Customer Training Workshop Traveo™ II Inter-Processor Communication (IPC)

Q4 2020

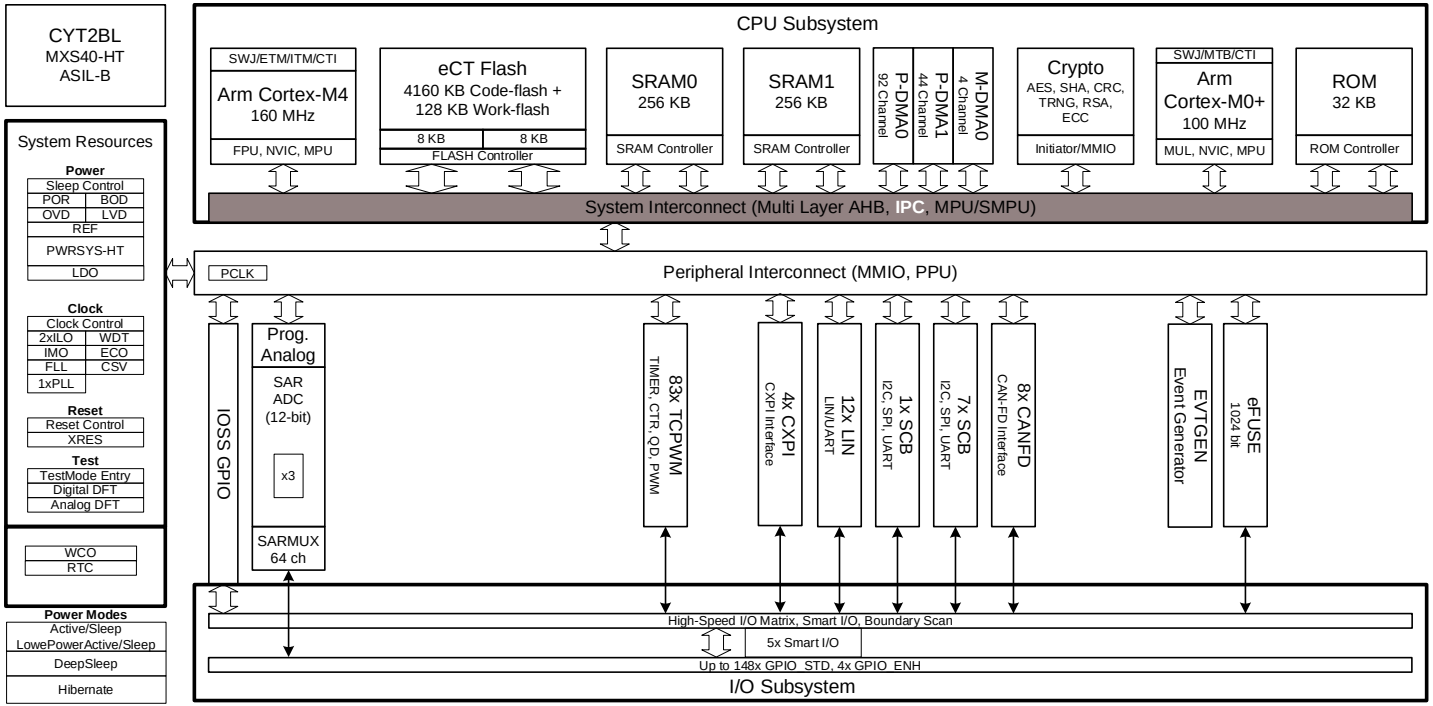
Target Products

› Target product list for this training material

Family Category	Series	Code Flash Memory Size
Traveo™ II Automotive Body Controller Entry	CYT2B6	Up to 576KB
Traveo II Automotive Body Controller Entry	CYT2B7	Up to 1088KB
Traveo II Automotive Body Controller Entry	CYT2B9	Up to 2112KB
Traveo II Automotive Body Controller Entry	CYT2BL	Up to 4160KB
Traveo II Automotive Body Controller High	CYT3BB/4BB	Up to 4160KB
Traveo II Automotive Body Controller High	CYT4BF	Up to 8384KB
Traveo II Automotive Cluster	CYT3DL	Up to 4160KB
Traveo II Automotive Cluster	CYT4DN	Up to 6336KB

Introduction to Traveo II Body Controller Entry

› IPC is implemented in CPUSS

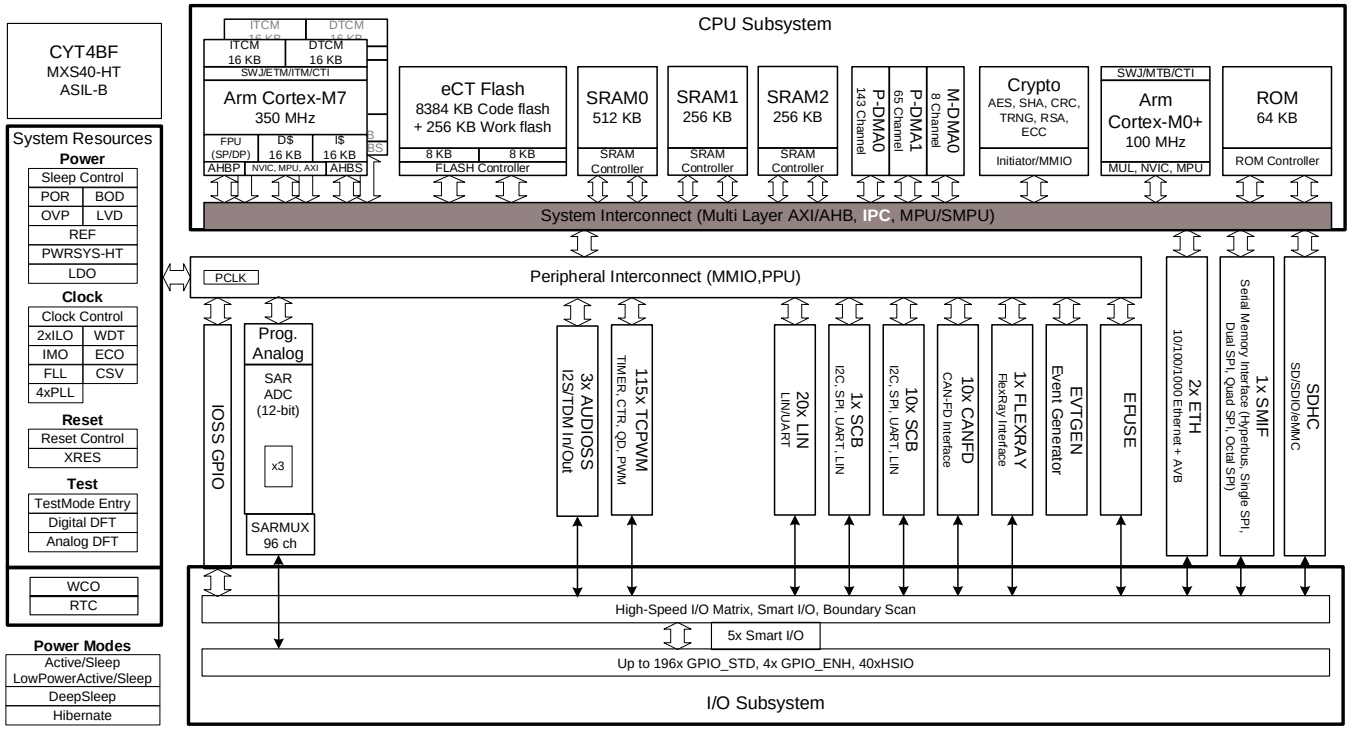


Hint Bar

Review TRM chapter 5 for additional details

Introduction to Traveo II Body Controller High

› IPC is implemented in CPUSS

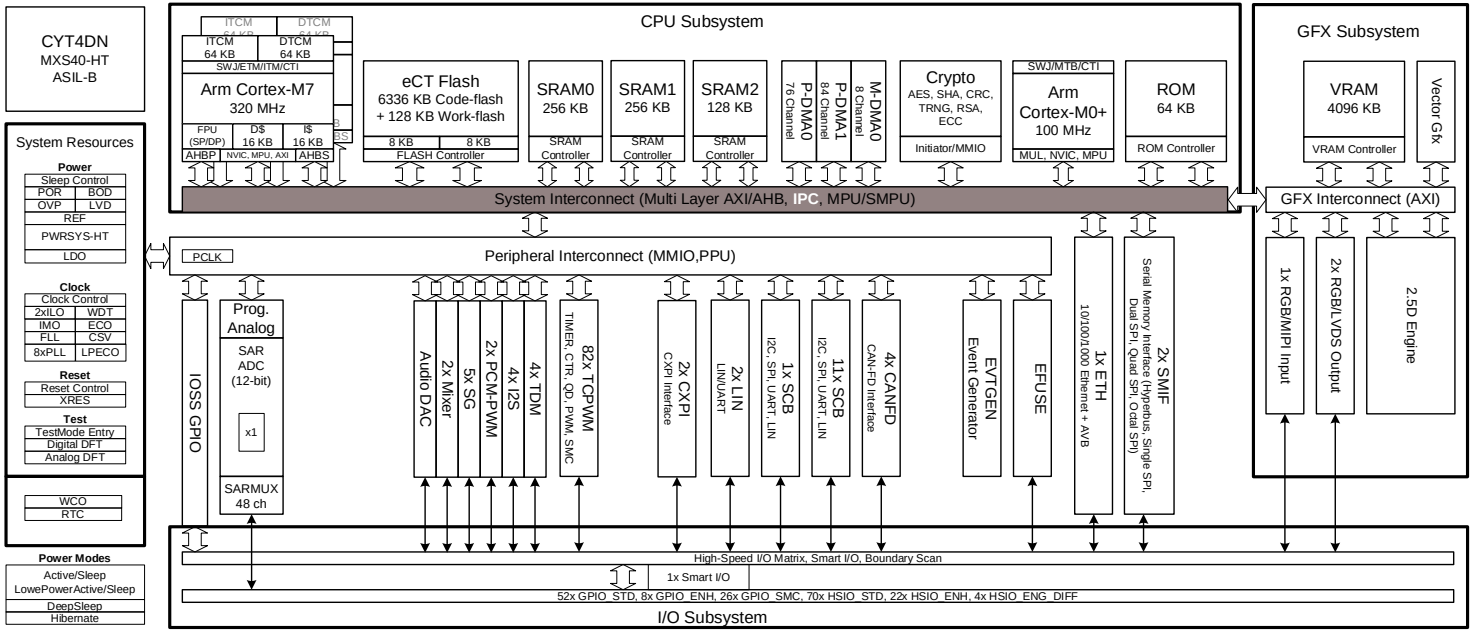


Hint Bar

Review TRM chapter 5 for additional details

Introduction to Traveo II Cluster

› IPC is implemented in CPUSS



Hint Bar

Review TRM chapter 5 for additional details

Inter-Processor Communication (IPC) Overview

- IPC provides the functionality for multiple processors to communicate and synchronize, and includes:
  - Locks for mutual exclusion access
  - Notification and release event generation
  - Data communication
  - Lock status indications
  - Interrupt generation of event to each processor
  - Some IPC channel and interrupt structures are reserved for API use (*)
- IPC has two structure types
  - IPC channel
  - IPC interrupt

Hint Bar

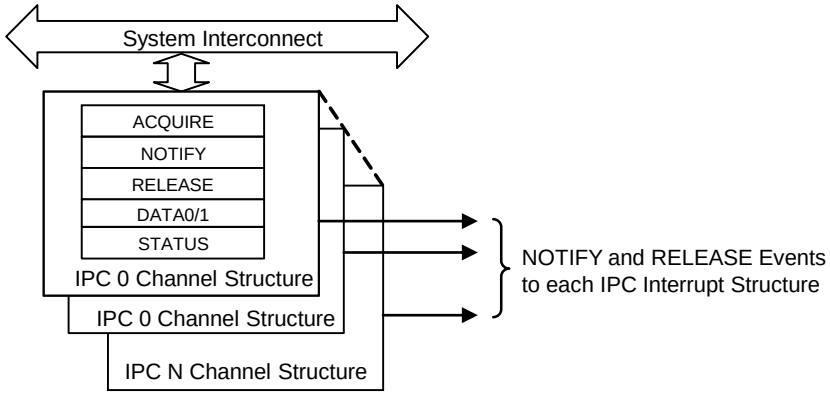
Review TRM sections 5.1.1 and 5.1.2 for additional details

*** Training section reference for additional details about API:**

Nonvolatile Memory Programming

IPC Channel Structure

- Channel structure hardware registers are implemented as:
  - IPC_ACQUIRE: Provides lock feature by reading
  - NOTIFY: Generates notification event
  - RELEASE: Releases the IPC channel
  - DATA0/1: 32-bit register to hold data[1]
  - STATUS: Lock status for the IPC channel[2]



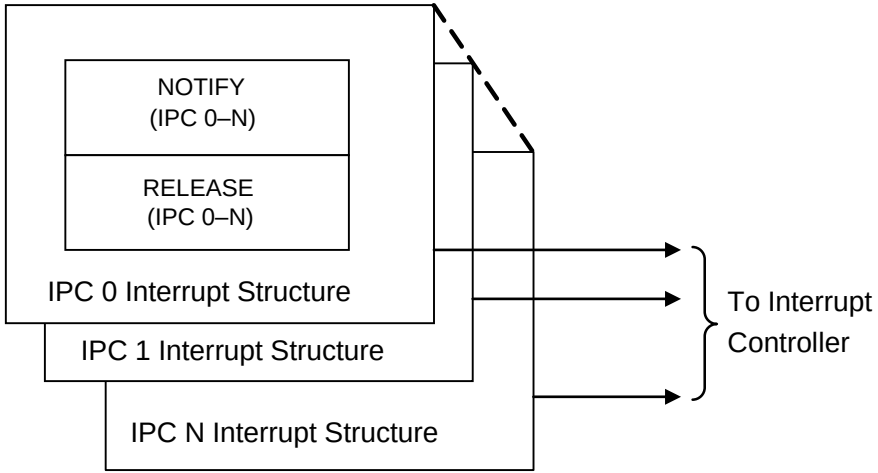
Hint Bar

Review TRM section 5.1.1 for additional details

[1] Can place a address pointer and data size when passing large amounts of data (for example, DATA0 uses address pointer, DATA1 uses data size).
[2] Provides the processor's ID, protection context, and other details.

IPC Interrupt Structure

- Each IPC interrupt has a corresponding IPC interrupt structure that is triggered by a notification or release event from any IPC channel

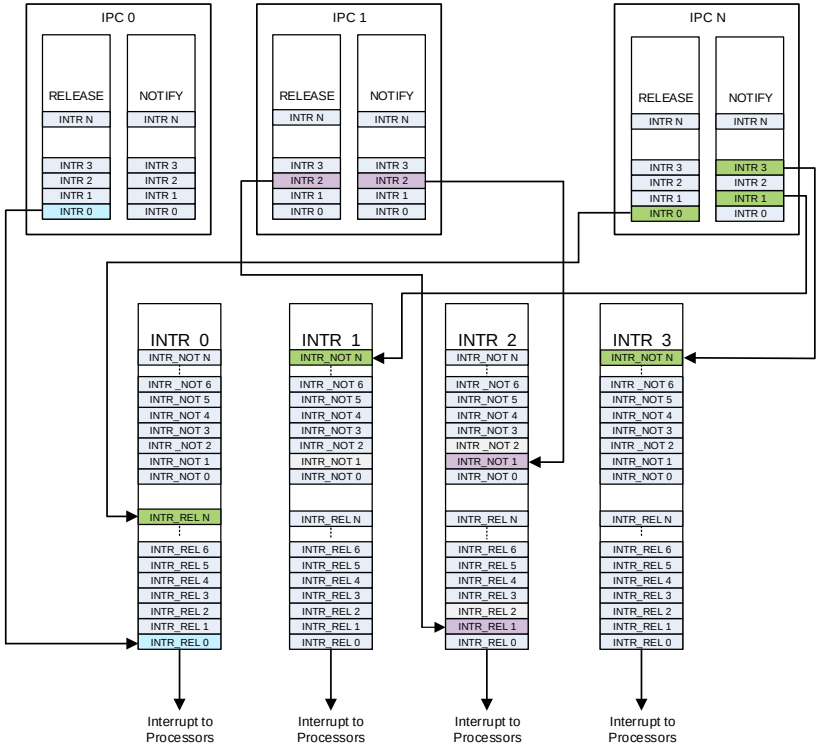


Hint Bar

Review TRM section 5.1.2 for additional details

IPC Channels and Interrupts

- Each IPC interrupt structure[1] configures an interrupt line, which can be triggered by a notify or release event of any IPC channel
  - An IPC interrupt can be triggered from any of the IPC channels in the system
  - The event generated from an IPC channel can trigger any or multiple interrupt structures



Hint Bar

Review TRM section 5.1.3 for additional details

Training section reference for additional details about API

Nonvolatile Memory Programming

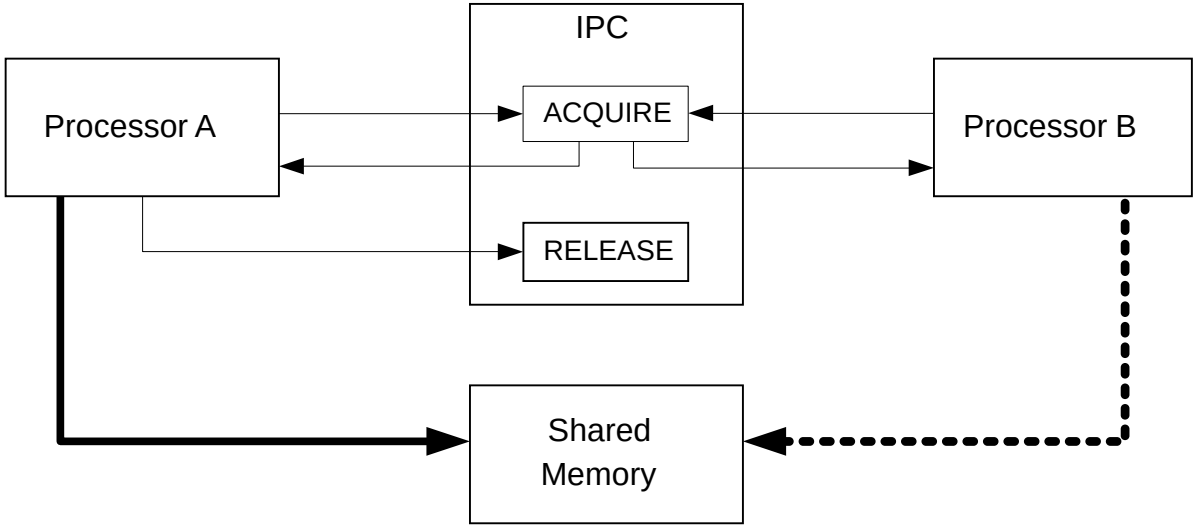
Advantage:

IPC provides the functionality for multiple processors to communicate and synchronize their activities

[1] Any processor can use all interrupt structures. However, some interrupt structures are reserved by the ROM API.

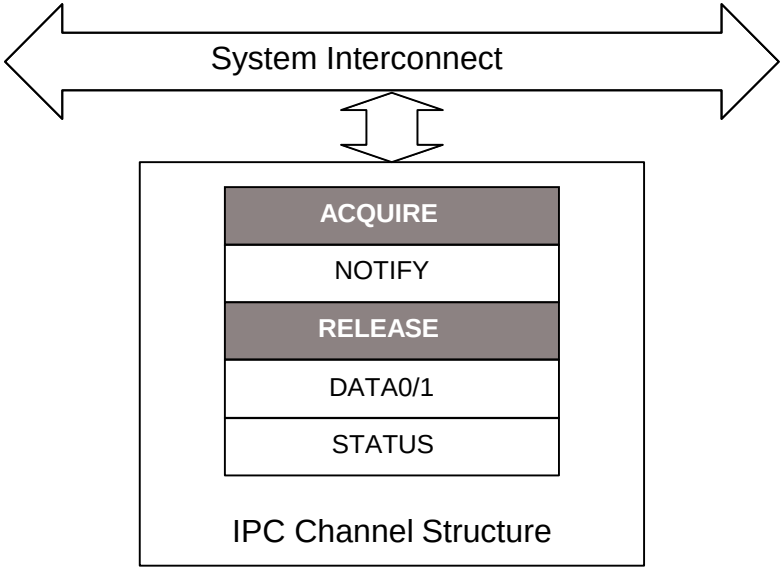
Use Case for Exclusive Control Using IPC

- Implementing Locks
  - Example of exclusive control using lock function of IPC



Use Case for IPC Channel Structure

- ACQUIRE: Locks control by reading
- RELEASE: Unlocks control



Hint Bar

See the datasheet for the number of IPC channels implemented

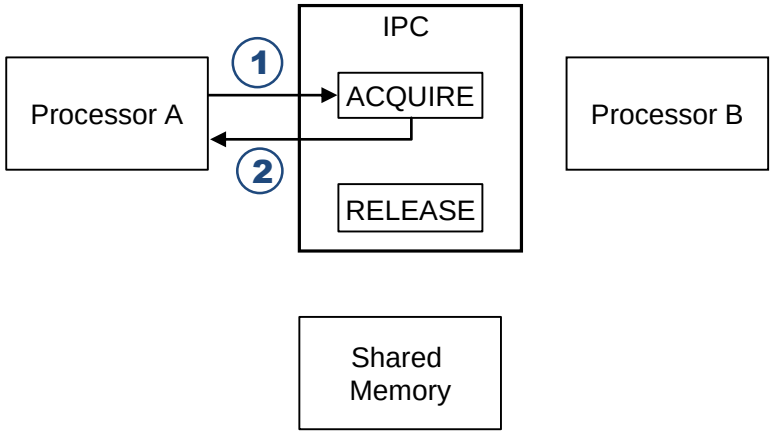
Review TRM section 5.1.1 for additional IPC Channel details

Review TRM section 5.1.2 for additional IPC Interrupt details

Implementing Locks

- Processor A accesses shared memory area usable by Processor A and B
- Only the processor that acquires the lock can access the shared memory

1. Processor A acquires the lock
2. Processor A can read “1”, and lock acquisition succeeds[1]



Hint Bar

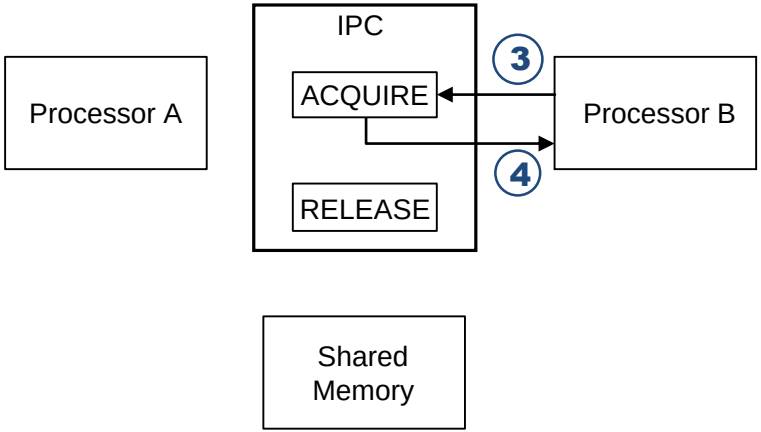
Review the Register TRM for additional details

[1] If read ACQUIRE.SUCCESS = 1, the read acquired the lock. If read ACQUIRE.SUCCESS = 0, the read did not acquire.

Implementing Locks

- Processor A accesses the shared memory area usable by Processor A and B
- Only the processor that acquires the lock can access the shared memory

1. Processor A acquires a lock
2. Processor A can read “1”, and lock acquisition succeeded
3. Processor B acquires the lock
4. Processor B cannot acquire the lock because Processor A has already acquired it[1]



Hint Bar

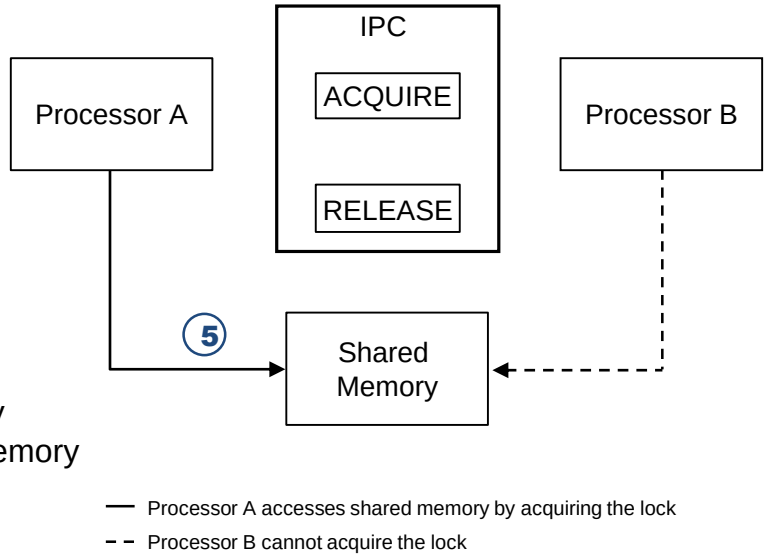
Review the Register TRM for additional details

[1] If another processor has already acquired the lock, register reading is “0” (lock cannot be acquired).

Implementing Locks

- Processor A accesses the shared memory area usable by Processor A and B
- Only the processor that acquires the lock can access the shared memory

1. Processor A acquires a lock
2. Processor A can read “1”, and lock acquisition succeeded
3. Processor B acquires lock
4. Processor B cannot acquire lock because Processor A has already acquired it
5. Processor A accesses shared memory
   Processor B cannot access shared memory

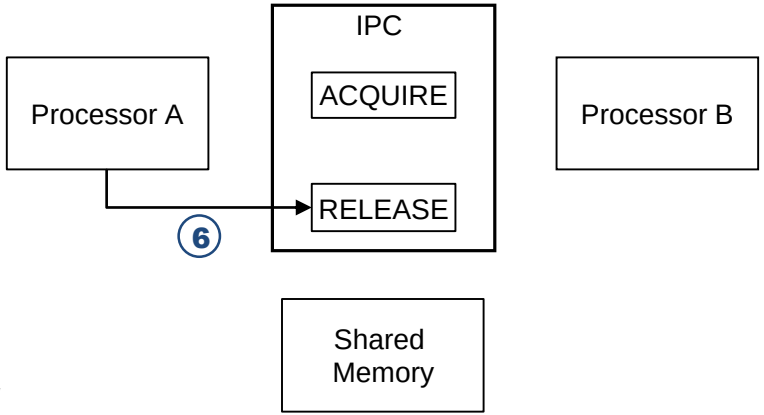


Hint Bar

Implementation of lock function requires the software to set rules

Implementing Locks

- Processor A accesses the shared memory area usable by Processor A and B
- Only the processor that acquires the lock can access the shared memory



1. Processor A acquires a lock
2. Processor A can read “1”, and lock acquisition succeeded
3. Processor B acquires lock
4. Processor B cannot acquire lock because Processor A has already acquired it
5. Processor A accesses shared memory
   Processor B cannot access shared memory
6. Processor A releases the lock after accessing shared memory
   Both Processor A and Processor B can acquire the lock after release

Hint Bar

Review TRM section 5.1.3 for additional details

Advantage:

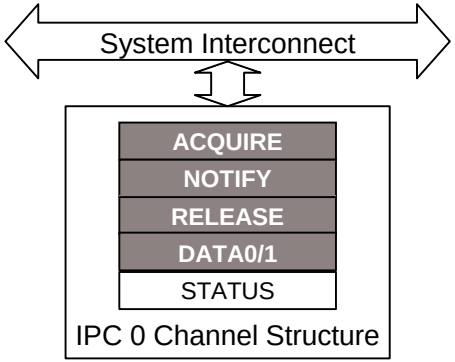
Prevents reading or writing from other processors while writing or reading to shared memory

Use Case for Message Passing Using IPC

- Message Passing
  - Example of short message passing using the DATA register of IPC for communication between processors

Message Passing

- IPC Channel Structures
  - ACQUIRE: Locks control by reading
  - NOTIFY: Generates notification event
  - RELEASE: Generates release event
  - DATA: Data for message passing
  - STATUS: Indicates Lock status of IPC
- IPC Interrupt Structures
  - NOTIFY: Generates NOTIFY interrupt
  - RELEASE: Generates RELEASE interrupt
  - The interrupts can trigger from any IPC structure



Hint Bar

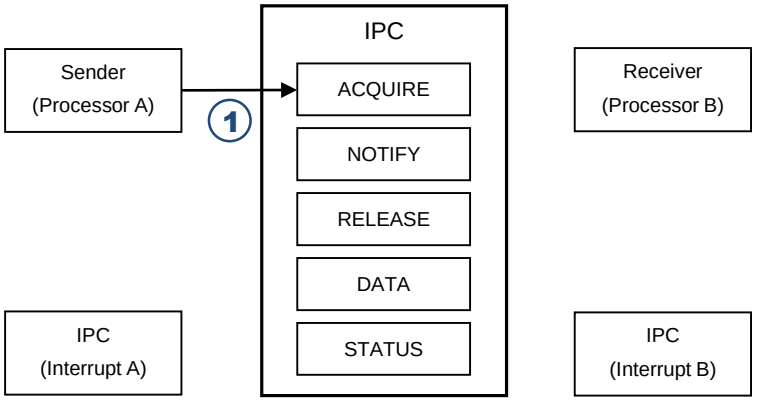
Review TRM section 5.1.1 for additional IPC Channel details

Review TRM section 5.1.2 for additional IPC Interrupt details

Message Passing

- IPC[1] can use the message passing function between processors
  - Using security module from Processor A to Processor B by the API's message communication

1. Sender acquires a lock from the IPC channel



Hint Bar

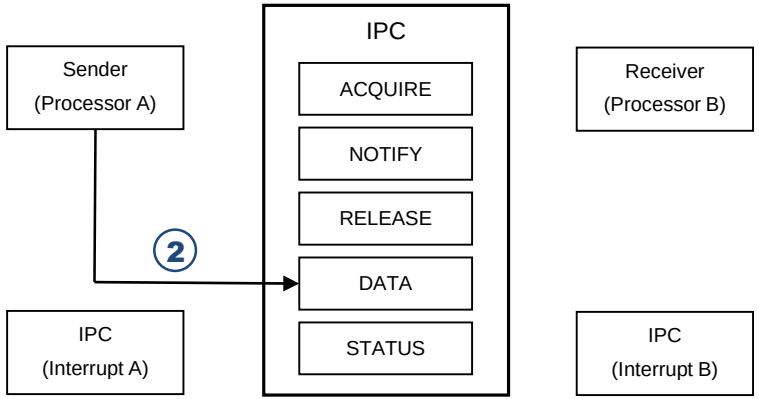
Review TRM section 5.3 for additional details

[1] IPC features two 32-bit data registers (DATA/1).

Message Passing

- IPC can use the message passing function between processors
  - Using security module from Processor A to Processor B by the API's message communication

1. Sender acquires a lock from the IPC channel
2. Sender writes the message data to be sent



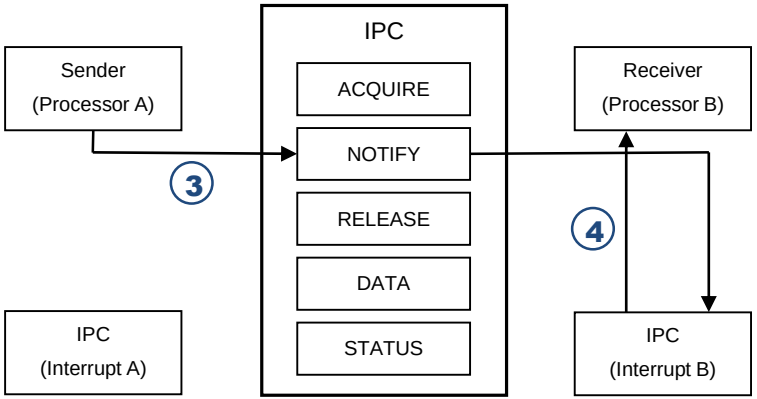
Hint Bar

Review the Register TRM for additional details

Message Passing

- IPC can use the message passing function between processors
  - Using security module from Processor A to Processor B by the API's message communication

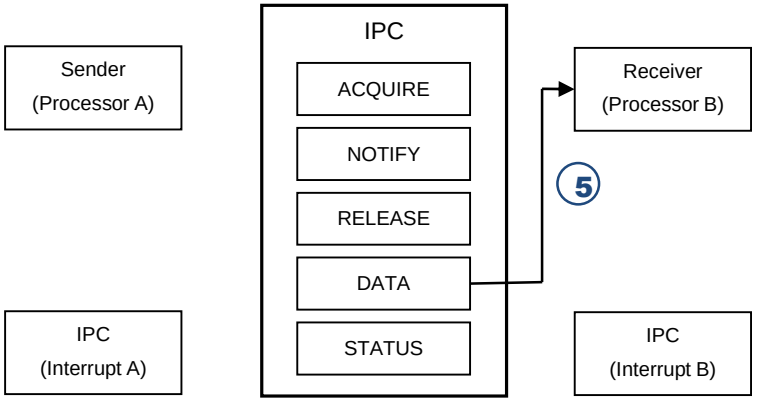
1. Sender acquires a lock from the IPC channel
2. Sender writes the message data to be sent
3. Sender generates a notification event (by writing to the NOTIFY register) to the receiver
4. IPC generates interrupt to the receiver



Message Passing

- IPC can use the message passing function between processors
  - Using security module from Processor A to Processor B by the API's message communication

1. Sender acquires a lock from the IPC channel
2. Sender writes the message data to be sent
3. Sender generates a notification event (by writing NOTIFY register) to the receiver
4. IPC generates interrupt to the receiver

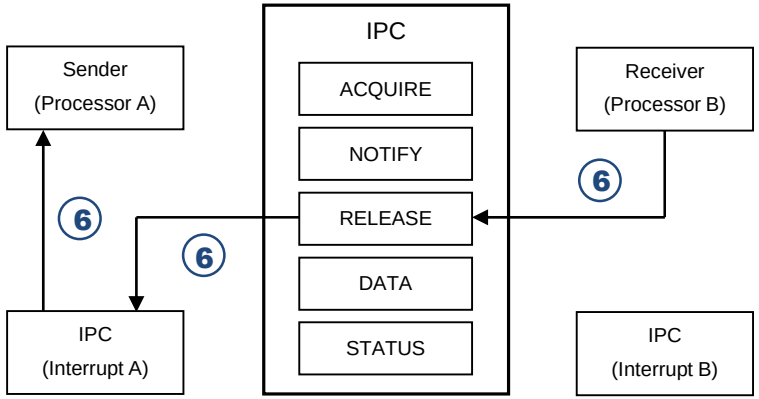


5. Receiver reads message data from the DATA register after checking the triggered IPC channel

Message Passing

- IPC can use the message passing function between processors
  - Using security module from Processor A to Processor B by the API's message communication

1. Sender acquires a lock from the IPC channel
2. Sender writes the message data to be sent
3. Sender generates a notification event (by writing NOTIFY register) to the receiver
4. IPC generates interrupt to the receiver
5. Receiver reads message data from the DATA register after checking the triggered IPC channel
6. Receiver releases the channel using the RELEASE register and also generates a release event



Hint Bar

Use Case:
Security Module from Main CPU to Secondary CPU by API message communication

If release event was not masked, a release event is generated for the sender

Use Case for Large Message Passing Using IPC

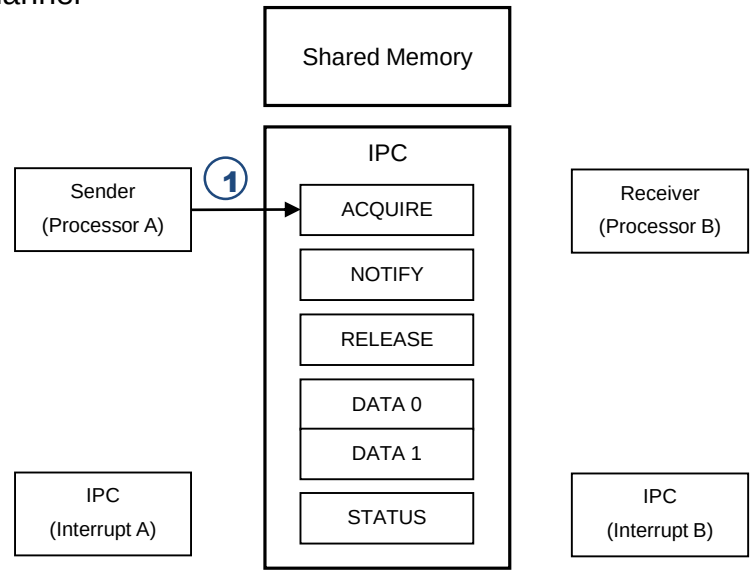
- Large Message Passing
  - Example of large message passing using IPC
  - Large message passing can be activated by storing message data in shared memory and passing the address pointer of the message data by using the IPC DATA register (the following slides illustrate the large message passing sequence)
  - It can make the passed data variable in length by passing the address pointer and size (for example, DATA0 uses address pointer, DATA1 uses data size)

Use Case:

Large Message Passing

- Larger messages (>32-bit x2) can be sent as pointers
  - Flash operation using APIs from Processor A to Processor B

1. Sender acquires a lock from the IPC channel



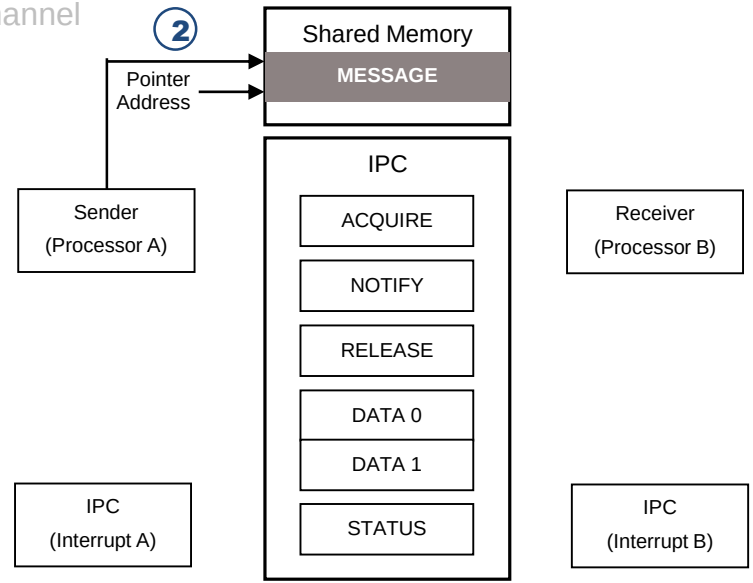
Hint Bar

Review TRM section 5.3 for additional details

Use Case for Large Message Passing

- Larger messages (>32-bit x2) can be sent as pointers
  - Flash operation using APIs from Processor A to Processor B

1. Sender acquires a lock from the IPC channel
2. Sender writes the message data to the shared memory[1]



Hint Bar

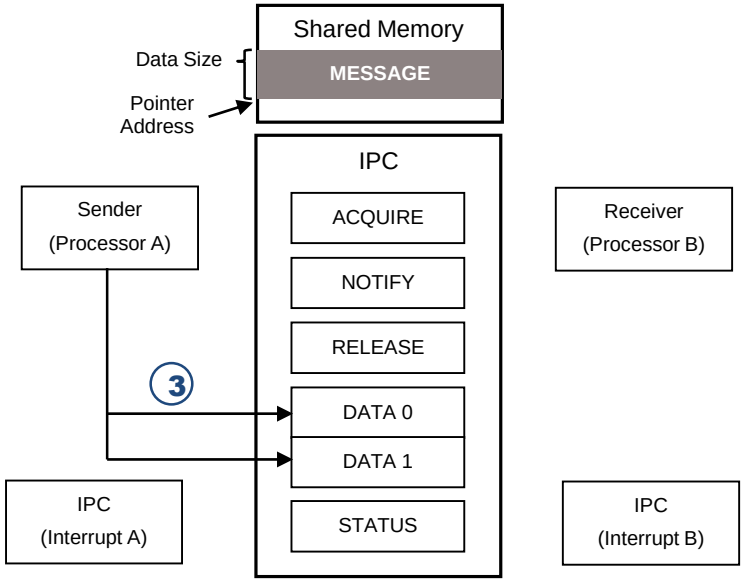
The MPU and SMPU are effective in preventing message destruction of memory used for data passing from other masters

[1] Messages in shared memory must be protected from erroneous rewriting from other masters.

Large Message Passing

- Larger messages (>32-bit x2) can be sent as pointers
  - Flash operation using APIs from Processor A to Processor B

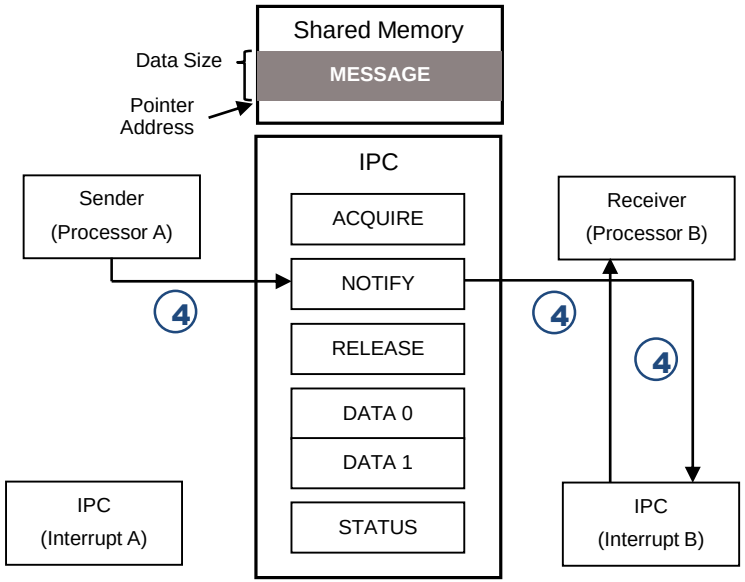
1. Sender acquires a lock from the IPC channel
2. Sender writes the message data to the shared memory
3. Sender writes pointer address to DATA0, and data size to DATA1



Large Message Passing

- Larger messages (>32-bit x2) can be sent as pointers
  - Flash operation using APIs from Processor A to Processor B

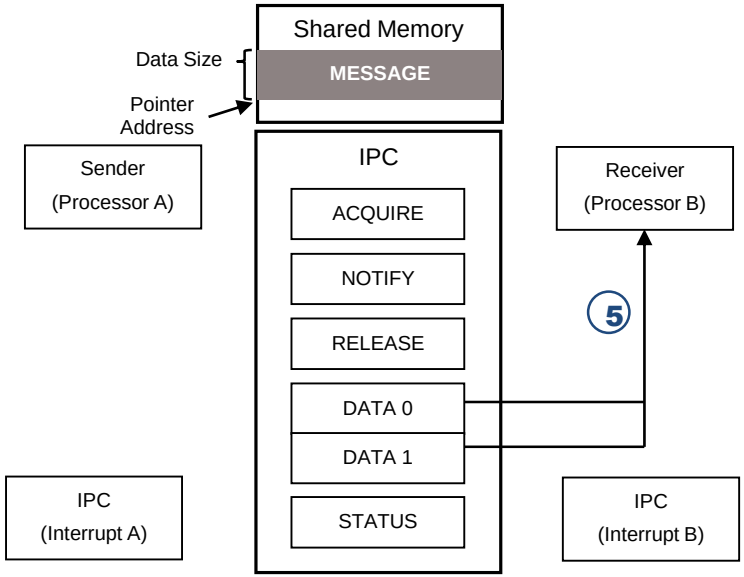
1. Sender acquires a lock from the IPC channel
2. Sender writes the message data to the shared memory
3. Sender writes pointer address to DATA0, and data size to DATA1
4. Sender generates a notification event to the receiver



Large Message Passing

- Larger messages (>32-bit x2) can be sent as pointers
  - Flash operation using APIs from Processor A to Processor B

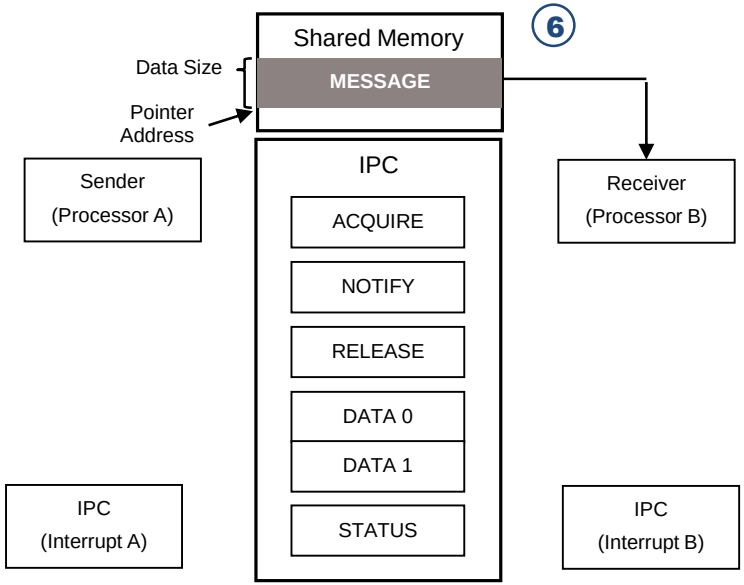
1. Sender acquires a lock from the IPC channel
2. Sender writes the message data to the shared memory
3. Sender writes pointer address to DATA0, and data size to DATA1
4. Sender generates a notification event to the receiver
5. Receiver can read the pointer and data size from the DATA0/1 by using the notification event



Large Message Passing

- Larger messages (>32-bit x2) can be sent as pointers
  - Flash operation using APIs from Processor A to Processor B

1. Sender acquires a lock from the IPC channel
2. Sender writes the message data to the shared memory
3. Sender writes pointer address to DATA0, and data size to DATA1
4. Sender generates a notification event to the receiver
5. Receiver can read the pointer and data size from the DATA0/1 by using the notification event
6. Receiver reads message from address indicated by pointer



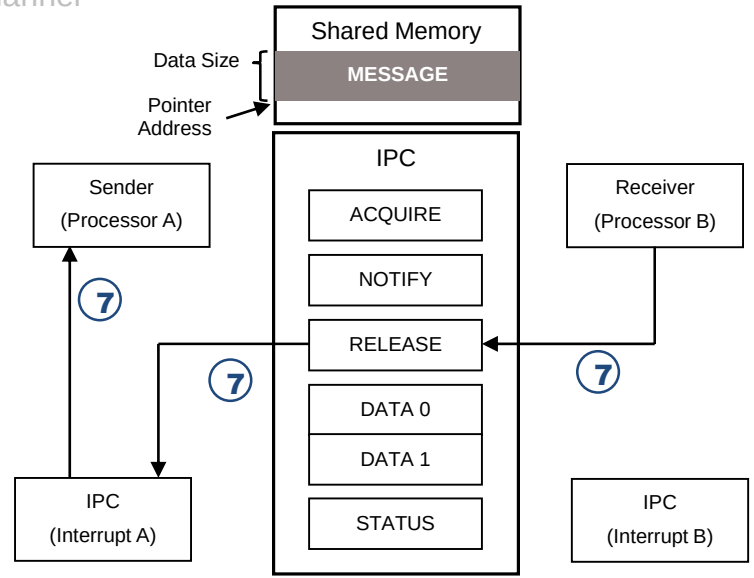
[1] Messages in memory must be protected from erroneous rewriting from other masters.

Use Case:

Large Message Passing

- Larger ,messages (> 32-bit x2) can be sent as pointers
  - Flash operation using APIs from Processor A to Processor B

1. Sender acquires a lock from the IPC channel
2. Sender writes the message data to the shared memory
3. Sender writes pointer address to DATA0, and data size to DATA1
4. Sender generates a notification event to the receiver
5. Receiver can read the pointer and data size from the DATA0/1 by using the notification event
6. Receiver reads message from address indicated by pointer
7. Receiver releases the IPC channel and generates a release event



Hint Bar

Review TRM section 5.3 for additional details

Protection of message area is released when IPC is released

Training section reference for additional details about API

Nonvolatile Memory Programming

Use Case for Mixing Short and Large Message Using IPC

- Software requires to set rules according to the situation
- Case 1: Use a dedicated IPC structure depending on short and large messages
  - You can use the IPC6 structure for short messages and IPC7 structure for large messages only
- Case 2: Use a dedicated IPC interrupt structure depending on short and large messages
  - You can use the IPC6 interrupt structure for short messages and IPC7 interrupt structure for large messages only
  - Different interrupt handlers are required depending on the message size
  - You can use the same IPC channel structure
- Case 3: Use a dedicated code depending on short and large messages
  - Use part of the data register to identify short or large messages (for example, short: 0x05, large: 0x0a)
  - The receiver should check the message size
  - For short messages, passing data size is up to 64 bits including code and data

Note: Some IPC channel and interrupt structures are reserved for API use

Part of your life. Part of tomorrow.

Revision History

Revision	ECN	Submission Date	Description of Change
**	6084432	03/12/2018	Initial release
*A	6390493	11/21/2018	Added slides 2, 7, and 20. Updated slides 3 and 4 Deleted IPC Channel/Interrupt Register Structures
*B	6633414	7/22/2019	Updated slide 2-4. Added slide 5.
*C	7060646	01/06/2021	Updated slide 2-3, 23 - 30. Added slide 31