SC-HEAP Modeling Guideline

2019.1.18
Eiichi Arai
Automotive Control MCU Software Section
Automotive MCU Software Department
Automotive Software Business Division
Renesas Electronics Corporation

Purpose

This document describes the Guideline for peripheral macro development which is connected to SC-HEAP simulator.

Table of contents

Table of contents

1. Modeling Guideline and Requirements

1.1. Summary of Model Guideline & Requirements

The baseline requirement is for timed-functional behavior-level models that can be used as executable specifications and are sufficiently accurate to support full Driver and Timing Critical Software development and test.

On the other hand, high speed simulation is also required.

As above solution, SC-HEAP supports 2 types of simulation mode.

One is a high accuracy simulation mode called AT(Approximately timed) mode.

The other is a high speed simulation mode called LT(Loosely timed) mode.

We will support the dynamic switching between both modes.

1.2. Target Abstraction

The target abstractions of the SC-HEAP_E3 are as follows.

Abstraction defined by STARC*		OSCI TLM2.0 Coding style	Target of the models
Untimed		Loosely Timed	
Approximately timed	ATTR		LT mode
	ATBP	Approximately Timed	AT mode
	ATBC		
Cycle-accurate			

*:page 2-7 of the ”STARC_TLMGuide_2ndr2(2011Mar23)_j.pdf

1.3. Common Model Requirements

All Models shall be:

- Functionally complete: Models will be feature complete with respect to IP Block Guides. All module specification features are modelled, including all registers, interfaces, modes, etc. unless explicitly agreed on any exclusions.
- The models are timed-functional behavior-level models and will implement all programmer view behavior, including all registers, interrupts, control flow, and all data flow transfers.
- Event driven
- Vendor, Tool and Backplane independent.
- Provide error checking of user violations in programming
- Support debug accesses
- Support VCD dumping of ports
- Can optionally provide more detailed tracing information
- Cycle Approximate. The combination of Control Cycle Accuracy and Data Token Accuracy defined below is called Cycle Approximate. (AT mode)

The CPU to Module bus interface, including the DMA Communication model, shall be:

  - Bus Level Transactions for High Performance
  - Cycle Accurate at Transaction Boundary at AT mode.

1.4. Control Model Specific Requirements

The list of Control Modules includes modules such as:

- Timers, CLKGEN, Reset, LVDT, DMA, INTC, Multi-LIN Master, etc…

For all Control Module Models they shall:

- Communication is modelled as bit level digital signals
- Where applicable, the models will be parameterize-able and reusable for new MCU configurations (parameterized channels, memory size etc.)
- Models shall be Cycle Accurate at AT mode.
- Accuracy at AT mode shall be sufficient for Driver and timing critical SW development and test.
- On the other hand, LT mode support fast simulation. (The accuracy shall be sufficient for Driver SW development.)

1.5. Communication Model Specific Requirements

The list of Communication Modules includes modules such as

- UART, CSI, AFCAN, FLEXRAY, Ethernet, ADC, MLB

For all Communication Module Models they shall:

- Communication is Token based.
- Serial communication will be abstracted by a token-based one for high performance:

Internal MCU Data Transfers	Bus Cycle Token Accurate (resolution at bus cycle token)
External MCU Data Transfers	Data Token Accurate (resolution at one token)
Control Signals	Cycle Accurate

- Accuracy at AT mode is sufficient for Driver and timing critical SW development and test.
- On the other hand, LT mode support fast simulation. (The accuracy shall be sufficient for Driver SW development.)

1.6. Model Scope Guidelines

- At AT mode, each module model shall be written at the cycle-approximate abstraction level. Core CPU, Bus, DMA and interrupt controller modules and other control module models shall be written at the cycle-accurate abstraction level.
- At LT mode, each module model shall be written at the loosely-timed abstraction level.
- Each module model shall encapsulate its own data within a structure or a class
- Each module model shall check for end user violations of specification recommendations and/or requirements.
- The messages shall be routed through a common display routine so that they can be easily filtered.
- Each message shall describe the module name (instance name) of the simulation object that detected the problem as well as any address/data, or usage information that is appropriate.
- Messages shall be classified into the following categories
  - ERROR
  - WARNING
  - INFO
- To execute the simulator is executed both on 32 bits and 64 bits Personal Computer, “long” or “long double” shall be avoided from source code of the models as variable type.
  The simulator shall be applied to ILP32, LP64 and LLP64 as Data Type Models.

1.7. Bus Interface Requirements

- Coding style of the Approximately Timed using the return path(2 phase AT) is used for high accuracy simulation mode. (AT mode)
- Coding style of the Loosely Timed with temporal decupling is used for high speed simulation mode. (LT mode)
- The CPU to Module bus interface, including the DMA Communication model, shall be Cycle Accurate at Transaction Boundary at AT mode.
  LT mode is not untimed but looser than AT mode. e.g.The bus interface considers bus collision at AT mode but not at LT mode.

1.8. Error/Warning/Info Message Output Requirements

- Error Message

  This message should be output:

  - when invalid value is set and it is irrecoverable.

    Ex. In case that address area of peripherals are overlapped in bus map.

  - when accessed data has already been de-allocated.

    Ex. In case that a object of tlm_generic_payload has already been de-allocated and a bus slave IP accesses it.

  Simulation should be stopped after the message is output.

- Warning Message

  This message should be output:

  - when invalid value is set and it is recoverable. Simulation should continue to run with default value.

    Ex. In an ADC which AD-ch is parameterized and is possible to set up to 16ch, simulation continues to run with default value(ex.16ch) when user sets 17ch.

  - when a function is operated in violation of restriction which is described in HW specification of IP. The operation should be ignored.

    Ex. In register access, user writes ‘1’ to bit *A* during bit *B* = ‘1’, though write-operation of bit *A* during bit *B* = ‘1’ is restricted in the specification.

  Simulation should not be stopped after the message is output.

- Info message

  This message might be output:

  - when there are any information for debugging

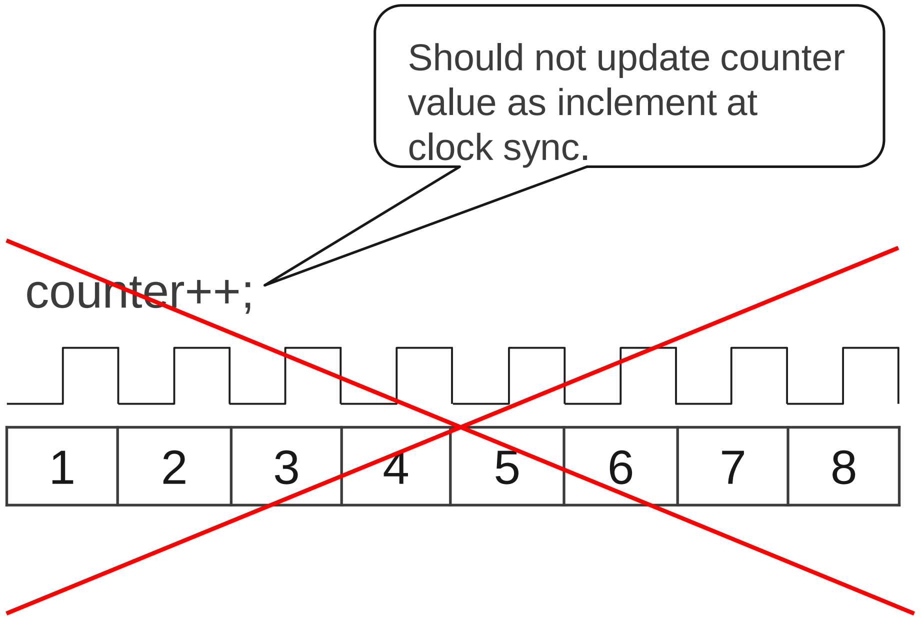
  Simulation should not be stopped after the message is output.

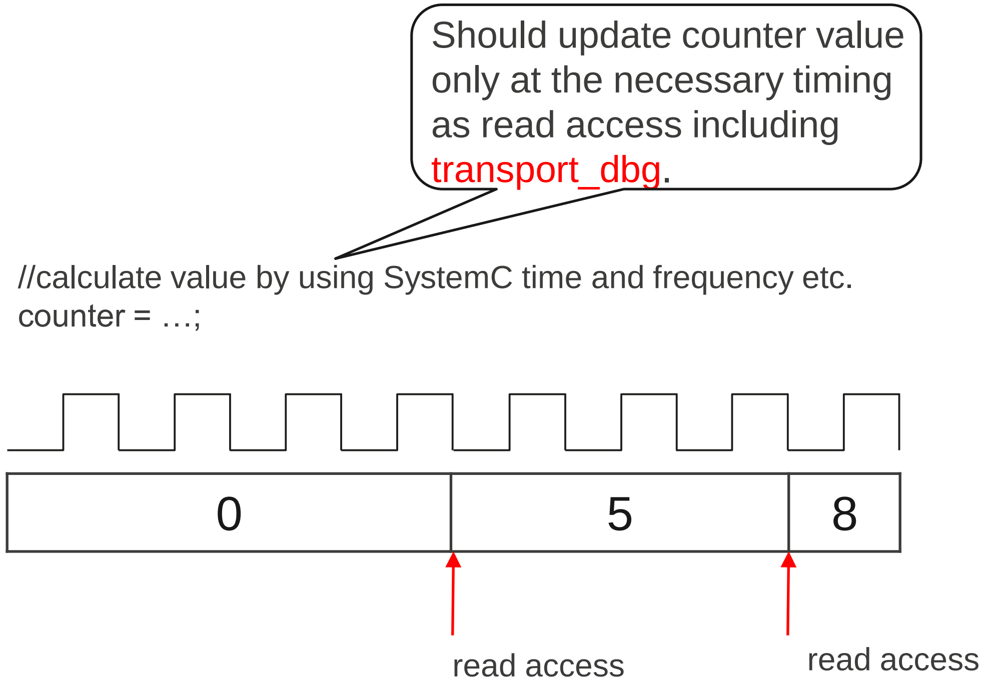
In the other cases, simulation should not be stopped and any messages are not output.

1.9. Minimize the invoke of process

- The invoke of process shall be minimized for improving simulation speed. That is to say, the model shall be reduced the number of calling SystemC functions as possible.
  - For example, Timer Model

    In case of the model which have counter register, the counter register shall(should) be updated at necessary timing not with clock sync.




Should update counter value only at the necessary timing as read access including transport_dbg.
//calculate value by using SystemC time and frequency etc.
counter = …;
0
5
8
read access
read access

2. Model Interfaces

2.1. Signal (sc_signal)

The type of the signal drived by the processes more than 2 should be
`sc_signal< {data_type}, sc_core::SC_UNCHECKED_WRITERS >`.

* It may become obsolete to set the environment variable "SC_SIGNAL_WRITE_CHECK" to avoid the several driver error checked by Accellera SystemC.
"SC_SIGNAL_WRITE_CHECK" is not prescribed by IEEE-1666-2011 either.

2.2. Bus Interface

Refer to “SC-HEAP : Bus I/F outline(Appendix of this Modeling Guideline)”

2.3. Pin

Peripheral IP

- All of the pins except the followings are “sc_logic”*
    - Interrupt signals are “bool”
    - Clock is “sc_dt::uint64”
    - Reset is “bool”
    - All DMAC ports are “bool”
    - The port connected with above ports are “bool”

* : sc_logic is required by some Tool vendor

Platfrom subsystem inside

- “bool”

The input port that behavior is not defined at Hi-Z input in the SDM, treats Hi-Z input as Low input.

2.4. Interrupt Request Signals

- Pin I/F same as RTL
- The data type is “bool”.

2.5. Peripheral Registers

Memory mapped registers in peripheral models should be visible on software debugger with “transport_dbg”. Debug access rule is as follows.

[rule of debug access]

- Write access
  - If the unacceptable value is written to register, it’s not written and the error message doesn’t display.
  - Cannot write in reserved bits and the error message doesn’t display.
  - Cannot write in read-only-bit and the error message doesn’t display.
- Process invoked by access
  - Write debug access invokes same process which is invoked by normal write access.
  - Basically, read debug access doesn’t invoke the process which is invoked by normal read access. Exceptionally, for read-modify-write register, read debug access invokes same process which is invoked by normal read access, since the write access is controlled by the user. (i.e. if the user doesn't write it, any process doesn’t invoked)
- Access depending on state
  - If the accessible condition for the register is varied by each internal state (mode), the process invoked by debug access is same as by normal access.
  - During reset, read debug access can be done but write debug access is ignored.
- Accessed address
  - Can access any address, but if accessed address includes unassigned address, debug read returns 0 and debug write is ignored.
- Accessed size
  - Can access with 1, 2 or 4 bytes debug access.

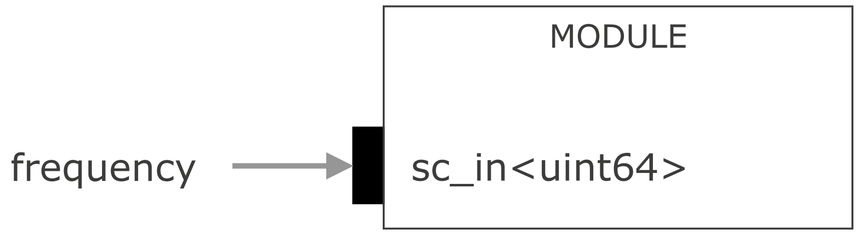
*** The red letter parts are the accesses which does not obey HW specification.**

2.6. Supply and Ground

- GND is not supported.
- Basically, each module doesn’t support the behavior at power supply variation.
- PRCCM(Power Reset Clock Control Module) and VCOMP(Voltage Comparator) support the behavior at power supply variation. Because the behavior is a main function for these modules. (The functions to generate interrupt at power supply variation, to indicate power voltage value on register and so on, must be supported.)

2.7. Clock

- Clock generator is implemented.
- Each module has clock input port that data type is “sc_dt::uint64”.
- The modules expect clock input from Clock generator in frequency[Hz].



- If clock related to operation is 0, the operation is stopped.
- If clock related to register access is 0, behavior at bus transaction reception is at below:

	Normal transaction (b_transport/nb_transport)	Debug transaction *1	Debug transaction *2
acceptance	accepted	accepted	accepted
error message	no	no	no
read operation	ignored	ignored	operated
write operation	ignored	ignored	ignored

*1 : It is transferred before nb_transport when CAISS executes instruction which causes memory access.
*2 : It is transferred when user accesses to memory using software debugger.

2.8. Reset

- Pin I/F same as HW. (The data type is “bool”)
- Model behavior during the reset:
  - Basically, the behavior shall be same as HW.
  - About the function which is not specified in the HW specification:
    - The behavior at bus transaction reception is as follows:

	Normal transaction (b_transport/nb_transport)	Debug transaction *1	Debug transaction *2
acceptance	accepted	accepted	accepted
error message	no	no	no
read operation	out of spec*3	out of spec*3	operated
write operation	out of spec*3	out of spec*3	ignored

*1 : It is transferred before nb_transport when CAISS executes instruction which causes memory access.
*2 : It is transferred when user accesses to memory using software debugger.
*3 : It is out of spec. It had better to be ignored.

    - The functions which are invoked by reset port are operated.
    - The functions which are not invoked by reset port are never operated.

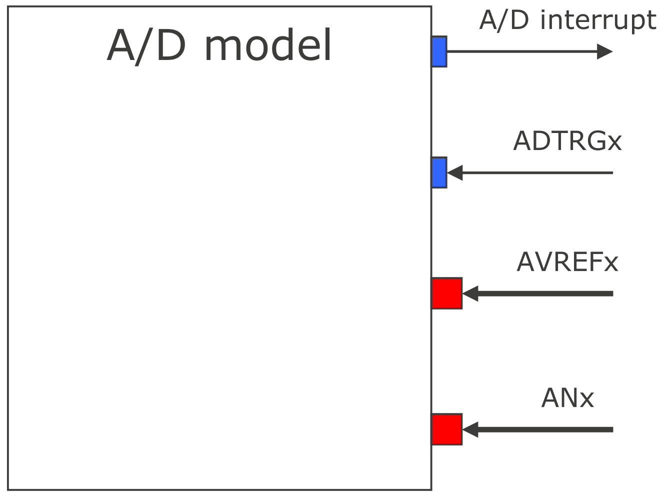
2.9. I/O Port

- sc_logic*

* : sc_logic is required by ASTC
(Australian Semiconductor Technology Company)

2.10. A/D Interface

- A/D model reads value of ANx when request is input with ATRG or start bit of the register.

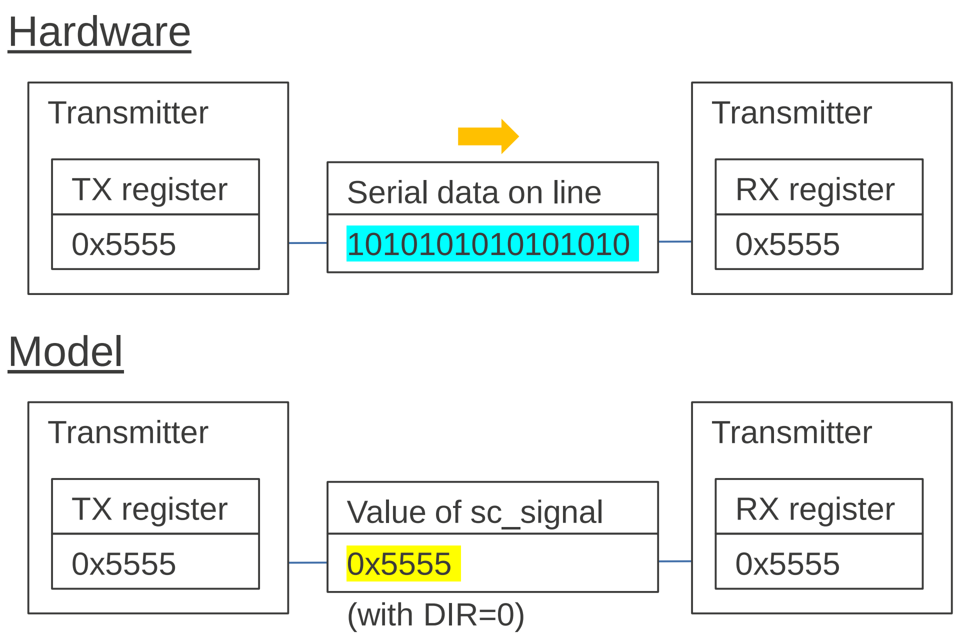


● ANx	: analog input port. sc_unit32.
● AVREFx	: reference port. sc_uint32. Value is fixed at simulation start.
● ADTRGn	: external trigger port. sc_logic.
● ADENDx	: A/D interrupts. bool.

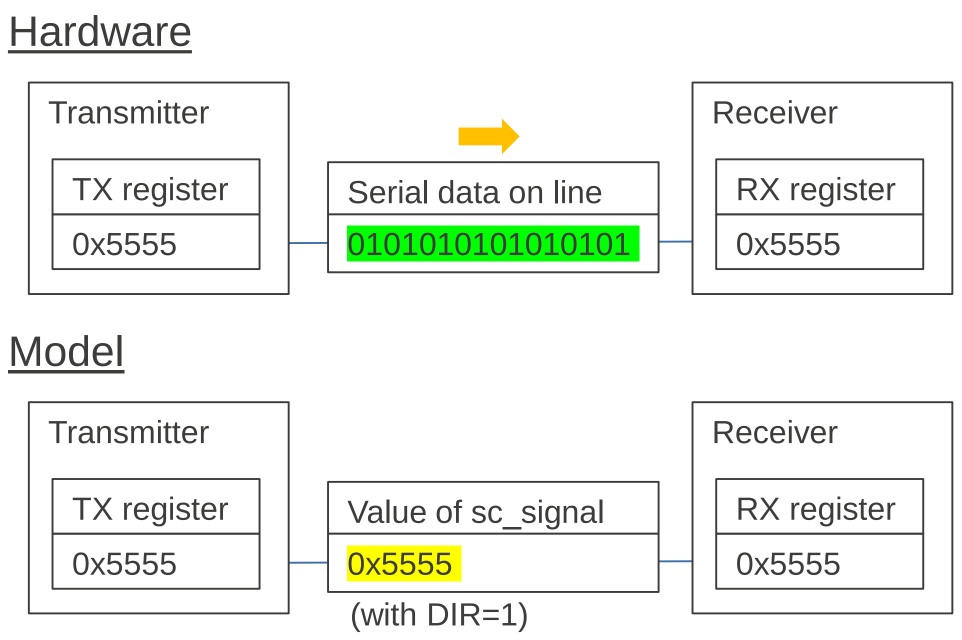
2.11. Common requirement for serial communication I/F(data packing)

- Basically, when serial communication data is abstracted into token such as “unsigned int”, the member of payload and so on, the bit-order (LSB first/MSB first) shall not be considered at data packing.
- If the bit-order information should be transferred between transmitter and receiver, another information shall be used like DIR(direction) bit of CONTROL port in SPI.(refer to 2.11)

Example for LSB first



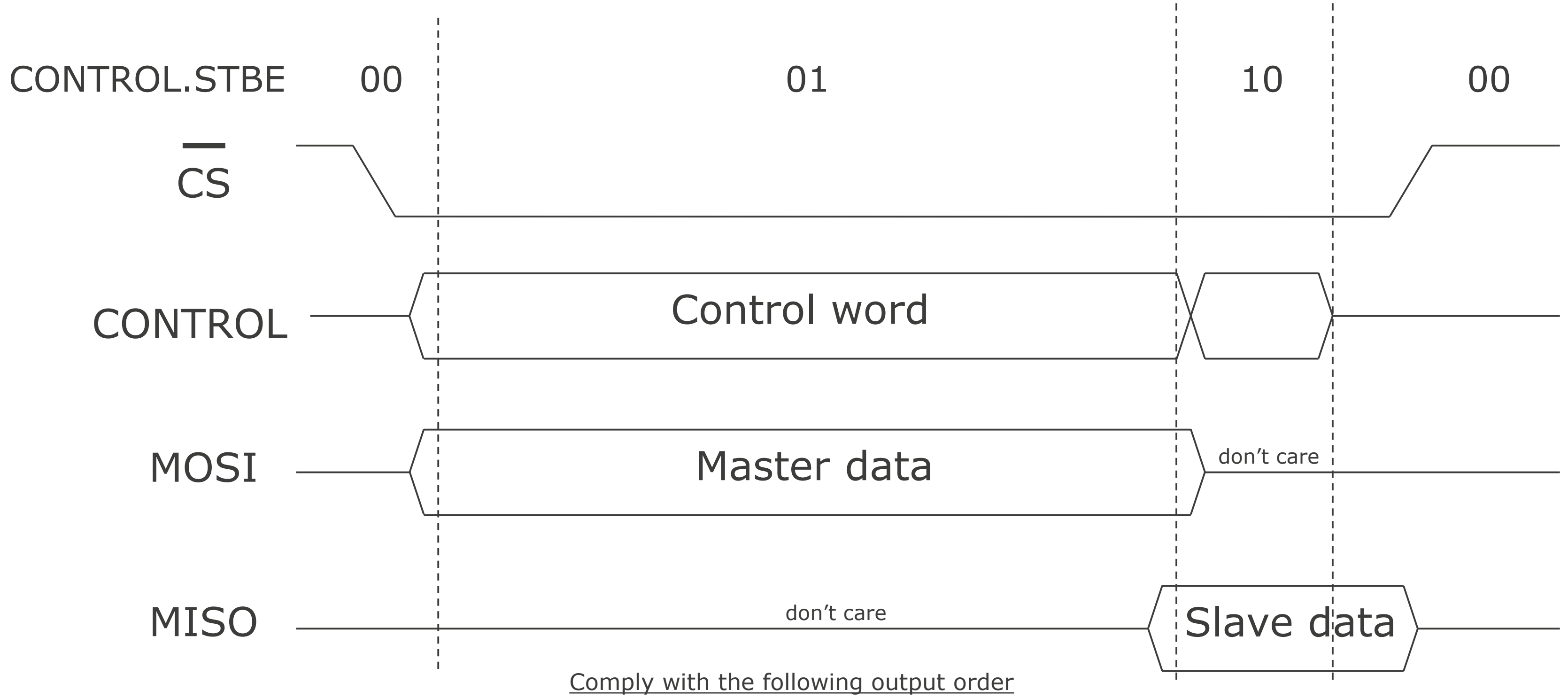
Example for MSB first



Background of the requirement : In the previous model development, there was a case of the bug which the bit-order was inverted between TX register and RX register because the bit-order was considered only on the transmitter side.

2.12. SPI

- Serial data is send in parallel data port that data width is 32 bits. (MISO, MOSI or SDI, SDO)
- Transfer start is expressed with CONTROL.STBE=01, and transfer end is expressed with 10.



Comply with the following output order

1. Asserting CS and Master Data
2. CONTROL.STBE = 01
3. Slave Data
4. CONTROL.STBE = 10
5. Negating CS

2.12. SPI (Explanation of CONTROL port)

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	CSTBE1	CSTBE0
Default	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	SPE	MSTR	Reserved	Reserved	Reserved	CPOL	CPHA	DIR	STBE1	STBE0	SIZE5	SIZE4	SIZE3	SIZE2	SIZE1	SIZE0
Default	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- CSTBE[1:0]: CLK strobe (Not used for SC-HEAP_E3)
- SPE: SPI enable (not used for SC-HEAP_E3)
- MSTR: Master/slave mode select
  - 0 = slave, 1 = master
- CPOL: Clock polarity
  - 0 = High active clock, 1 = Low active clock
- CPHA : Clock phase
  - 0 = data is sampled on 1st clock edge, shifted on 2nd
  - 1 = data is shifted on 1st clock edge, sampled on 2nd
  - See http://handyboard.com/oldhb/techdocs/moto-6811-techref.pdf
- DIR : Direction
  - 0 = MSB first, 1 = LSB first
- STBE[1:0]: Data strobe
  - Used to time and synchronize transfer cycle
  - 00=Idle, 01=Capture first bit, 10=Capture last bit, 11=Transmission aborted
- SIZE[5:0]: Data size
  - Direct, un-encoded value of data bits (0b001000 = 8 data bits)

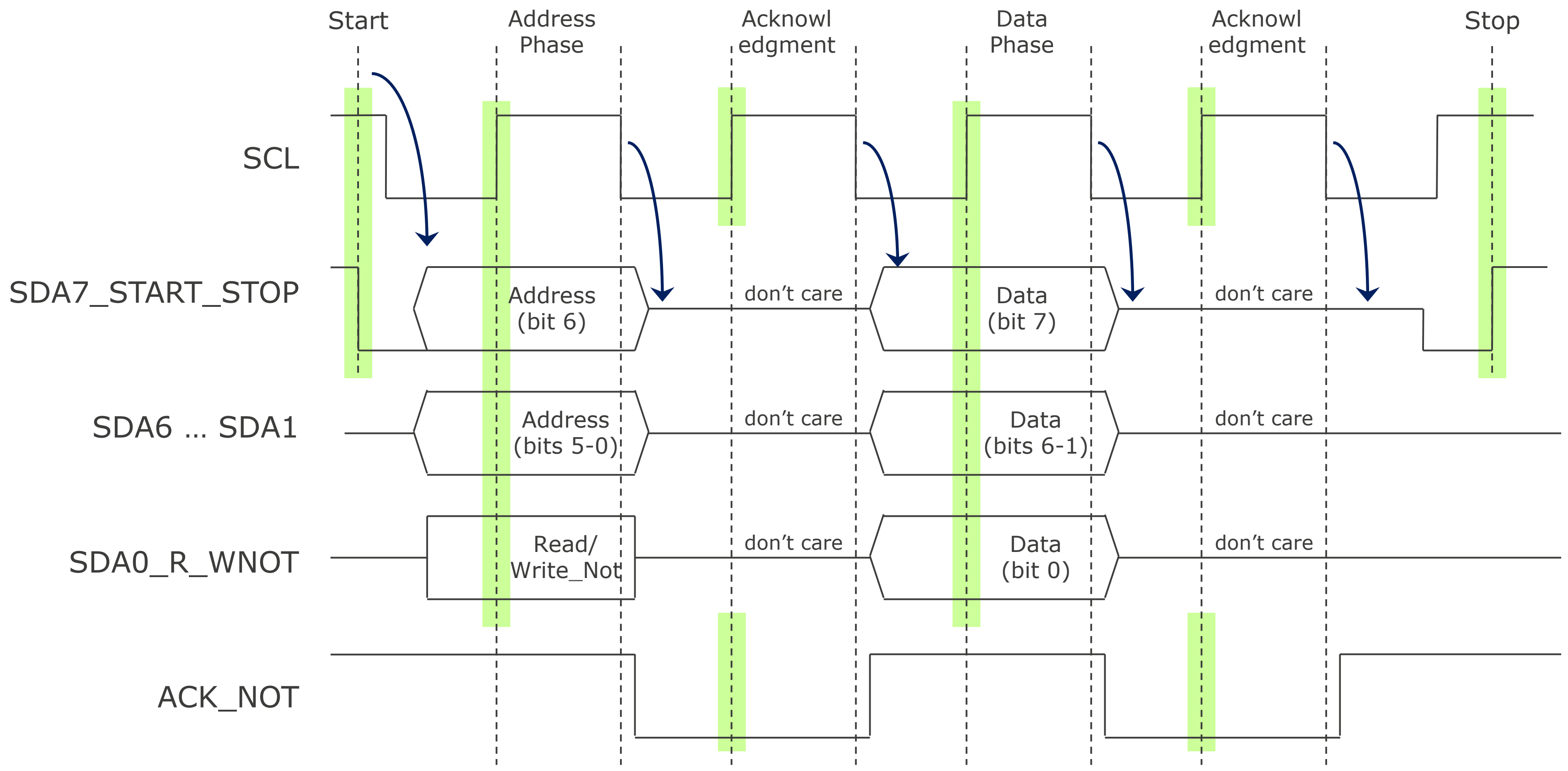
2.12. SPI (data type of the ports)

- Data type of MISO/MOSI(SDI/SDO) and CONTROL is “sc_logic[32]”*.

* : sc_logic is required by ASTC
(Australian Semiconductor Technology Company)

2.13. I2C

- SDA(data bit + start/stop bit + R/W) is parallel port(sc_logic x 8).
- SCL outputs clock duration for transfer.
- Arbitration is not supported.

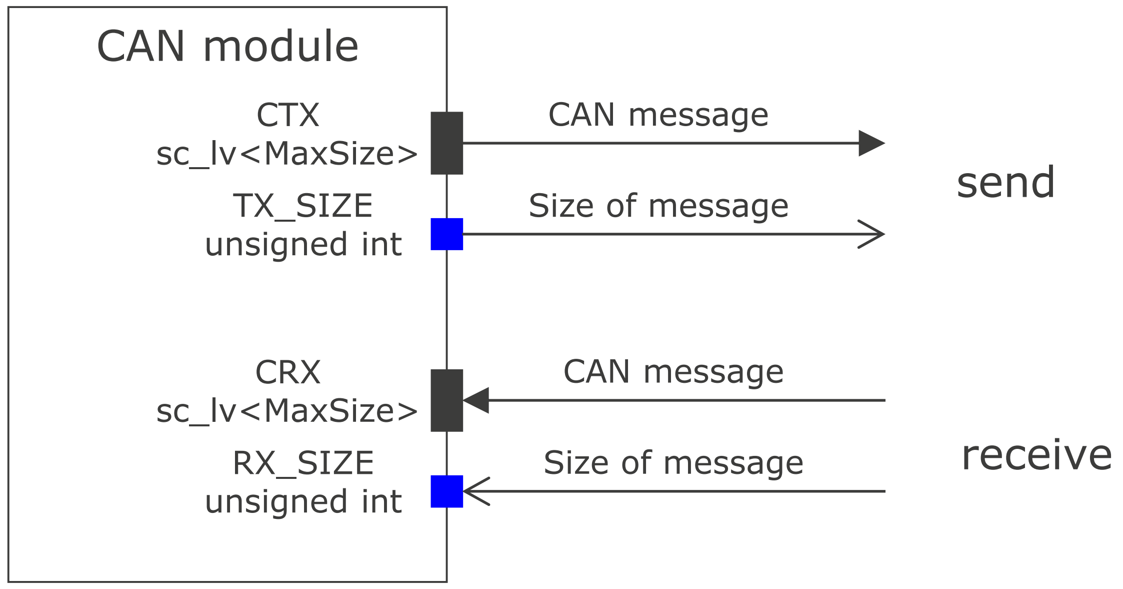


TBD

2.14. CAN

- Simulation speed shall be increased using parallel port I/F or TLM I/F of CAN ports. (T.B.D.)

<u>Image of parallel port I/F</u>

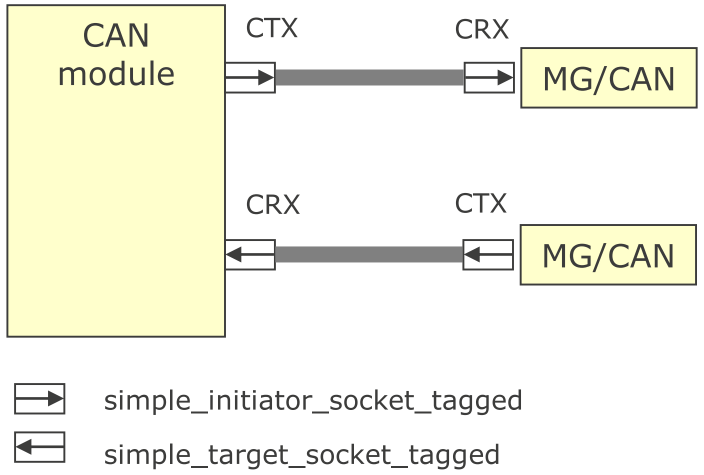


TBD

2.14. CAN

Image of TLM I/F

Use the OSCI TLM 2.0. Only LT coding style is supported.



```
plant.CTX[0]( MG/CAN.CRX[0]);
MG/CAN.CTX[0]( plant.CRX[0]);
```

Use original payload

```
enum eRTRtype {
    emDataFrame = 0,
    emRemoteFrame = 1
 };
enum eIDEtype {
    emNormal = 0,
    emExtended = 1
 };
struct CANPayload {
    int          SID;
    int          EID;
    unsigned char data[8];
    enum eRTRtype RTR;
    int          DLC;
    enum eIDEtype IDE;
};
```

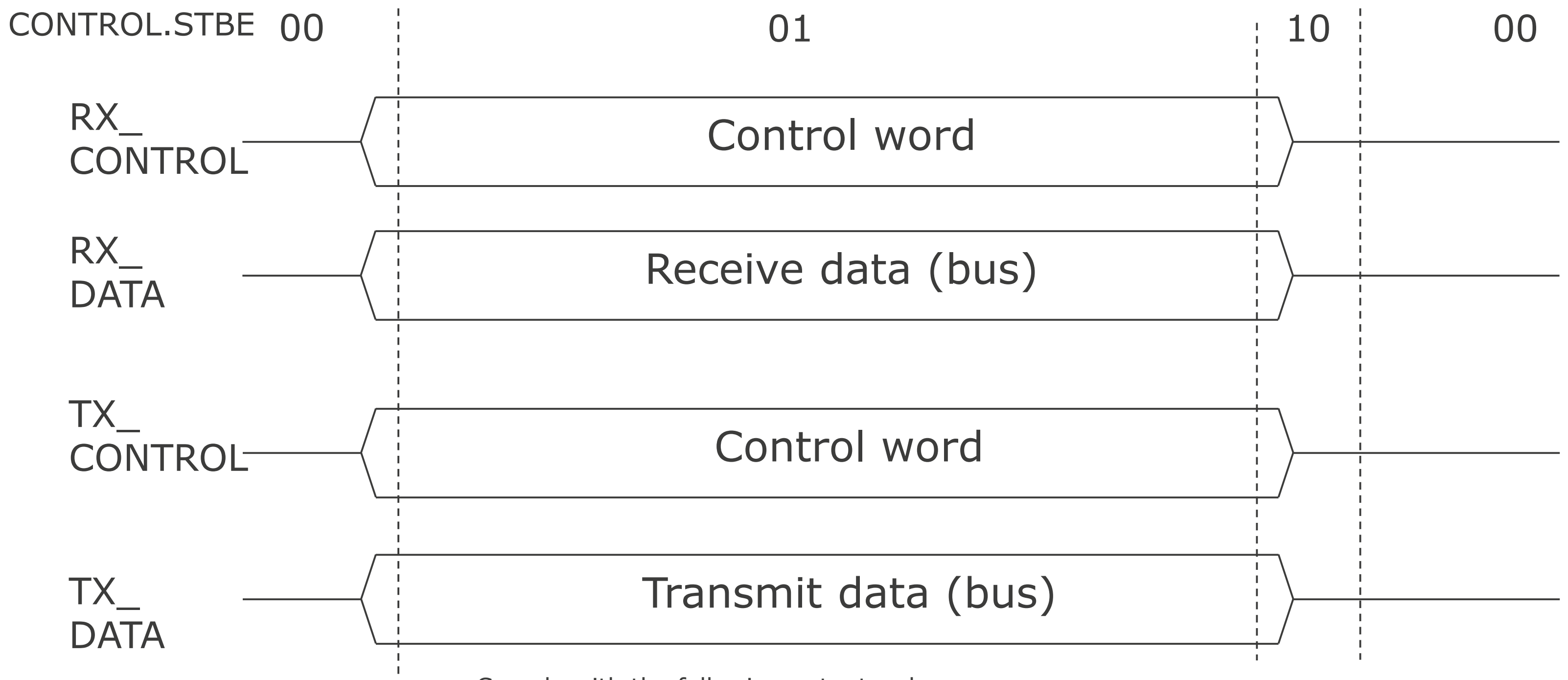
TBD

2.15. FlexRay

- Simulation speed is increased using TLM I/F, etc. (T.B.D.)

2.16. UART and LIN

- Use the control line(TX_CONTROL/RX_CONTROL) and the data line(TX_DATA/RX_DATA) similar as SPI. (parallel port I/F)
- In addition, it is used a pin (interrupt signals, etc) peculiar to IP
- Slave RX lines are connected to Master TX lines, and vice versa



Comply with the following output order

1. Data
2. CONTROL.STBE = 01
3. CONTROL.STBE = 10 (Data should be kept to here)

2.16. UART and LIN (TX/RX_CONTROL

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	BITT15	BITT14	BITT13	BITT12	BITT11	BITT10	BITT9	BITT8	BITT7	BITT6	BITT5	BITT4	BITT3	BITT2	BITT1	BITT0
Default	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TE	TC	Reserved	NUM3	NUM2	NUM1	NUM0	DIR	STBE1	STBE0	Reserved	Reserved	SIZE3	SIZE2	SIZE1	SIZE0
Default	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- BITT[15:0] : Bit timing
  - Tenths of microseconds per bit
  - Example: 4800bps -> 208.3us per bit -> 2083
- TE: For LIN. It is used to identify slave which is transmitting with reception multiplexer. (Not used for SC-HEAP_E3)
- TC: Transmit Complete
  - 0 = busy, 1 = completed
- NUM[3:0]: For LIN. The number of data bytes in data field of 1 LIN frame.
- DIR: Direction
  - 0 = MSB first, 1 = LSB first
- STBE[1:0]: Data strobe
  - Used to time and synchronize transfer cycle
  - 00=Idle, 01=Start bit, 10=Stop bit, 11=Transmission aborted
- SIZE[3:0]: Data bit size in a data
  - Direct, unencoded value of data bits + parity bits
  - Example: 0b001000 = 8 data bits, or 7 data bits + 1 parity bit

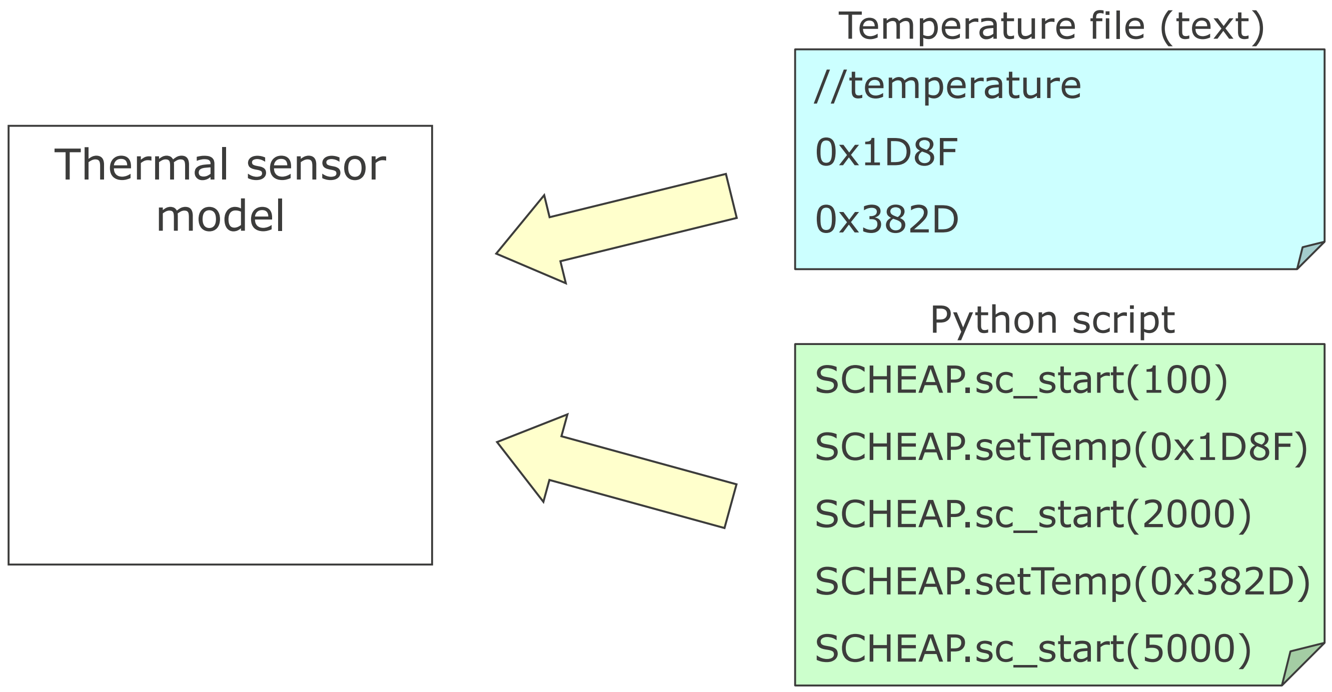
TBD

2.17. Ethernet

- Simulation speed is increased using TLM I/F, etc. (T.B.D.)

2.18. Thermal Sensor Interface

- Temperature is provided by user through a text file or Python I/F.
- In case of text file, the model reads the temperature data by 1 line from the text file at the request reception with signal or start bit of register.
- In case of Python I/F, user can set the temperature data in any timing from Python script or console.

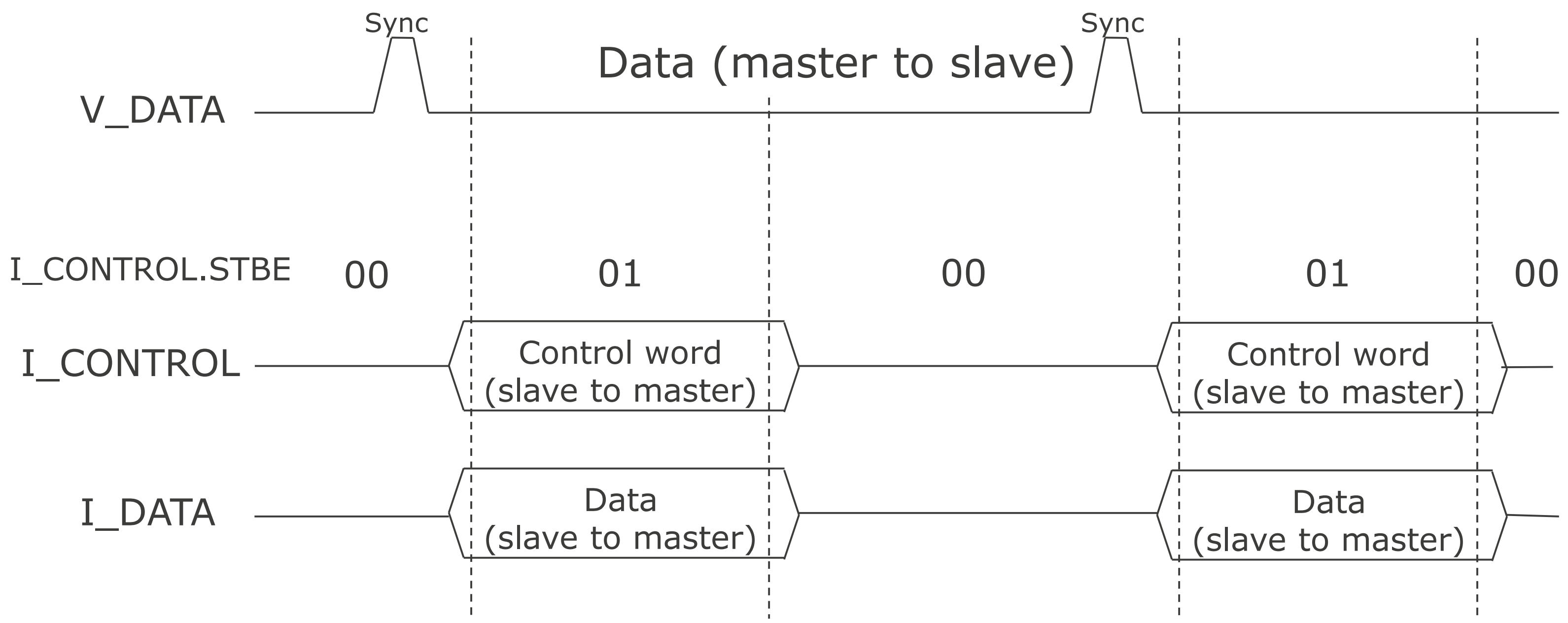


```
//temperature
0x1D8F
0x382D
```

```
SCHEAP.sc_start(100)
SCHEAP.setTemp(0x1D8F)
SCHEAP.sc_start(2000)
SCHEAP.setTemp(0x382D)
SCHEAP.sc_start(5000)
```

2.19. PSI5

- Use only the data line(V_DATA) of bool type I/F for data sending from master to slave.
- Use the control line(I_CONTROL) and the data line(I_DATA) similar as UART for data sending from slave to master. (parallel port I/F. Data type of I_COMTROL and I_DATA is unsigned int.)



<u>Comply with the following output order for Slave-to-master-communication</u>

1. Data
2. I_CONTROL.STBE = 01
3. I_CONTROL.STBE = 00 (Data of I_DATA should be kept to here)

2.19. PSI5 (I_CONTROL)

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	BITT15	BITT14	BITT13	BITT12	BITT11	BITT10	BITT9	BITT8	BITT7	BITT6	BITT5	BITT4	BITT3	BITT2	BITT1	BITT0
Default	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

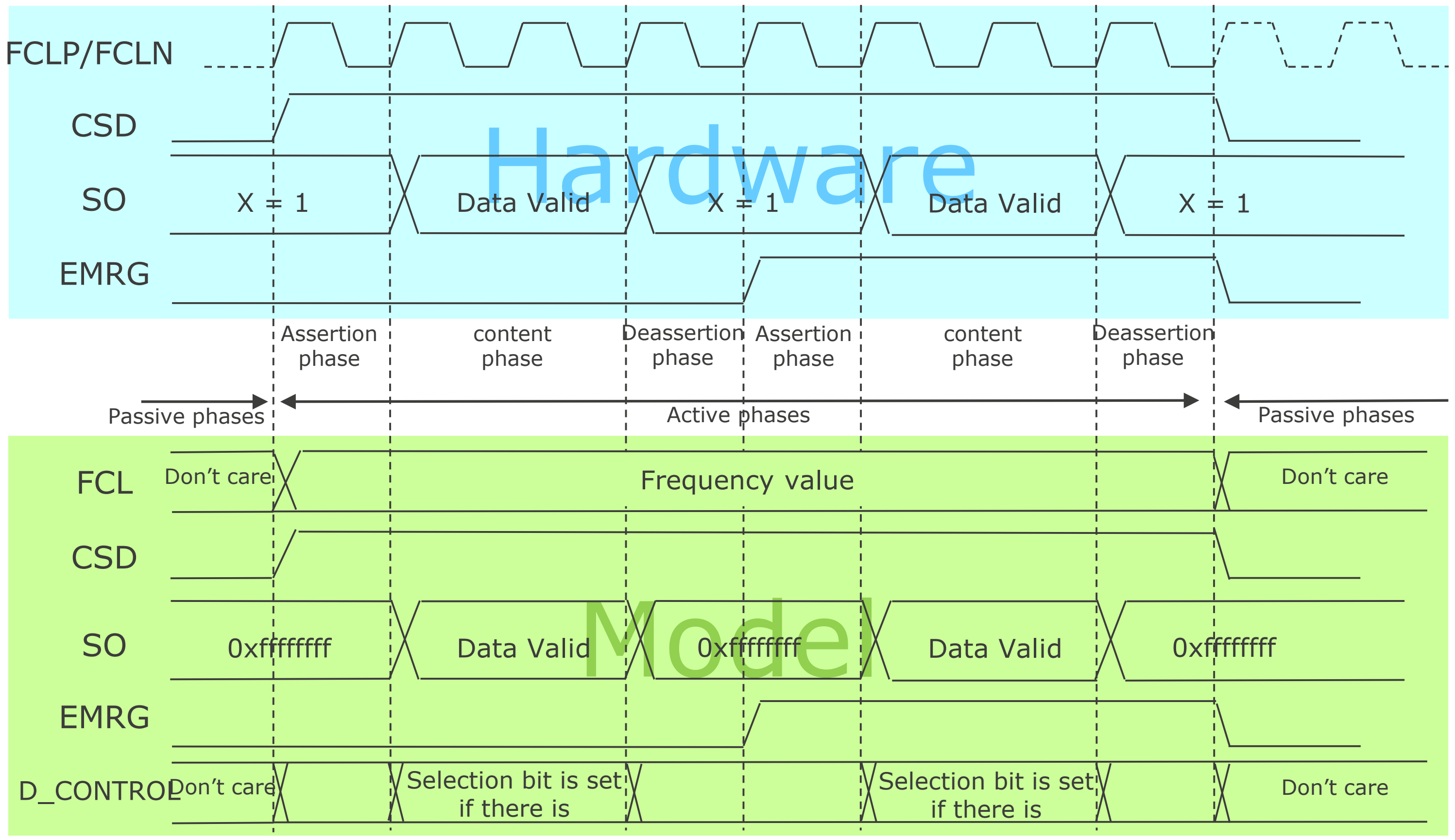
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	DIR	STBE1	STBE0	SIZE5	SIZE4	SIZE3	SIZE2	SIZE1	SIZE0
Default	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- BITT[15:0] : Bit timing
  - Tenths of nanoseconds per bit
  - Example: 125kbps -> 8000ns per bit -> 8000
- DIR: Direction
  - 0 = MSB first, 1 = LSB first
- STBE[1:0]: Data strobe
  - Used to time and synchronize transfer cycle
- 00=Idle, 01=Start bit, 11=Transmission aborted (stop bit is not supported in PSI5)
- SIZE[5:0]: Data bit size in a data
  - Value of data bits + parity bits(or CRC bits). Not include start bits.
  - Example: 0b001101 = 10 data bits + 3 CRC bits, or 12 data bits + 1 parity bit

2.20. RHSB* (Downstream communication)

* : Renesas High Speed Bus

- Serial data is sent in parallel data port whose data width is 64 bits.(SO)
- Chip select and EMRG are bool pins same as HW.(CSD and EMRG)
- Clock port is “sc_dt::uint64” same as 2.7. Clock.(FCL)
- Selection bit is set in CONTROL port during content phase if configured.(CONTROL)



Comply with the following output order for Down stream communication

1. About SO, Data Valid includes the selection bit as duration but not it as value.
2. Slave shall recognize data as valid after waiting for “Assertion phase” + “selection bit”.
3. The inversion configuration does not affect FCL and SO. Because, SO is abstracted.

2.20. RHSB* (CONTROL port in Downstream)

* : Renesas High Speed Bus

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
Default	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	CLP	SCLP	SIZE6	SIZE5	SIZE4	SIZE3	SIZE2	SIZE1	SIZE0	SLCT
Default	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SLCT: selection bit

SIZE: data/command bit size

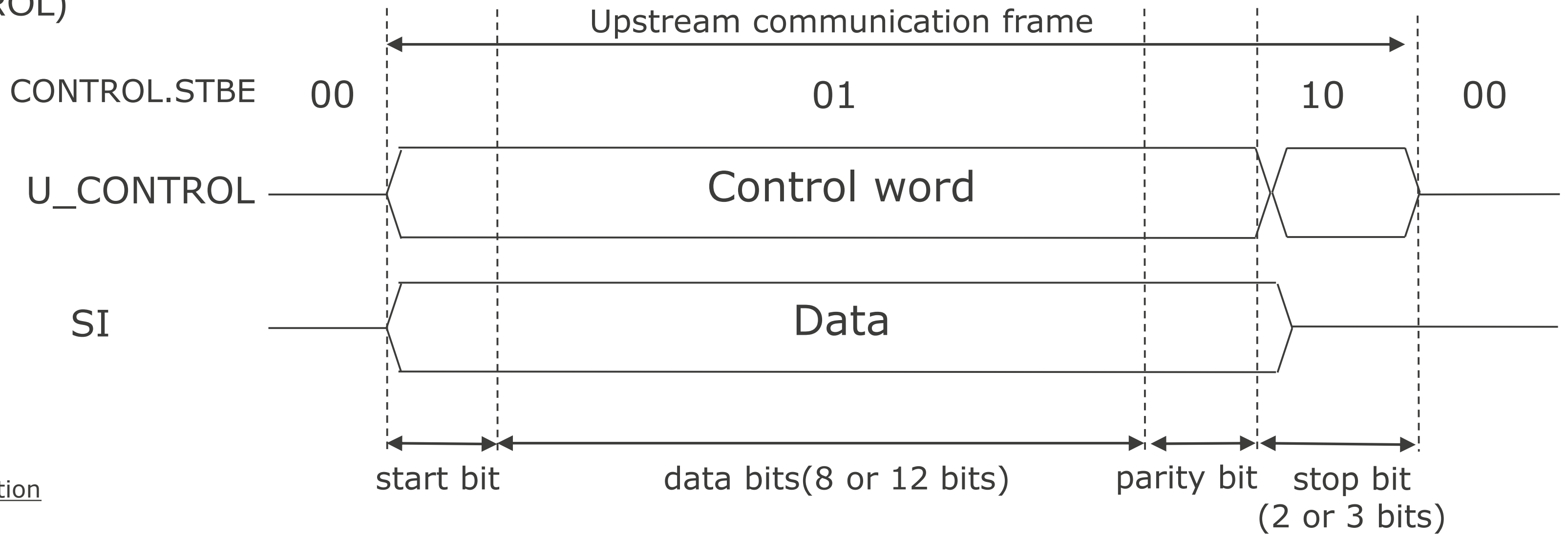
SOLP: Serial Out Line Polarity (reflex of RHSBjSDCi.SOLPn bit)

CLP: Clock Line Phase (reflex of RHSBjDCR.CLP bit)

2.20. RHSB* (Upstream communication)

* : Renesas High Speed Bus

- Serial data is sent in parallel data port whose width is 32 bits.(SI)
- Use CONTROL port similar as UART.(CONTROL)



<u>Comply with the following output order for Upstream communication</u>

1. Data includes parity bit
2. CONTROL.STBE = 01 (start the communication)
3. CONTROL.STBE = 10 (Data should be kept to here)

2.20. RHSB* (CONTROL port in Upstream)

* : Renesas High Speed Bus

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	BITT15	BITT14	BITT13	BITT12	BITT11	BITT10	BITT9	BITT8	BITT7	BITT6	BITT5	BITT4	BITT3	BITT2	BITT1	BITT0
Default	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	STBE1	STBE0	Reserved	Reserved	SIZE3	SIZE2	SIZE1	SIZE0
Default	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BITT[15:0] : Bit timing

- nanoseconds per bit
- Example: 8MHz -> 125ns per bit -> 125

STBE[1:0]: Data strobe

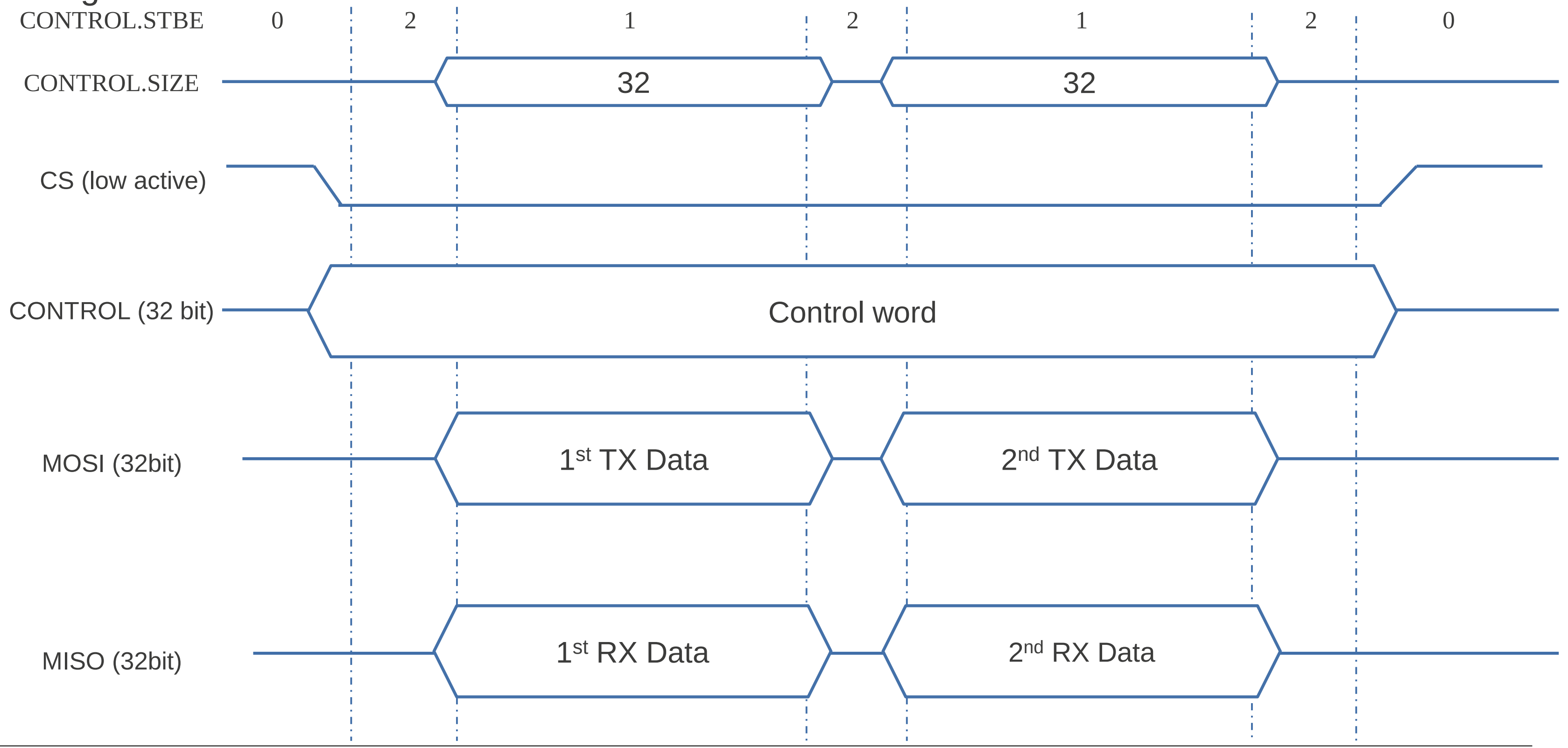
- Used to time and synchronize transfer cycle
- 00=Idle, 01=Start bit, 10=Stop bit, 11=Transmission aborted

SIZE[3:0]: Data bit size in a data

- Direct, un-encoded value of data bits + parity bits
- Example: 0b001001 = 8 data bits + 1 parity bit

2.21 SPI2

- Serial data is sent parallel via 32 bits data ports, i.e. MISO, MOSI or SDI, SDO.
- Data communication is expressed with CONTROL.STBE = 1, and wait time is expressed with STBE = 2.
- Transaction is frame-based, and full-duplex.
  - Frame length ranges from 32 to 128 bits
  - The CONTROL.SIZE is fixed during a frame transfer.



Transmit & receive mode
MSPInTXEm = 1, MSPInRXEm = 1: Tx/Rx
MSPInFLENm = 20H: frame length is 32 bits
MSPInCFSETm = 0002H: frame count is 2 times

Compling with the following output order:

1. Asserting CS
2. STBE = 2 (setup time)
3. Asserting 1st TX & RX data
4. STBE = 1 (communication)
5. STBE = 2 (inter-data time)
6. Asserting 2nd TX & RX data
7. STBE = 1 (communication)
8. Repeating step 5, 6, & 7 if frame count is greater than 2
9. STBE = 2 (hold time)
10. Negating CS

2.21 SPI2 (Explanation of CONTROL port)

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	~~CSTBE1~~	~~CSTBE0~~
Default	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	~~SPE~~	~~MSTR~~	SAMP	CPOL	CPHA	DIR	STBE1	STBE0	SIZE7	SIZE6	SIZE5	SIZE4	SIZE3	SIZE2	SIZE1	SIZE0
Default	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- CSTBE[1:0]: CLK strobe (don’t use for U2A/MSPI)
- SPE: SPI enable (don’t use for U2A/MSPI)
- MSTR: master/slave mode select (don’t use for U2A/MSPI as master & slave data are conveyed simultaneously)
- SAMP: This bit controls the internal sampling timing for receive data in the master mode.
  - 0: The sampling timing of reception is standard sampling point of SPI protocol
  - 1: The sampling timing of reception is next edge sampling point of SPI protocol
- CPOL: clock polarity
  - 0 = high active clock; 1 = low active clock
- CPHA: clock phase (see http://handyboard.com/oldhb/techdocs/moto-6811-techref.pdf)
  - 0 = data is sampled on 1st clock edge, shifted on 2nd
  - 1 = data is shifted on 1st clock edge, sampled on 2nd
- DIR: direction
  - 0 = MSB first; 1 = LSB first
- STBE[1:0]: Data strobe
  - Used to time and synchronize transfer cycle
  - 0 Idle; 1 Communication; 2 Setup/inter-data/hold/wait time; 3 Transmission aborted
- SIZE[7:0]: Data size
  - Direct, un-encoded value of data bits (e.g. 0b1000_0000 = 128 data bits)

2.21 SPI2 (Data type of the CONTROL ports)

- Data type of MISO/MOSI (SDI/SDO) and CONTROL is “sc_uint<32>”

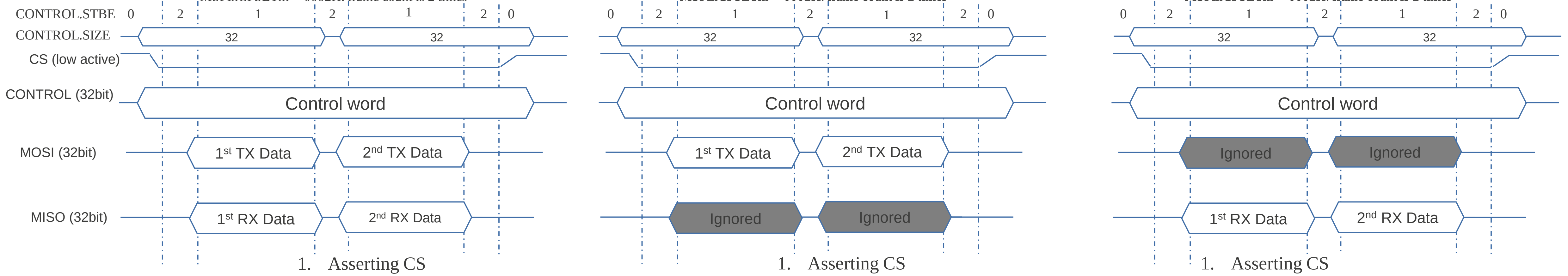
2.21 SPI2 (Examples with transfer size <= 32)

- U2A/MSPI has 3 transaction modes:
  - Transmit & receive mode
  - Transmit-only mode
  - Receive-only mode

(1) Transmit & receive mode
MSPInTXEm = 1, MSPInRXEm = 1: Tx/Rx
MSPInFLENm = 20H: frame length is 32 bits
MSPInCFSETm = 0002H: frame count is 2 times

(2) Transmit-only mode
MSPInTXEm = 1, MSPInRXEm = 0: Tx
MSPInFLENm = 20H: frame length is 32 bits
MSPInCFSETm = 0002H: frame count is 2 times

(3) Receive-only mode
MSPInTXEm = 0, MSPInRXEm = 1: Rx
MSPInFLENm = 20H: frame length is 32 bits
MSPInCFSETm = 0002H: frame count is 2 times



(1)
1. Asserting CS
2. STBE = 2 (setup time)
3. Asserting 1st TX & RX data
4. STBE = 1 (communication)
5. STBE = 2 (inter-data time)
6. Asserting 2nd TX & RX data
7. STBE = 1 (communication)
8. STBE = 2 (hold time)
9. Negating CS

(2)
1. Asserting CS
2. STBE = 2 (setup time)
3. Asserting 1st TX data
4. STBE = 1 (communication)
5. STBE = 2 (inter-data time)
6. Asserting 2nd TX data
7. STBE = 1 (communication)
8. STBE = 2 (hold time)
9. Negating CS

(3)
1. Asserting CS
2. STBE = 2 (setup time)
3. Asserting 1st RX data
4. STBE = 1 (communication)
5. STBE = 2 (inter-data time)
6. Asserting 2nd RX data
7. STBE = 1 (communication)
8. STBE = 2 (hold time)
9. Negating CS

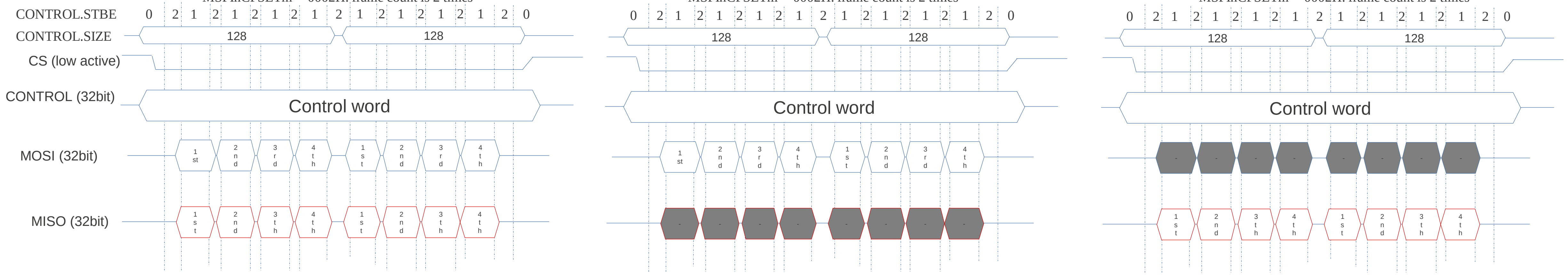
2.21 SPI2 (Examples with transfer size > 32)

- For the case big transfer size, i.e. greater than 32, the transfer is split into many 32-bit ones.
  - E.g. 128 bits transfer = 4 x 32 bits transfer
  - E.g. 127 bits transfer = 3 x 32 bits transfer + 1 x 31 bits transfer
  - E.g. 33 bits transfer = 1 x 32 bits transfer + 1 x 1 bits transfer

<u>(1) Transmit & receive mode</u>
MSPInTXEm = 1, MSPInRXEm = 1: Tx/Rx
MSPInFLENm = 80H: frame length is 128 bits
MSPInCFSETm = 0002H: frame count is 2 times

<u>(2) Transmit-only mode</u>
MSPInTXEm = 1, MSPInRXEm = 0: Tx
MSPInFLENm = 80H: frame length is 128 bits
MSPInCFSETm = 0002H: frame count is 2 times

<u>(3) Receive-only mode</u>
MSPInTXEm = 0, MSPInRXEm = 1: Rx
MSPInFLENm = 80H: frame length is 128 bits
MSPInCFSETm = 0002H: frame count is 2 times



Notes:
: TX Data
: RX Data

(1)
1. Asserting CS
2. STBE = 2 (setup time)
   1. Asserting 1st 32bit TX & RX data
   2. STBE = 1 (communication)
   3. STBE = 2 (wait time)

   repeat 4 times for the 1st frame count
3. STBE = 2 (inter-data time)
   1. Asserting 1st 32bit TX & RX data
   2. STBE = 1 (communication)
   3. STBE = 2 (wait time)

   repeat 4 times for the 2nd frame count
4. STBE = 2 (hold time)
5. Negating CS

(2)
1. Asserting CS
2. STBE = 2 (setup time)
   1. Asserting 1st 32bit TX data
   2. STBE = 1 (communication)
   3. STBE = 2 (wait time)

   repeat 4 times for the 1st frame count
3. STBE = 2 (inter-data time)
   1. Asserting 1st 32bit TX data
   2. STBE = 1 (communication)
   3. STBE = 2 (wait time)

   repeat 4 times for the 2nd frame count
4. STBE = 2 (hold time)
5. Negating CS

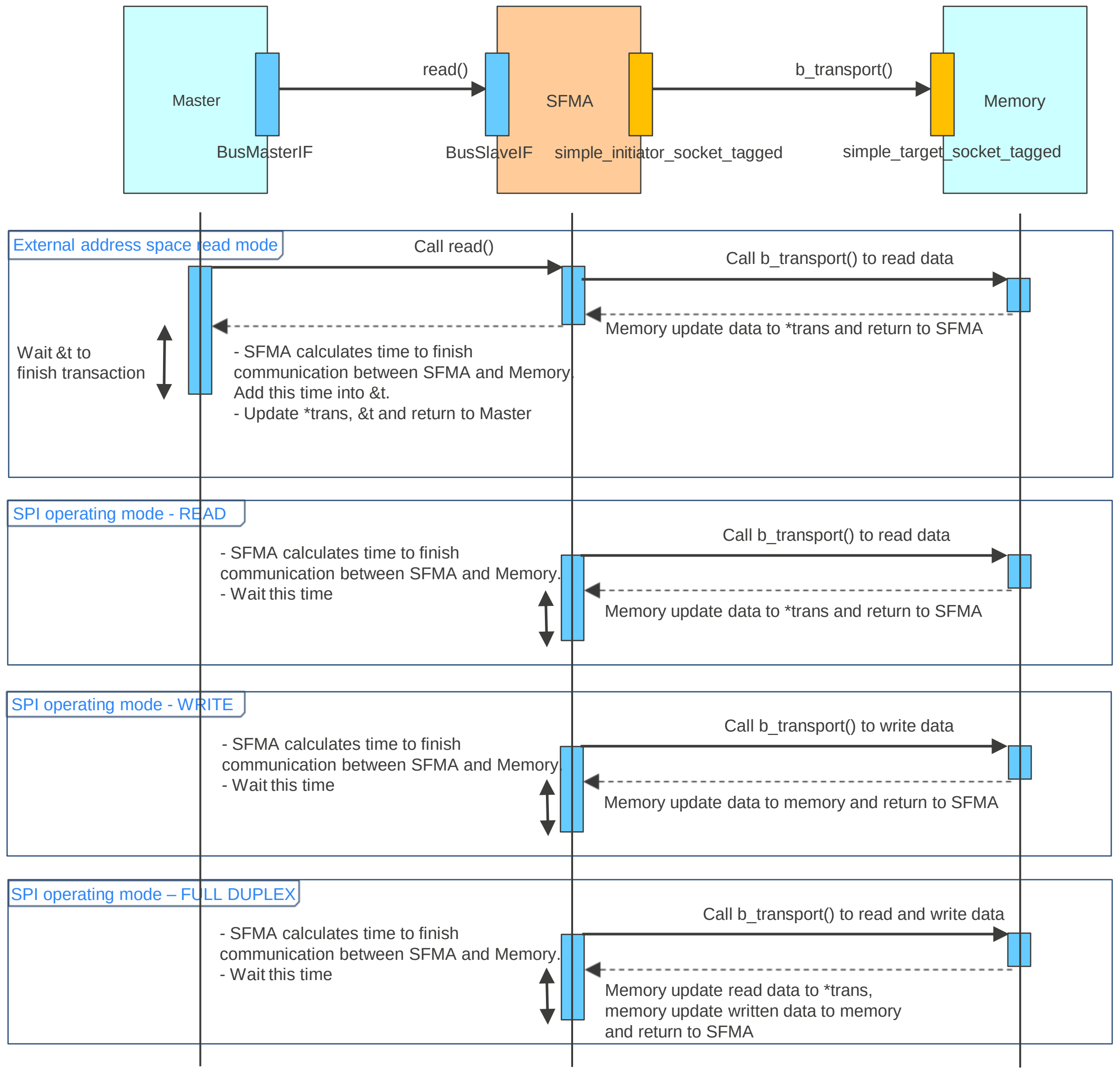
(3)
1. Asserting CS
2. STBE = 2 (setup time)
   1. Asserting 1st 32bit RX data
   2. STBE = 1 (communication)
   3. STBE = 2 (wait time)

   repeat 4 times for the 1st frame count
3. STBE = 2 (inter-data time)
   1. Asserting 1st 32bit RX data
   2. STBE = 1 (communication)
   3. STBE = 2 (wait time)

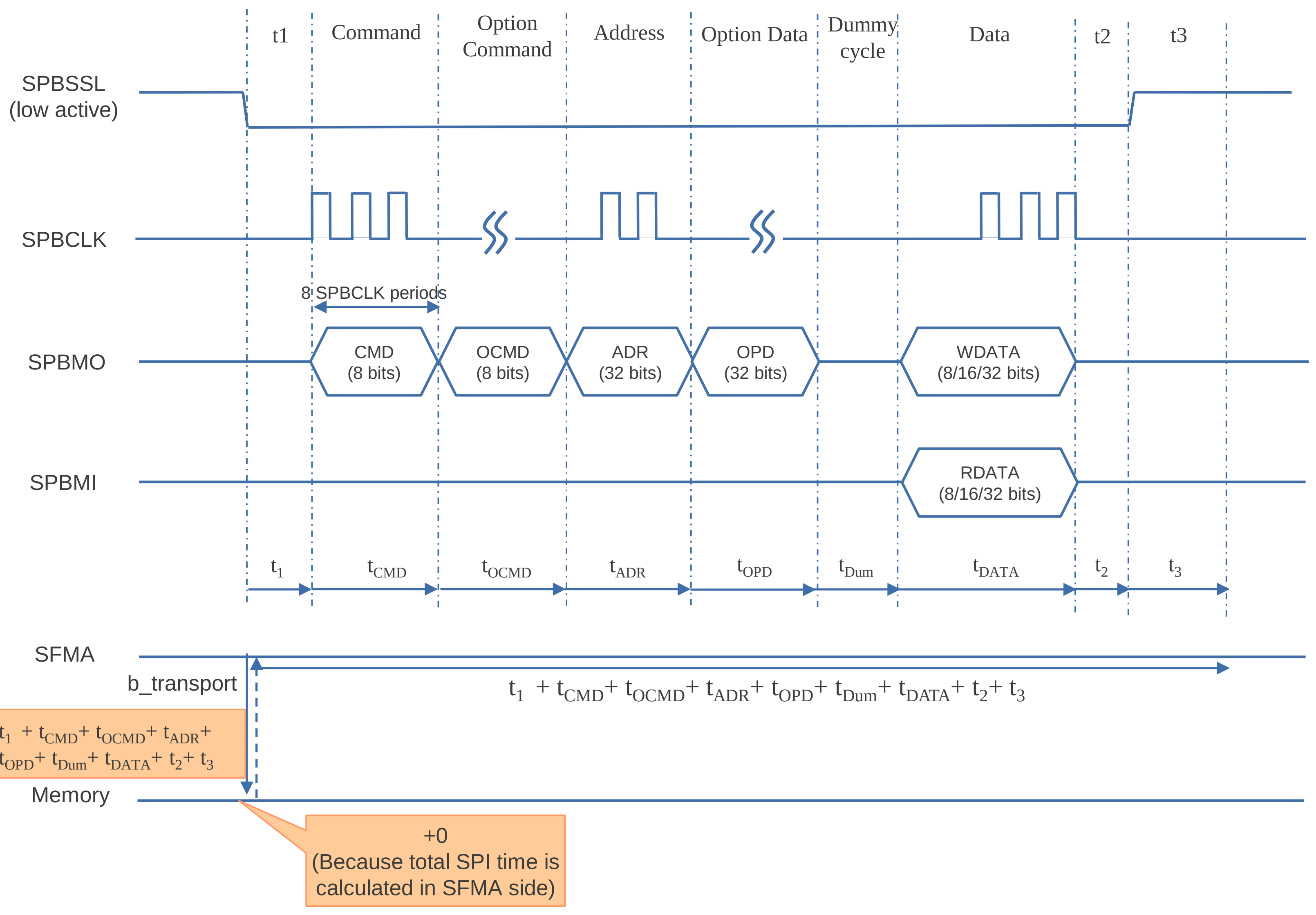
   repeat 4 times for the 2nd frame count
4. STBE = 2 (hold time)
5. Negating CS

2.22 SFMA (Communication with memory using socket)

- Interface between SFMA model and flash memory is using socket.
- Only LT coding style is in connection between SFMA and Memory. So, b_transport() is always called. (Not call nb_transport_fw())
- In full duplex, read data and write data are process same time.


Master
BusMasterIF
read()
BusSlaveIF
SFMA
simple_initiator_socket_tagged
b_transport()
simple_target_socket_tagged
Memory
External address space read mode
Call read()
Call b_transport() to read data
Memory update data to *trans and return to SFMA
Wait &t to
finish transaction
- SFMA calculates time to finish
communication between SFMA and Memory.
Add this time into &t.
- Update *trans, &t and return to Master
SPI operating mode - READ
Call b_transport() to read data
- SFMA calculates time to finish
communication between SFMA and Memory.
- Wait this time
Memory update data to *trans and return to SFMA
SPI operating mode - WRITE
Call b_transport() to write data
- SFMA calculates time to finish
communication between SFMA and Memory.
- Wait this time
Memory update data to memory and return to SFMA
SPI operating mode – FULL DUPLEX
Call b_transport() to read and write data
- SFMA calculates time to finish
communication between SFMA and Memory.
- Wait this time
Memory update read data to *trans,
memory update written data to memory
and return to SFMA

2.22 SFMA (Communicate time calculation)



2.22 SFMA (Communicate time calculation)

- The time unit used at following explanation is SPBCLK clock period.

1. Command transfer time if enable: t_{CMD} = 8 / command bus size
2. Option command transfer time if enable: t_{OCMD} = 8 / option command bus size
3. Address transfer time if enable:
   (3.1) If ADE[3] = 1, t_{ADR} = 32 / address bus size
   (3.2) If ADE[3] = 0, t_{ADR} = 24 / address bus size
4. Option data transfer time if enable:
   (4.1) If OPDE[3:0] = 0b1000, t_{OPD} = 8 / option data bus size
   (4.2) If OPDE[3:0] = 0b1100, t_{OPD} = 16 / option data bus size
   (4.3) If OPDE[3:0] = 0b1110, t_{OPD} = 24 / option data bus size
   (4.4) If OPDE[3:0] = 0b1111, t_{OPD} = 32 / option data bus size
5. Read/write data transfer time if enable: t_{DATA} = data size / data bus size
6. Delay t_1, t_2, t_3 and dummy cycle time (t_{Dum}) are described in register description

2.22 SFMA (Initiator socket transport data structure)

- **TLM transaction extension:** create a new TLM extension class (SFMA_TlmExtension) to send transaction command type, read/write data size and data field enable information to flash

1. **Read/write data size:** is set to SFMA TLM extension.
   Example: “mSfmaExtension->setReadWriteDataSize(size);”
2. **Transaction command type**
   (1.2.1) If both write data and read data are disable, write command type is set to extension.
   Example: “mSfmaExtension->setTransactionCommand(emTrans_Write_Command);”
   (1.2.2) If both write data and read data are enable, read_write command type is set to extension.
   Example: “mSfmaExtension->setTransactionCommand(emTrans_ReadWrite_Command);”
   (1.2.3) If only write data is enable, write command type is set to extension.
   Example: “mSfmaExtension->setTransactionCommand(emTrans_Write_Command);”
   (1.2.4) If only read data is enable, read command type is set to extension.
   Example: “mSfmaExtension->setTransactionCommand(emTrans_Read_Command);”

2.22 SFMA (Initiator socket transport data structure)

- **TLM transaction extension:** create a new TLM extension class (SFMA_TlmExtension) to send transaction command type, read/write data size and data field enable information to flash

3. **Data field enable:** contain 6 valid bits to reflect enable information in data pointer of command (bit 0), option command (bit 1), address (bit 2), option data (bit 3), write data (bit 4), read data (bit 5).

   Example: mSfmaExtension->setDataFieldEnable(0b10001);

Bit	7	6	5	4	3	2	1	0
	Reserved	Reserved	RDATA enable	WDATA enable	OPD enable	ADR enable	OCMD enable	CMD enable

4. **Extension information is set to TLM transaction.**

   Example: “tlmTrans.set_extension(mSfmaExtension);”

2.22 SFMA (Initiator socket transport data structure)

- **TLM data pointer:** has structure as following:

Bit	141 … … 14	13 12 11 10	9 8 7 6	5 4 3 2	1	0
	READ DATA (128 bytes)	WRITE DATA	OPD	ADR	OCMD	CMD

1. **Data length** is 142 bytes. If any data field is disable, the default value is 0.
2. **Command** (CMD): byte 0
3. **Option command** (OCMD): byte 1
4. **Address** (ADR) : byte 2 – 5
5. **Option data** (OPD): byte 6 – 9. Byte order is OPD3 at byte 9, OPD2 at byte 8, OPD1 at byte 7, OPD0 at byte 6. If any option data byte is disable, the option data is shifted to right.
   Example: if OPD0 and OPD1 are disable, OPD3 is at byte 7 and OPD2 is at byte 6.
6. **Write data** (WDATA): byte 10 – 13
7. **Read data** (RDATA): byte 14 – 141. Read data maximum size is 128 bytes in case Burst Read Operation. If SFMA transports read or read_write command type, flash memory will put read data to READ DATA position. SFMA model will read data from returned transaction from memory.
   Example: “memcpy(&data,tlmTrans.get_data_ptr()+emReadDataIndex,transferDataLen/8);”
8. Set data pointer to TLM transaction.
   Example: “tlmTrans.set_data_ptr(mDataArray);”

2.22 SFMA (Initiator socket transport data structure)

- **TLM transaction length:** is set to fixed size of data pointer 142 bytes.

  Example: “tlmTrans.set_data_length(142);”

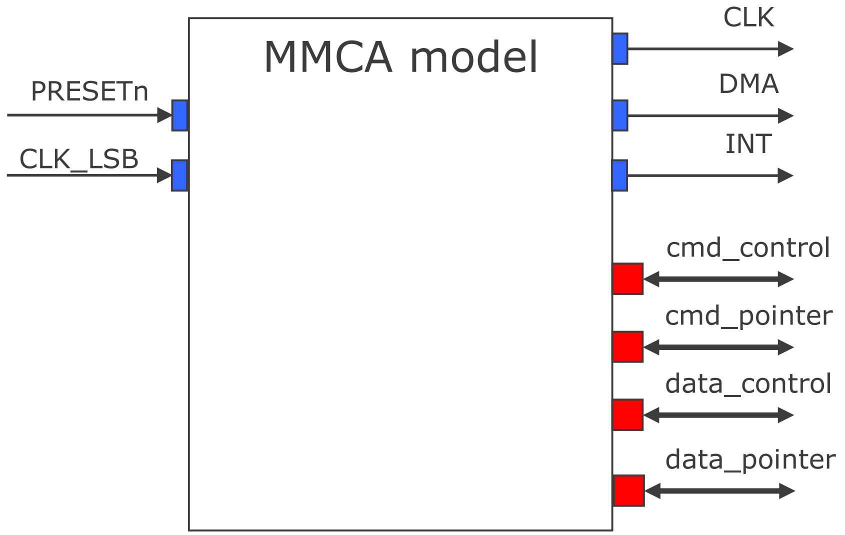
- **Communication time:** total SPI communication time is sent to memory by set to argument of b_transport function

  Example: “is->b_transport(tlmTrans, spiTime);”

Note: In Memory side, after Memory receives a transaction b_transport from SFMA, Memory should not accept any more transaction during SPI communication time.

2.23. MMCA (Interface)

- A/D communicates with Card model via command ports (cmd_control and cmd_pointer) and data ports (data_control and data_pointer).
  - Use command ports (cmd_control and cmd_pointer) to transmit command from MMCA to Card; and receive response from Card to MMCA.
  - Use data ports (data_control and data_pointer) to write data from MMCA to Card; and read data from Card to MMCA.



● cmd_control (sc_inout<uint32_t>)	: contains info to control command.
● cmd_pointer (sc_inout<uint8_t*>)	: pointer to command block.
● data_control (sc_inout<uint32_t>)	: contains info to control data.
● data_pointer (sc_inout<uint8_t*>)	: pointer to data block.

Refer to next slide for detail about control ports and data ports

2.23. MMCA (Explanation of cmd_control and data_control port)

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	LEN	LEN	LEN	LEN	LEN	LEN	LEN	LEN	LEN	LEN	LEN	LEN	LEN	LEN	LEN	LEN
Default	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	INFO	INFO	INFO	INFO	Reserved	Reserved	STBE	STBE
Default	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The cmd_control and data_control have same struct.

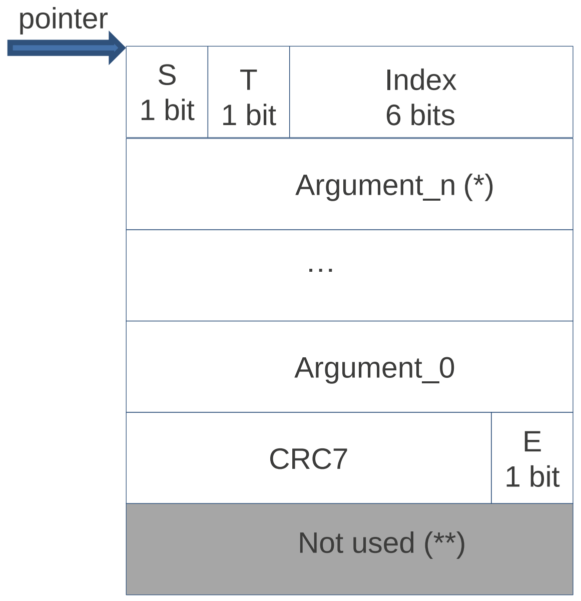
- STBE[1:0]: strobe signal for data/command:
  - 00: Idle
  - 01: Start
  - 10: Stop
  - 11: Abort transaction (in case stop clock or reset)
- INFO[7:4]: Type of data/command:

  In case of cmd_control port:
  - 0x1: Command transfer
  - 0x2: Command response
  - 0x3: Keep low in boot mode

  In case of data_control port:
  - 0x4: Data send (HOST -> CARD)
  - 0x5: Data receive (CARD -> HOST)
  - 0x6: CRC status
  - 0x7: BUSY command
  - 0x8: Boot ACK
  - 0x9 -> 0xF: not used.
- LEN[31:16]: Size of data/command in data_pointer port or cmd_pointer port

2.23. MMCA (Explanation of cmd_pointer port)

Structure of array (for cmd_pointer) is described in figure below:



Note (*): The size of argument is depended on type of command.

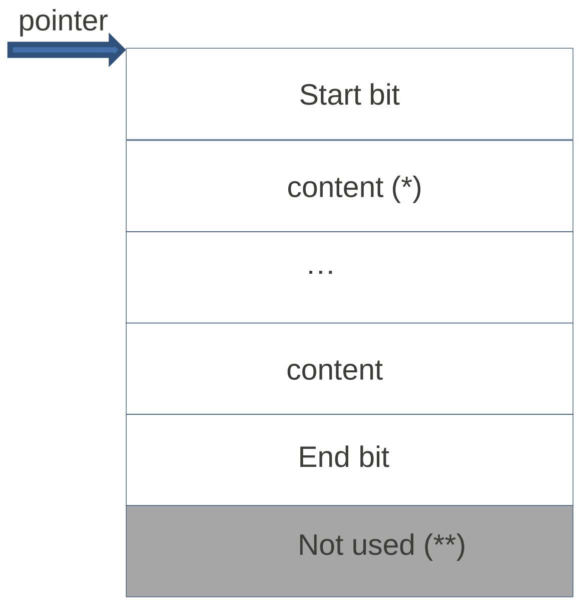
- In case, MMCA sends command to CARD, the argument is the command.
- In case, the CARD sends response (for command) to MMCA, the argument is response.

(**): Number of not used byte is depended on the size of argument.

- In case, size of argument is maximum, there is no “not used” byte.
- In case, size of argument is not maximum, there are “not used” bytes.

2.23. MMCA (Explanation of data_pointer port)

Structure of array (for data_pointer) is described in figure below:



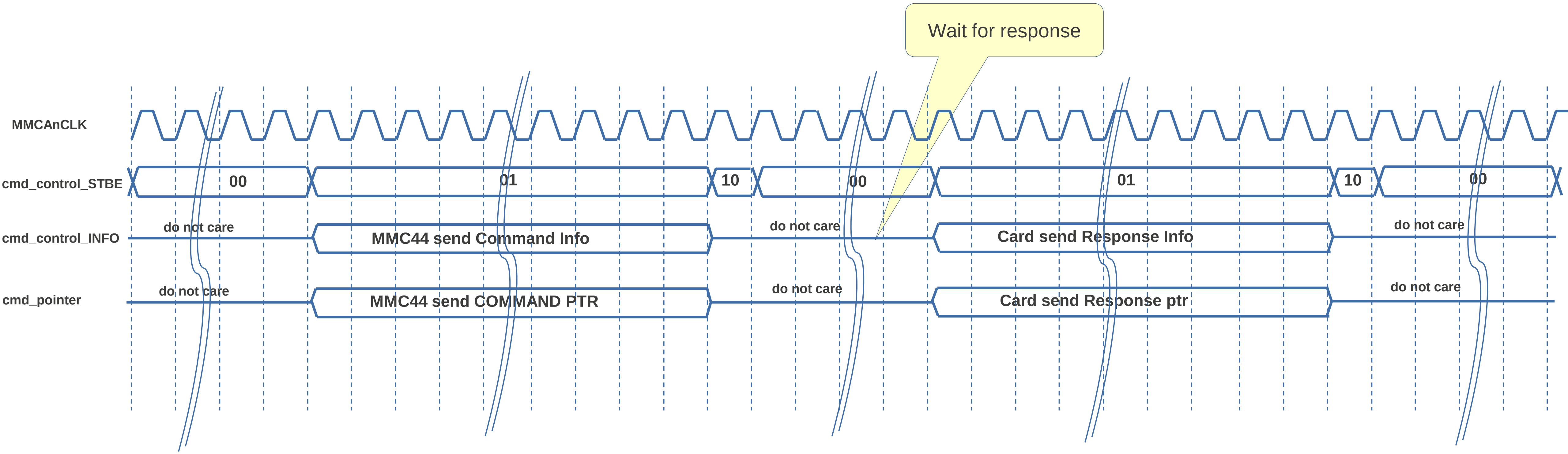
Note (*): The size of content is depended on the type of data:

- In case, the MMCA sends/receives data to/from CARD, the content is data and CRC info.
- In case, the CARD sends response to written data of MMCA, the content is CRC status.
- In case, the CARD is busy (not response to MMCA), the content is empty.
- In case, the CARD sends Boot ACK info to MMCA, the content is Boot ACK info.

(**): Number of not used byte is depended on the size of content.

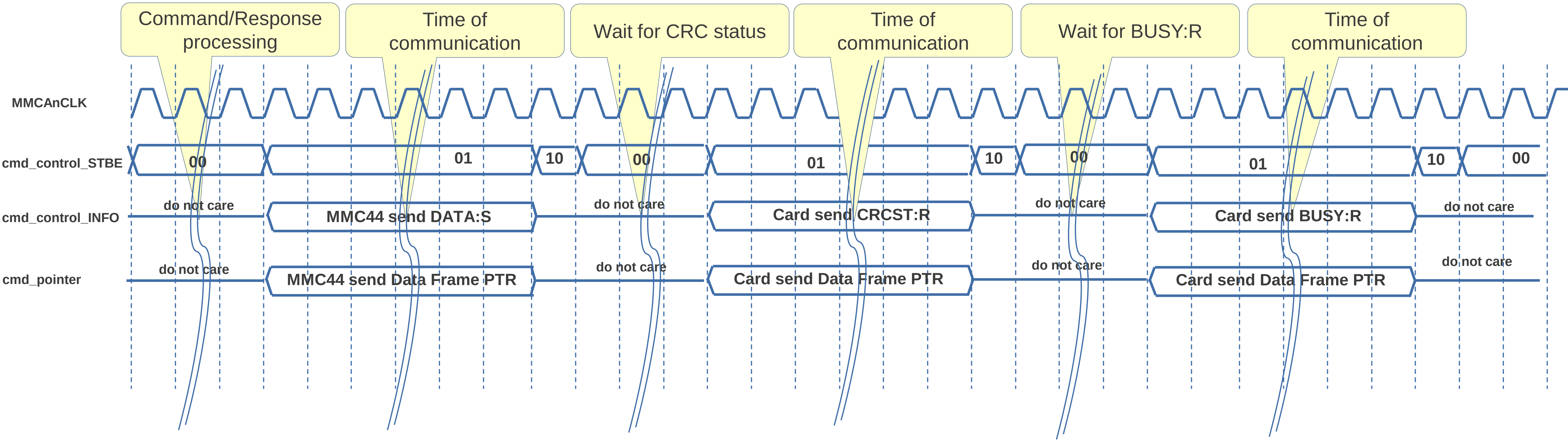
- In case, size of content is maximum, there is no “not used” byte.
- In case, size of content is not maximum, there are “not used” bytes.

2.23. MMCA (Send Command, Receive response)

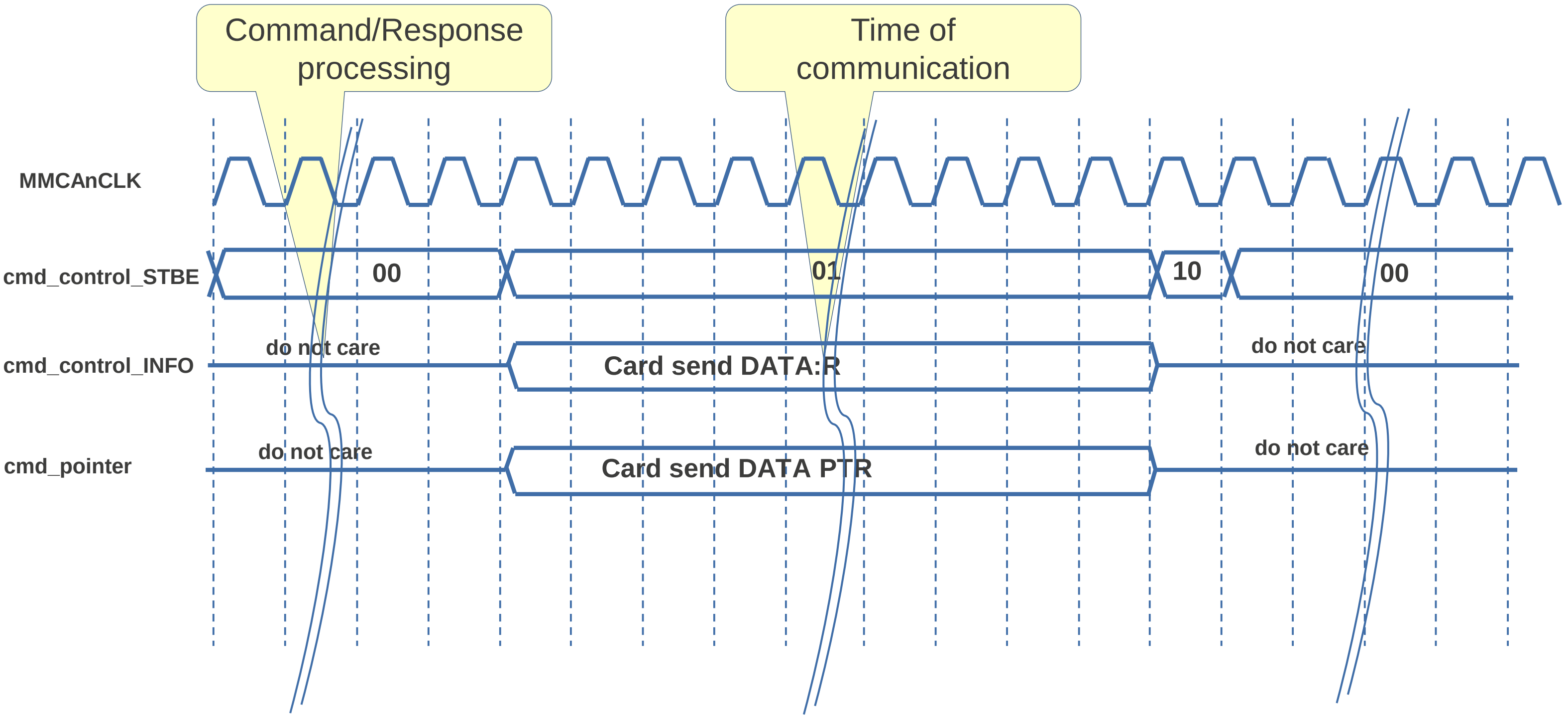


- Serial data is send in parallel data port that data width is 32 bits. (MISO, MOSI or SDI, SDO)
- Transfer start is expressed with CONTROL.STBE=01, and transfer end is expressed with 10.

2.23. MMCA (Single write)



2.23. MMCA (Single read)



List of the Specification to be confirmed of defined

- CAN I/F
- Flex Ray I/F
- Ethernet I/F

Revision History

Revision	Contents	Approved	Checked	Created
1.00	■ Create new	K.Sato 04/27/2012	-	Eiichi.Arai 04/27/2012
2.00	■ Added the rule of debug access for peripheral register in 2.5. Peripheral registers ■ Re-drew figures in page 2.12. SPI, 2.12. SPI (Explanation of CONTROL port), 2.13. I2C, 2.14. UART and LIN and 2.16. UART and LIN (TX/RX_CONTROL)	K.Sato 06/05/2012	-	E.Arai 06/05/2012
3.00	■ Added the behavior at reset in 2.8. reset	-	K.Sato 08/04/2014	E.Arai 08/04/2014
4.00	■ Added NUM bits in TX/RX_CONTROL in 2.16. UART and LIN (TX/RX_CONTROL) ■ Added 2.19. PSI5 ■ Added 1.1.7 Error/Warning/Info Message Output Requirements ■ Added Python method in 2.18. Thermal sensor I/F	K.Yoneyama 11/05/2014	K.Sato 11/05/2014	E.Arai 11/05/2014
5.00	■ Changed the slide master ■ Added 2.1. Signal (sc_signal) ■ Added 2.20. RHSB	K.Yoneyama 12/05/2016	K.Sato 12/05/2016	E.Arai 12/05/2016
6.00	■ Added 2.21 SPI2 ■ Updated appendix (scheap_e3_bus_if_outline_E.ppt)	K.Sato 3/7/2018	-	E.Arai 3/7/2018
7.00	■ Added 1.9. Minimize the invoke of process ■ Updated 2.7. Clock to add guide when clock is 0 (for operation, register access) ■ Added 2.11. Common requirement for serial communication I/F(data packing) ■ Added 2.22. SFMA ■ Added 2.23. MMCA			E.Arai K.Motomura Chan Le 1/14/2019

Renesas Electronics Corporation.