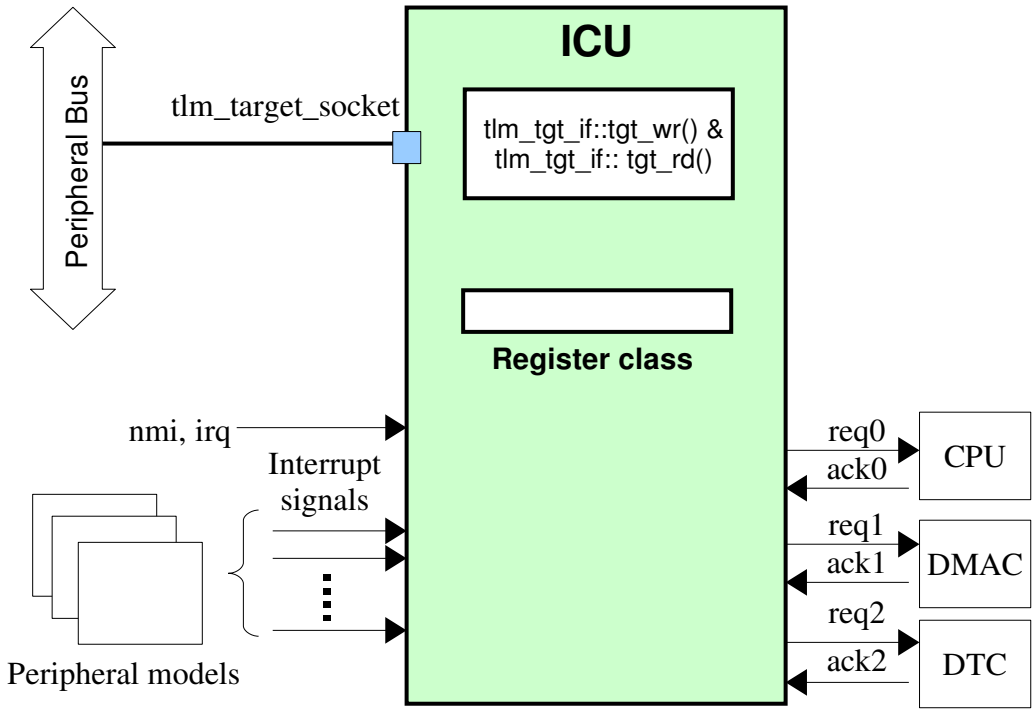


Figure 3.2: Block diagram of ICU model

- tlm_tgt_if::tgt_wr(), tlm_tgt_if::tgt_rd(): Readers may be unclear for them
- There is no connection inside the model

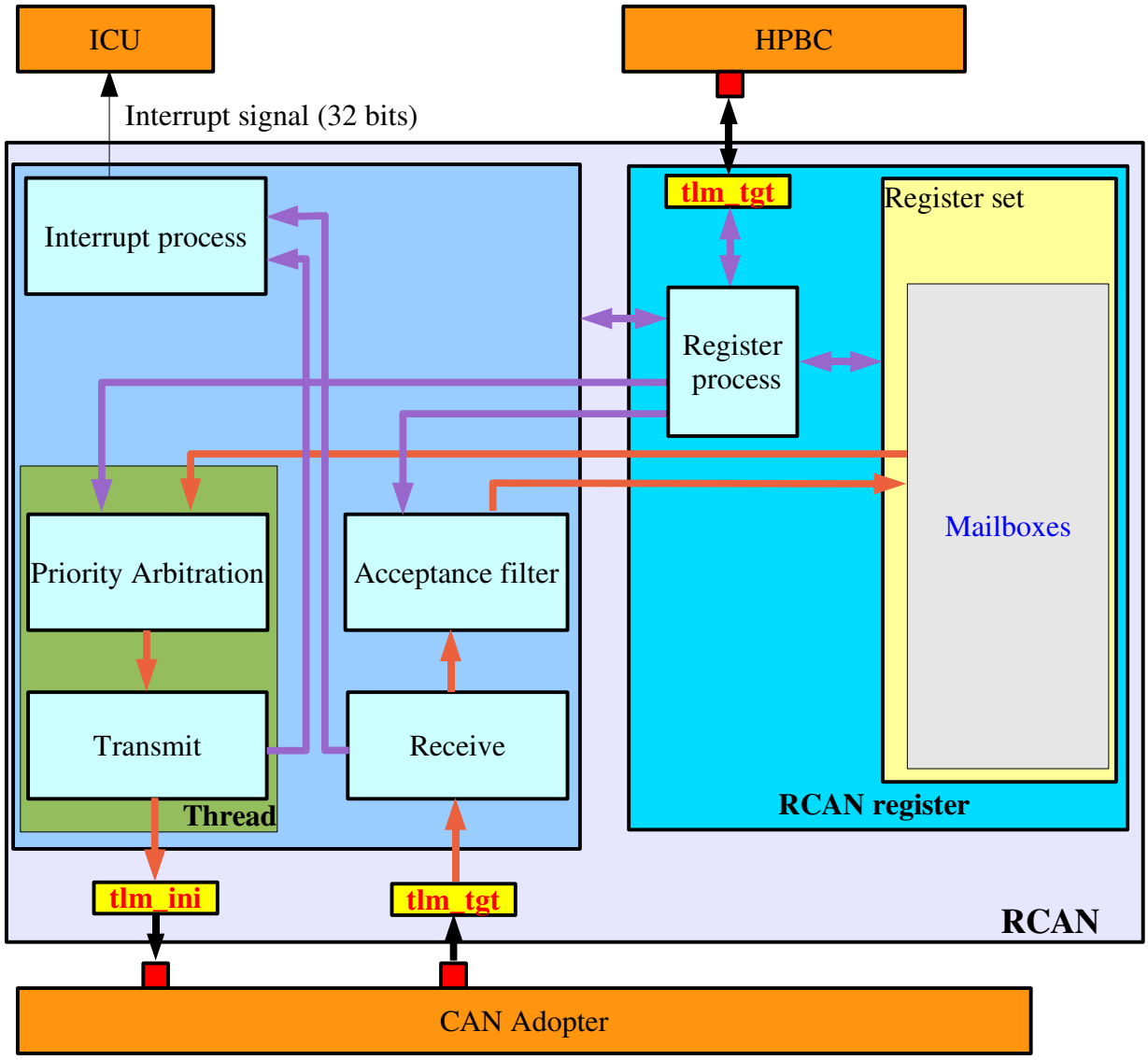


Figure 3.3: Block diagram of RX610 RCAN model

→ Please consider to make the block diagram **as simply as possible**:

- Many colors can challenge readers

- Many legends (keys) can make readers hard to refer to

Example 3: A not-good explanation

No.	Component			Description
1	*RCAN register class*	*Register set*	*Mailboxes*	Registers contain the content of message (C0MBj with j = 0 to 31)
			Other registers	The other registers are used to control or configure RCAN model.
		Register Process		This operation sub-block is used to control the processing of read/write register.
		tlm_tgt		This is TLM target I/F common class (tlm_tgt_if class). This class provides API functions to help RCAN register communicate with outside modules
3	*Operation block*	*Interrupt Process*		This operation sub-block is used to process the interrupt signal (transmit, receive and error interrupts)
		Priority Arbitration		This operation sub-block is used to arbitrate the priority of transmit mailboxes

Table 3.1: RX610 RCAN model block diagram explanation

→ This explanation is very good for explain the inside components and their operations in general. However, no connections are explained.

4. List of implemented registers[1]

This chapter is to described whether or not the registers of the model are implemented. They is described as the following table:

No.	Register	Bit	Description	Implementation (Yes/No)

Table 4.1: List of implemented registers

- **Register**: *Name* (*Short name* – #bit). Example: CAN0 Control register (COCTLR – 26 bits)
- **Bit**: *Name* [*Highest_bit* : *Lowest_bit*]. Example: BOM[12:11]

Note: The above items are mandatory. If the model has some special features, the additional columns could be made

Example:

No.	Register	Bit	Description	Implementation (Yes/No)
1	CAN0 Control register (COCTLR – 26 bits)	RBOC[13]	Forcible Return From Bus-off	No
		SLPM[10]	CAN sleep mode	Yes
		CANM[9:8]	CAN operation mode	Yes
2	CAN0 Bit Configuration Register (C0BCR – 32 bits)	BRP[25:16]	Pre-scaler division ratio	Yes

Table 4.2: Example of List of implemented registers

1 The title is usually “List of registers” or “List of supported registers”. It should be “List of implemented registers” for clearer meaning

5. List of implemented ports[1]

In “Block diagram” chapter, the connections are described in general, for example, TLM interface, or port connections, etc. However, “List of implemented port” is just to describe all ports which are listed in Hardware manual and REQ.

No.	Port	I/O	Bit width	Description	Implementation (Yes/No)	Reason

Table 5.1: List of implemented ports

Note:

- TLM interface should not be listed because it is just a way of implementation
- Ports which are not listed in Hardware manual or REQ should not be listed

Example:

No.	Port	I/O	Bit width	Description	Implementation (Yes/No)	Reason
1	bus	I	32	An interrupt request from BUSERR is assigned to bus[0].	Yes	
2	fcu	I	32	Interrupt requests from FRDYI and FIFERR are assigned to fcu[1:0], respectively.	Yes	
3	cmt	I	32	Interrupt requests from CMTn (n=3 ~ 0) are assigned to cmt[3:0], respectively.	No	Un-supported for CMT model
4	can0	I	32	Interrupt requests from TXM0, RXM0, TXF0, RXF0 and ERS0 (vector number: 60 ~ 56, edge-trigger protocol) are assigned to can0[4:0], respectively.	No	Un-supported for RCAN model

Table 5.2: List of implemented ports

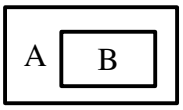
1 The title is usually “List of supported pins” or “List of supported ports”. Please unify it as “List of implemented ports”

6.Direction for user

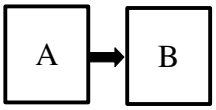
6.1 File Structure

This chapter is to describe the relationships among all files related to the model. There are two parts:

- **A figure to express file relationship**: two relationships as the followings



: File A includes file B



: The prototypes were declared in the file A will be defined in the file B

- **File description to explain all files in the figure**[1]:

No.	File name	Version	Developed/Reused	Description

Table 6.1: File description

- **Developed/Reused**: the column is to indicate the origin of the file

  Developed: this file is created for the model

  Reused: this file is reused from the common classes or the other models
- **Version**: If the file is reused, its version should be remarked
- **Class name**: maybe class name is clarified in some documents as 1 column. However, it is unnecessary because this part is all about files not classes.

Example:

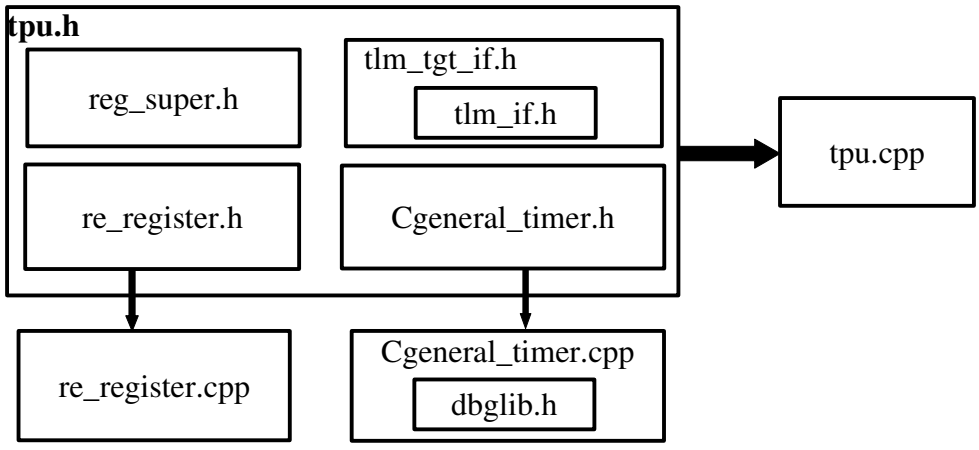


Figure 6.1: Example of File relationship

1 Maybe this table is written in many styles, but please unify it

No.	File name	Version	Developed/ Reused	Description
1	tpu.h		Developed	header file of TPU
2	tpu.cpp		Developed	implementation of TPU
3	Cgeneral_timer.h	1.12	Reused	header file of General Timer
4	Cgeneral_timer.cpp	1.14	Reused	implementation of General Timer
5	dbglib.h	1.0	Reused	support some functions for debugging in the General Timer model
6	re_register.h	1.10	Reused	header file of the re_register class
7	re_register.cpp	1.11	Reused	implement the attributes and operations of common register class
8	reg_super.h	1.6	Reused	general class for models to access the memory array
9	tlm_tgt_if.h	1.7	Reused	header file of the tlm_tgt_if class
10	tlm_if.h	1.5	Reused	Header file of the tlm_if class

Table 6.2: Example of File description

6.2 Input/Output files

Please write “There is no input or output file.” if the model has no input/output files. Or else, please write these file as the following table

No.	File name	Type	I/O	Description

Table 6.3: Input/Output files

- **Type**: the file type must be defined, for example, PERL, C++ source, CSHELL, etc.
- **Description**: this column includes the purposes, for example, whether the file name is writable or not, whether the file is optional or mandatory, etc.

Note: “How to write input files” or “How to check the output files” should not be included in this part. They will be described in USR later.

6.3 How to connect to Verification Environment

This chapter includes two parts:

- **Scenario of verification environment (Optional)**: Because maybe there are many different ways to connect the model to the different verification environments, the scenario of this model needs to be described.

- **How to connect to verification environment (Mandatory)**: this part is described step by step

Example: This is an example from RIIC model

- Scenario: The Verification environment phase 2

- Connection: there are five basic steps to connect RIIC model to a verification environment

  Step 1: Declare an instance of Cri2c class in the top model or another suitable model.

  Step 2: Connect RIIC's target socket named m_tgt_socket with the initiator socket of bus model which connects to CPU.

  Step 3: Connect RIIC's target socket named m_i2c_tgt_socket with the initiator socket of I2C bus model.

  Step 4: Connect RIIC's initiator socket name m_i2c_ini_socket with the target socket of I2C bus model.

  Step 5: Connect RIIC's 32-bit interrupt signal named iic_n with INTC or another suitable model that receives the interrupt signal.

6.4 Handling command[1]

Describe the handling commands of the model as the following table

No.	Command	Argument			Description
		No.	Type	Default	

Table 6.4: List of Handling command

- **Command**: the name of the command
- **Description**: please write it as the following format:

[Purpose]
Argument 1: *[description for argument 1]*
Argument n: *[description for argument n]*
Messages: *[Only Write the item when the command notifies a message, not including Error message and Success message]*
[Message format]

1 The title is usually “handleCommand parameters”, please unify it to “Handling command”

Example:

No.	Command	Argument			Description
		No.	Type	Default	
1	WRITE_RG	1	Hex number with 0x	None	Write any value to the specific register of model - Argument 1: the address of the register - Argument 2: the written value - Message: If successful, the following message will be printed [instance_name] Writing is right [address]: [value]
		2	Hex number with 0x	None	

Table 6.5: Example of List of Handling command

6.5 Defined macro

Define the macros of the models as the following table

No.	Macro	Value	File name	Description

Table 6.6: Example of File description

- **Macro**: the name of the macro
- ***File name***: which file the macro is defined in

6.6 Component message

Messages of the model is described as the following table

No.	Severity (Error/Warning/Error)	Message	Description

Table 6.7: List of component messages

Example:

No.	Severity	Message	Description
1	Error	[RCAN] Error: Access size of [REGISTER] is wrong : [number] byte	Access byte is illegal (bigger than register size)
2	Error	[RCAN] Error: Write [value] to [bit] is prohibited	Output message when the setting prohibited value to the bits of register
3	Error	[RCAN] Error: Writing wrong [address] of [REGISTER]	Write to wrong address register
4	Error	[RCAN] Error: Reading wrong [address] of [REGISTER]	Read from wrong address register
5	Error	[RCAN] Error: Pointer data is Null in [FUNCTION]	Pointer data is null in [FUNCTION] : [FUNCTION] is functions that relate to accessing data such as tgt_wr, tgt_rd

Table 6.8: List of messages in RX610 RCAN

7. Flow diagram

This chapter is very important for readers to have a first sight of “What the model does?”. From the view of readers, there are some suggestions:

- The readers have no ideas of which classes or methods are implemented

→ Please AVOID using class names, method names, and attribute names

- It is difficult to have a clear picture when reading Sequence diagram of UML. Actually, specific objects (ex. class) are necessary for Sequence diagram

→ Please use another way to help readers understand which the model does more clearly

- This chapter is to help the readers understand the model from overview to detail, NOT to make readers confused or challenged

Flow diagram includes two main parts:

- **Main flow**: Provide readers a overview of the model 's operations. The readers must know what begins, what operations are, and what ends.
- **Detail/Specific flow**: there may be many flows in this part in order to detail the operations which are defined in Main flow

The followings are to explain these two parts with some examples from CRC/RX600 project

Note:

- All diagrams in this chapter are written according to **Activity Diagram of UML**
- There must be explanation for each diagram
- Please be as simple and clear as possible

7.1 Main flow

The main flow includes

- Condition to start the model
- Condition to end the model. Sometimes, if there is not such a condition, please go to the end if all operations of the model are done
- All operations of the model. Please be overview, NOT detail. All detail will be expressed in the other flows

Example:

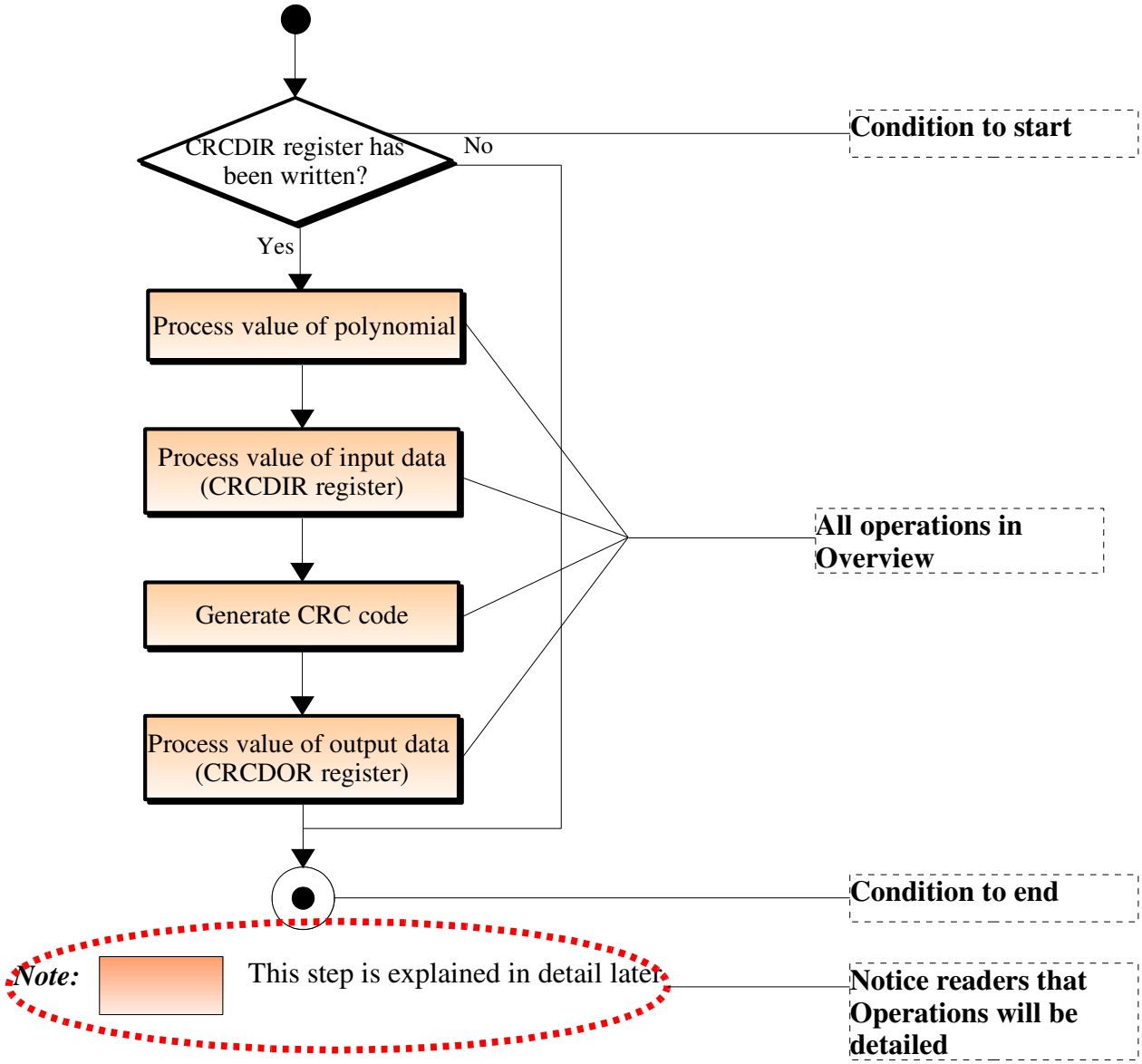


Figure 7.1: Flow diagram of Main flow of RX600 CRC model

The titles of the other flows depend on the Main flow. For example, with the above main flow, there are four flows **at least**:

7.2 Processing value of polynomial

7.3 Processing value of input data

7.4 Generate CRC code

7.5 Process value of output data

For detail flow, the more flows can be written if writers want to separate the operations into many flows. For example, if writers want to separate “Processing value of polynomial” into two

flow, they will write as the followings

7.2 Processing value of polynomial

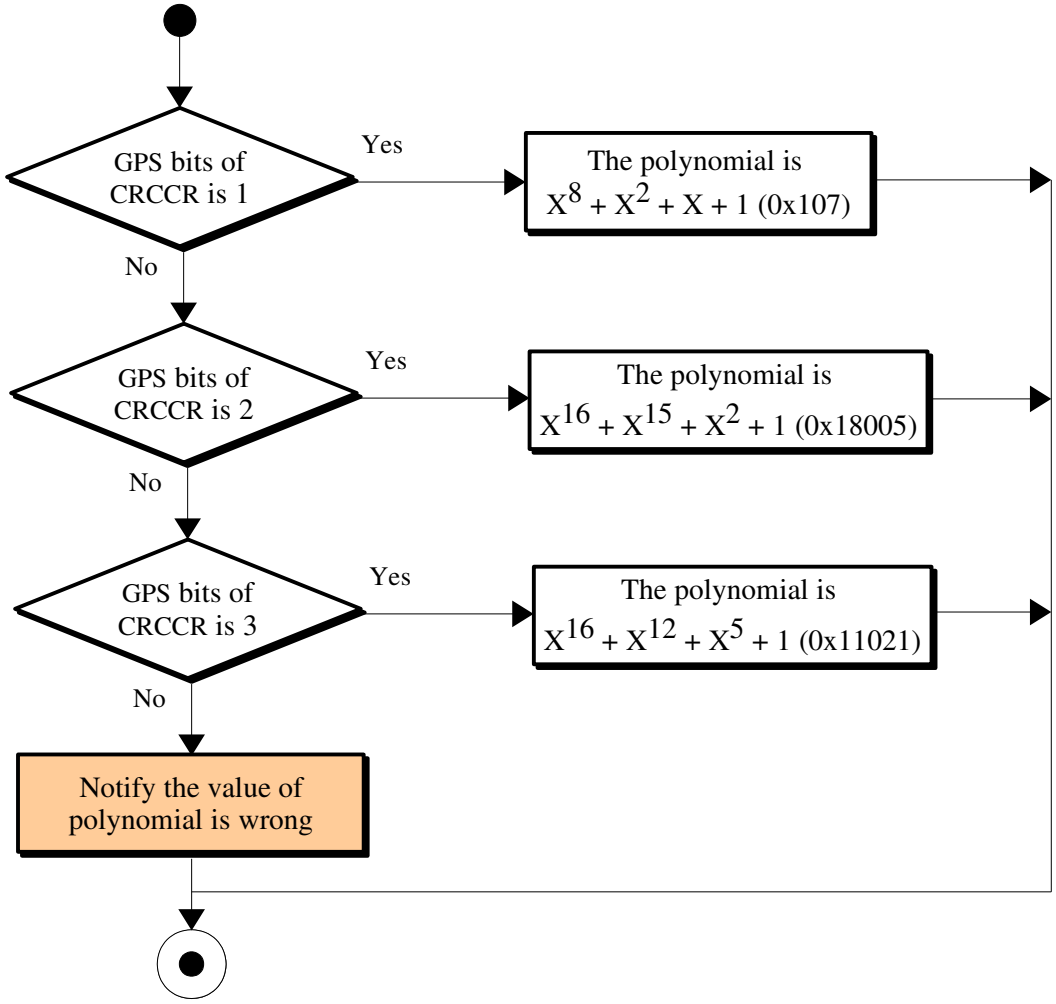


Figure 7.2: Flow diagram of processing polynomial value

This is the main flow of “Processing value of polynomial”. Writers color “Notify the value of polynomial is wrong” to indicate that this will be explained in more detail. The next flow is that as 7.2.1.

7.2.1 Notify the value of polynomial is wrong

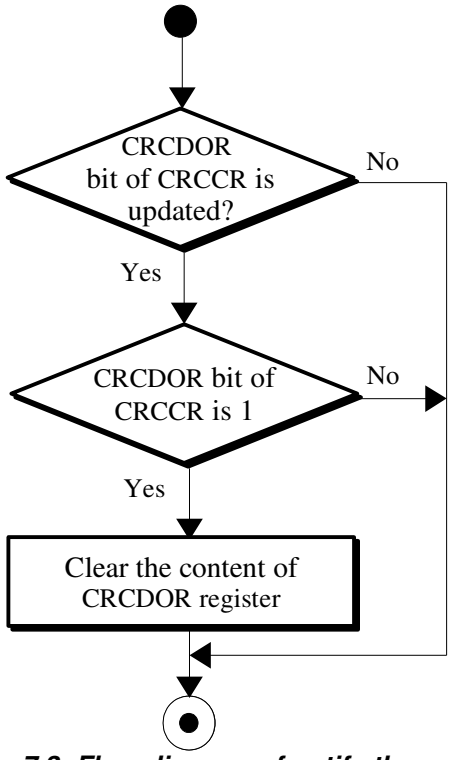


Figure 7.3: Flow diagram of notify the value of polynomial is wrong

8. Class explanation

The operations of the model are detailed in chapter 7. The purpose of this chapter is to express **"How the model is designed/implemented to satisfy such operations"**

8.1 Class Relationships[1]

This chapter is a chance for writers to introduce the classes to readers. It includes two parts

- **Figure of class relationships**: the figure is drawn according to **Class Diagram of UML**
- **The explanation of this figure**: the purpose is to introduce the classes as the following table

No.	Class name	Explanation

Table 8.1: Explanation of class relationships

Example:

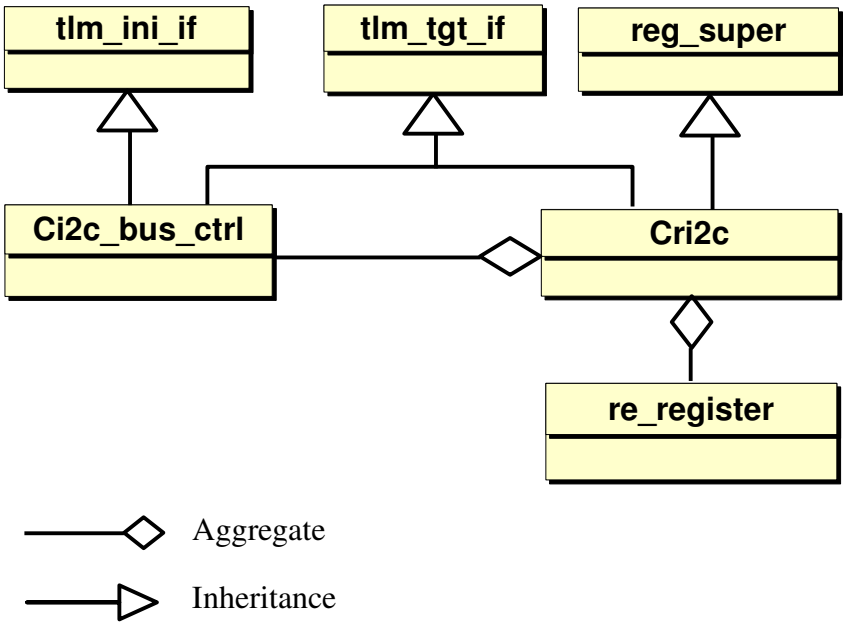


Figure 8.1: Example of Figure of Class Relationships

1 Sometimes the title is written as "Relationship of classes", please unify it to "Class Relationships" because "Relationship + of " is wrong in English

- Explanation:

No.	Class name	Explanation
1	tlm_ini_if	TLM initiator common class
2	tlm_tgt_if	TLM target common class
3	re_super	Class to read/write the memory
4	Ci2c_bus_ctrl	Ci2c_bus_ctrl classes is also instantiated in this class to communicate with I2C Bus.
5	Cri2c	RIIC model class
6	re_register	Cri2c class has many registers in it so re_register class is used as instance

Note: Usually, the explanation is written as the followings

- RIIC receives request from CPU to read or write data from/to registers. Therefore, TLM common class I/F (tlm_tgt_if) is used as an inherited class.
- Ci2c_bus_ctrl class is responsible for sending and receiving request from I2C Bus. Therefore, TLM common classes I/F (tlm_ini_if and tlm_tgt_if) are used as inherited classes.
- Cri2c class has many registers in it so re_register class is used as instance. Ci2c_bus_ctrl classes is also instantiated in this class to communicate

→ We don't need to explain like that because **Class Diagram of UML** helps us do this in case that the readers are familiar with UML.

8.2 Class class_name

8.2.1 Summary

This chapter is to give readers a overview of *class_name* class. It includes:

- Purposes: Why to design this class
- Special information (Optional)

Example: The summary of simple_mem_if class in CRC/RX600 project

This class is simple memory interface that provides API functions helping other modules access registers of CRC model. In this project, this class is used to verify CRC model, but it can separate communication portion from computing portion of CRC model. **(1)**

This class doesn't have any attribute, but the header file includes two global arrays declared as ***extern unsigned char m_array[0x10000]***, and ***extern unsigned char p_array[0x10000]***. These global array are considered as memory array which is accessed through call-back functions of general register class and port array. **(2)**

(1): Purposes

(2): Special information

8.2.2 Enumeration

This chapter is to describe the enumerations in *class_name* class

No.	Enumeration name	Element name	Meaning

Table 8.2: List of enumerations

8.2.1 Structures

This chapter is to describe the structures in *class_name* class

No.	Structures name	Element name	Type	Size	Meaning

Table 8.3: List of structures

8.2.2 Attributes

This chapter is to describe the attributes in *class_name* class

No.	Attribute name	Type	Default value	Level (Public/Private/Protected)	Meaning

Table 8.4: List of attributes

8.2.3 Function-call diagram

This chapter is usually written after the definitions of the methods of *class_name* class, but It is difficult for readers to have a whole picture of the methods. Therefore, this chapter should be written first. There are two parts

- **A figure of function call**: this is a overview picture of the implemented functions
- **Thread/Method description**: this specifies the threads/methods

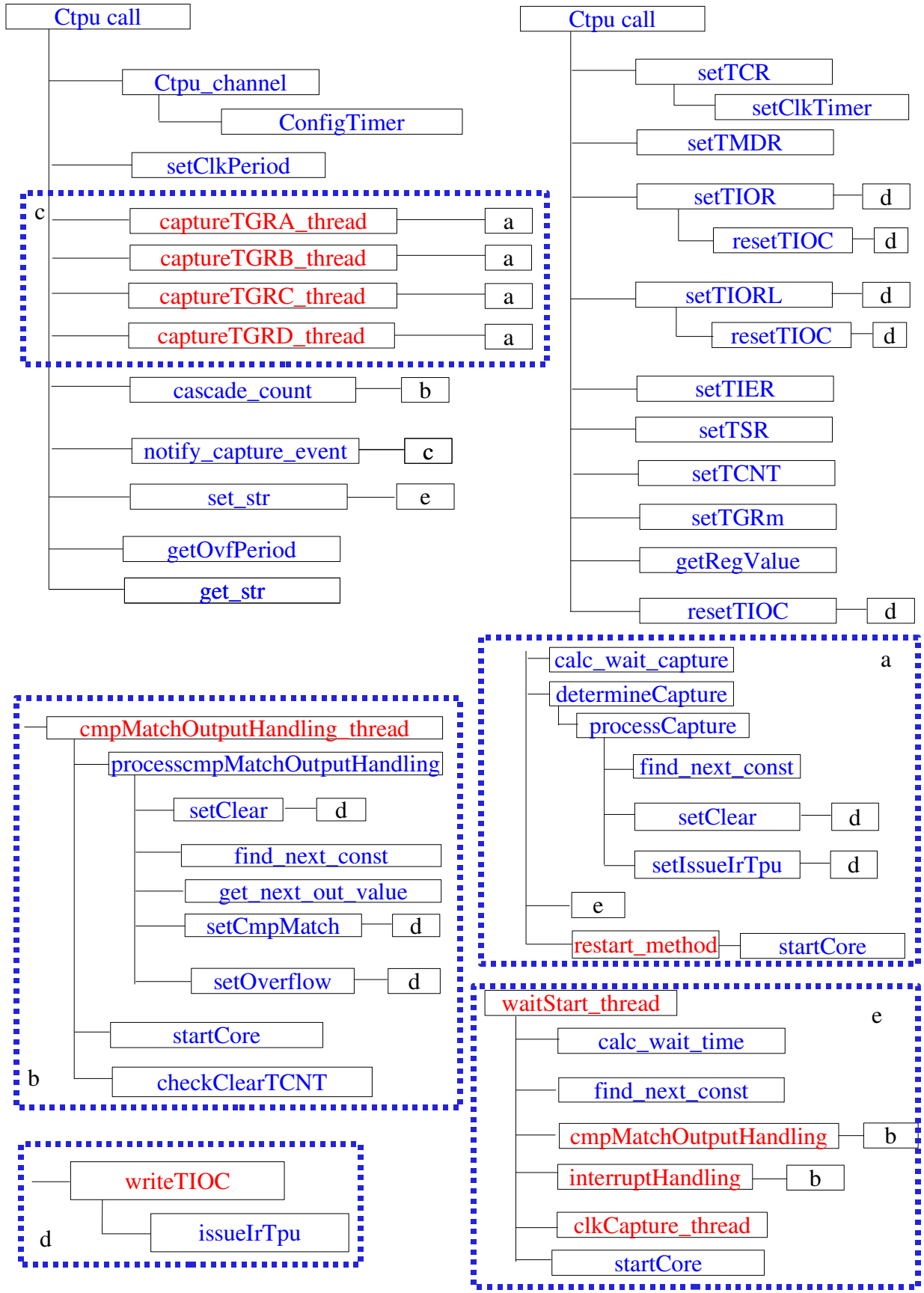


Figure 8.2: Function call diagram of Ctpu_channel class

The above figure is an example of function call diagram. This figure is written as usual with one additional detail: **Color the threads/methods.**

The Thread/Method description is written in the following table

No.	Name	Thread/Method	Event		Note
			Name	Direct function to notify	

Table 8.5: Thread/Method description of class_name class

- **Event**:
  In case the method/thread is triggered by PIN, please write the name of PIN in “Name” column and “Notified by PIN” in “Direct function to notify” column
  In case the method/thread is virtual method of the inherited class, please merge 2 columns “Name” and “Direct function to notify” into 1 column and write the short information about when the thread/method executes
- **Note**: In case the method/thread is virtual method of the inherited class, please write the name of this class in this column

Example:

No.	Name	Thread/ Method	Event		Note
			Name	Direct function to notify	
1	captureTGRA_thread	Thread	capture_event_a	notify_capture_event	
			tioca_i	Notified by PIN tioca_i	
2	interruptHandling	Method	Notified when the overflow happens		Virtual method from Cgeneral_timer

Table 8.6: Example of Thread/Method description of class_name class

8.2.4 Function description

The chapter is to define all functions which are mentioned in the chapter “Function call diagram”. There are three parts: Public functions, Protected functions, and Private functions. Each function is described as the following table

Thread/Normal		**Untimed/Timed/Both**	**Access Control**
Syntax			
Function			
Argument	**I/O**	**Meaning**	
	-	-	
Return value		**Meaning**	
		-	
Explanation			

9. Operation description

The flow diagrams in chapter No.7 are reflected in this chapter. All operations of the model are detailed in “Operation description” by **Sequence Diagram** of UML, and in equivalence with the **Activity Diagrams** (flow charts) in chapter No.7.

This chapter will help readers understand what the implemented classes do and how they work together.

Revision History				
Rev.	**Modified Contents**	**Approval**	**Reviewed by**	**Created by**
1.0	New Creation			Khang Le 11/02/2010