I2C 接口函数

*1. int FI2cPs_init(FMSH_device *dev)*

描述	* This function initializes the I2C driver. It disables and * clears all interrupts, sets the DMA mode to software handshaking, * sets the DMA transmit and receive notifier function pointers to NULL * and disables the I2C. It also attempts to determine the hardware * parameters of the device, if supported by the device.
参数	* @param dev is i2c device handle.
返回值	* @return * 0 -- if successful * -FMSH_ENOSYS -- hardware parameters for the device could not be * automatically determined

*2. void FI2cPs_enable(FMSH_device *dev)*

描述	* This function enables the I2C.
参数	* @param dev is i2c device handle.
返回值	* @return * none

*3. int FI2cPs_disable(FMSH_device *dev)*

描述	* This functions disables the I2C, if it is not busy (determined by

	* the activity interrupt bit). The I2C should not be disabled during * interrupt-driven transfers as the resulting driver behavior is * undefined.
参数	* @param dev is i2c device handle.
返回值	* @return * 0 -- if successful * -FMSH_EBUSY -- if the I2C is busy

*4. bool FI2cPs_isEnabled(FMSH_device *dev)*

描述	* This function returns whether the I2C is enabled or not.
参数	* @param dev is i2c device handle.
返回值	* @return * true -- the I2C is enabled * false -- the I2C is disabled

*5. bool FI2cPs_isBusy(FMSH_device *dev)*

描述	* This function returns whether the I2C is busy (transmitting * or receiving) or not.
参数	* @param dev is i2c device handle.
返回值	* @return * true -- the I2C device is busy * false -- the I2C device is not busy

*6. int FI2cPs_setSpeedMode(FMSH_device *dev, FIicPs_SpeedMode_T mode)*

描述	* This function sets the speed mode used for I2C transfers.
参数	* @param * dev is i2c device handle. * mode is the speed mode to set
返回值	* @return * 0 -- if successful * -FMSH_EPERM -- if the I2C is enabled * -FMSH_ENOSYS -- if the specified speed is not supported

*7. FIicPs_SpeedMode_T FI2cPs_getSpeedMode(FMSH_device *dev)*

描述	* *This function returns the speed mode currently in use by the I2C.*
参数	* *@param* * *dev is i2c device handle.*
返回值	* *@return* * *The current I2C speed mode.*

*8. int FI2cPs_setMasterAddressMode(FMSH_device *dev, FIicPs_AddressMode_T mode)*

描述	* This function sets the master addressing mode (7-bit or 10-bit).
参数	* @param * dev is i2c device handle. * mode is the addressing mode to set.
返回值	* @return * 0 -- if successful

	* -FMSH_EPERM -- if the I2C is enabled

*9. int FI2cPs_setSlaveAddressMode(FMSH_device *dev, FIicPs_AddressMode_T mode)*

描述	* This function sets the I2C slave addressing mode (7-bit or 10-bit).
参数	* @param * dev is i2c device handle. * mode is the addressing mode to set.
返回值	* @return * 0 -- if successful * -FMSH_EPERM -- if the I2C is enabled

*10. FIicPs_AddressMode_T FI2cPs_getSlaveAddressMode(FMSH_device *dev)*

描述	* This function returns the current slave addressing mode (7-bit or * 10-bit).
参数	* @param * dev is i2c device handle.
返回值	* @return * The current slave addressing mode.

*11. int FI2cPs_enableSlave(FMSH_device *dev)*

描述	* This function enables the I2C slave.
参数	* @param * dev is i2c device handle.
返回值	* @return

	* 0 -- if successful * -FMSH_EPERM -- if the I2C is enabled

*12. int FI2cPs_disableSlave(FMSH_device *dev)*

描述	* This function disables the I2C slave.
参数	* @param * dev is i2c device handle.
返回值	* @return * 0 -- if successful * -FMSH_EPERM -- if the I2C is enabled

*13. bool FI2cPs_isSlaveEnabled(FMSH_device *dev)*

描述	* This function returns whether the I2C slave is enabled or not.
参数	* @param * dev is i2c device handle.
返回值	* @return * true -- slave is enabled * false -- slave is disabled

*14. int FI2cPs_enableMaster(FMSH_device *dev)*

描述	* This function enables the I2C master.
参数	* @param * dev is i2c device handle.
返回值	* @return

	* 0 -- if successful * -FMSH_EPERM -- if the I2C is enabled

*15. int FI2cPs_disableMaster(FMSH_device *dev)*

描述	* This function disables the I2C master.
参数	* @param * dev is i2c device handle.
返回值	* @return * 0 -- if successful * -FMSH_EPERM -- if the I2C is enabled

*16. bool FI2cPs_isMasterEnabled(FMSH_device *dev)*

描述	* This function returns whether the I2C master is enabled or not.
参数	* @param * dev is i2c device handle.
返回值	* @return * true -- master is enabled * false -- master is disabled

*17. int FI2cPs_enableRestart(FMSH_device *dev)*

描述	* This function enables the use of restart conditions.
参数	* @param * dev is i2c device handle.
返回值	* @return

	* 0 -- if successful * -FMSH_EPERM -- if the I2C is enabled

*18. int FI2cPs_disableRestart(FMSH_device *dev))*

描述	* This function disables the use of restart conditions.
参数	* @param * dev is i2c device handle.
返回值	* @return * 0 -- if successful * -FMSH_EPERM -- if the I2C is enabled

*19. bool FI2cPs_isRestartEnabled(FMSH_device *dev)*

描述	* This function returns whether restart conditions are currently in * use or not by the I2C.
参数	* @param * dev is i2c device handle.
返回值	* @return * true -- restart conditions are enabled * false -- restart conditions are disabled

*20. int FI2cPs_setTargetAddress(FMSH_device *dev, uint16_t address)*

描述	* This function sets the target address used by the I2C master. When * not issuing a general call or using a start byte, this is the * address the master uses when performing transfers over the I2C bus.

参数	* @param * dev is i2c device handle. * address is target address to set
返回值	* @return * 0 -- if successful * -FMSH_EPERM -- if the I2C is enabled

uint16_t FI2cPs_getTargetAddress(FMSH_device *dev)

描述	* This function returns the current target address in use by the I2C * master.
参数	* @param * dev is i2c device handle.
返回值	* @return * The current target address.

*21. int FI2cPs_setSlaveAddress(FMSH_device *dev, uint16_t address)*

描述	* This function sets the slave address to which the I2C slave * responds, when enabled.
参数	* @param * dev is i2c device handle. * address is slave address to set
返回值	* @return * 0 -- if successful * -FMSH_EPERM -- if the I2C is enabled

*22. uint16_t FI2cPs_getSlaveAddress(FMSH_device *dev)*

描述	* This function returns the current address in use by the I2C slave.
参数	* @param * dev is i2c device handle.
返回值	* @return * The current I2C slave address.

*23. int FI2cPs_setTxMode(FMSH_device *dev, FIicPs_TxMode_T mode)*

描述	* This function sets the master transmit mode. That is, whether to * use a start byte, general call, or the programmed target address.
参数	* @param * dev is i2c device handle. * mode is transfer mode to set
返回值	* @return * 0 -- if successful * -FMSH_EPERM -- if the I2C is enabled

*24. FIicPs_TxMode_T FI2cPs_getTxMode(FMSH_device *dev)*

描述	* This function returns the current transmit mode in use by an I2C * master.
参数	* @param * dev is i2c device handle.
返回值	* @return * The current master transmit mode.

*25. int FI2cPs_setMasterCode(FMSH_device *dev, uint8_t code)*

描述	* This function sets the master code, used during high-speed mode * transfers.
参数	* @param * dev is i2c device handle. * code is master code to set
返回值	* @return * 0 -- if successful * -FMSH_EPERM -- if the I2C is enabled

*26. uint8_t FI2cPs_getMasterCode(FMSH_device *dev)*

描述	* Initializes an I2C peripheral.
参数	* @param * dev is i2c device handle.
返回值	* @return * The current high-speed mode master code.

*27. int FI2cPs_setSclCount(FMSH_device *dev, FIicPs_SpeedMode_T mode, FIicPs_SclPhase_T phase, uint16_t count)*

描述	* This function set the scl count value for a particular speed mode * (standard, fast, high) and clock phase (low, high).
参数	* @param * dev is i2c device handle. * mode is speed mode of count value to set

	* phase is scl phase of count value to set * value is count value to set
返回值	* @return * 0 -- if successful * -FMSH_EPERM -- if the I2C is enabled * -FMSH_ENOSYS -- if the scl count registers are hardcoded

*28. uint16_t FI2cPs_getSclCount(FMSH_device *dev, FIicPs_SpeedMode_T mode, FIicPs_SclPhase_T phase)*

描述	* This function returns the current scl count value for all speed * modes (standard, fast, high) and phases (low, high).
参数	* @param * dev is i2c device handle. * mode is speed mode to get count value of * phase is scl phase to get count value of
返回值	* @return * The current specified scl count value.

*29. uint8_t FI2cPs_read(FMSH_device *dev)*

描述	* This function reads a single byte from the I2C receive FIFO.
参数	* @param * dev is i2c device handle.
返回值	* @return * The character read from the I2C FIFO

*30. void FI2cPs_write(FMSH_device *dev, uint8_t character)*

描述	* This function writes a single byte to the I2C transmit FIFO.
参数	* @param * dev is i2c device handle. * datum is byte to write to FIFO
返回值	* @return * none

*31. void FI2cPs_issueRead(FMSH_device *dev)*

描述	* This function writes a read command to the I2C transmit FIFO. This * is used during master-receiver/slave-transmitter transfers and is * typically followed by a read from the master receive FIFO after the * slave responds with data.
参数	* @param * dev is i2c device handle.
返回值	* @return None.

*32. FIicPs_TxAbort_T FI2cPs_getTxAbortSource(FMSH_device *dev)*

描述	* This function returns the current value of the I2C transmit abort * status register.
参数	* @param * dev is i2c device handle.
返回值	* @return * The current transmit abort status.

*33. uint16_t FI2cPs_getTxFifoDepth(FMSH_device *dev)*

描述	* Returns how many bytes deep the I2C transmit FIFO is.
参数	* @param * dev is i2c device handle.
返回值	* @return * FIFO depth in bytes (from 2 to 256)

*34. uint16_t FI2cPs_getRxFifoDepth(FMSH_device *dev)*

描述	* Returns how many bytes deep the I2C transmit FIFO is.
参数	* @param * dev is i2c device handle.
返回值	* @return * FIFO depth in bytes (from 2 to 256)

*35. bool FI2cPs_isTxFifoFull(FMSH_device *dev)*

描述	* Returns whether the transmitter FIFO is full or not.
参数	* @param * dev is i2c device handle.
返回值	* @return * true -- the transmit FIFO is full * false -- the transmit FIFO is not full

*36. bool FI2cPs_isTxFifoEmpty(FMSH_device *dev)*

描述	* Returns whether the transmitter FIFO is empty or not.
参数	* @param * dev is i2c device handle.
返回值	* @return * true -- the transmit FIFO is full * false -- the transmit FIFO is not full

*37. bool FI2cPs_isRxFifoFull(FMSH_device *dev)*

描述	* This function returns whether the receive FIFO is full or not.
参数	* @param * dev is i2c device handle.
返回值	* @return * true -- the receive FIFO is full * false -- the receive FIFO is not full

*38. bool FI2cPs_isRxFifoEmpty(FMSH_device *dev)*

描述	* This function returns whether the receive FIFO is empty or not.
参数	* @param * dev is i2c device handle.
返回值	* @return * true -- the receive FIFO is empty * false -- the receive FIFO is not empty

*39. uint16_t FI2cPs_getTxFifoLevel(FMSH_device *dev)*

描述	* This function returns the number of valid data entries currently * present in the transmit FIFO.
参数	* @param * dev is i2c device handle.
返回值	* @return * Number of valid data entries in the transmit FIFO.

*40. uint16_t FI2cPs_getRxFifoLevel(FMSH_device *dev)*

描述	* This function returns the number of valid data entries currently * present in the receiver FIFO.
参数	* @param * dev is i2c device handle.
返回值	* @return * Number of valid data entries in the receive FIFO.

*41. int FI2cPs_setTxThreshold(FMSH_device *dev, uint8_t level)*

描述	* This function sets the threshold level for the transmit FIFO. When * the number of data entries in the transmit FIFO is at or below this * level, the tx_empty interrupt is triggered. If an interrupt-driven * transfer is already in progress, the transmit threshold level is not * updated until the end of the transfer.
参数	* @param * dev is i2c device handle.

	* level is level at which to set threshold
返回值	* @return * 0 -- if successful * -FMSH_EINVAL -- if the level specified is greater than the transmit * FIFO depth; the threshold is set to the transmit FIFO * depth. * -FMSH_EBUSY -- if an interrupt-driven transfer is currently in * progress; the requested level will be written to the * transmit threshold register when the current transfer * completes.

*42. uint8_t FI2cPs_getTxThreshold(FMSH_device *dev)*

描述	* This function returns the current threshold level for the transmit * FIFO.
参数	* @param * dev is i2c device handle.
返回值	* @return * The transmit FIFO threshold level.

*43. int FI2cPs_setRxThreshold(FMSH_device *dev, uint8_t level)*

描述	* This function sets the threshold level for the receive FIFO. When * the number of data entries in the receive FIFO is at or above this * level, the rx_full interrupt is triggered. If an interrupt-driven * transfer is already in progress, the receive threshold level is not * updated until the end of the transfer.

参数	* @param * dev is i2c device handle. * level is level at which to set threshold
返回值	* @return * 0 -- if successful * -FMSH_EINVAL -- if the level specified is greater than the receive * FIFO depth, the threshold is set to the receive FIFO * depth. * -FMSH_EBUSY -- if an interrupt-driven transfer is currently in * progress, the requested level is written to the * receive threshold register when the current transfer * completes.

*44. uint8_t FI2cPs_getRxThreshold(FMSH_device *dev)*

描述	* This function returns the current threshold level for the receive * FIFO.
参数	* @param * dev is i2c device handle.
返回值	* @return * The receive FIFO threshold level.

*45. void FI2cPs_setListener(FMSH_device *dev, FMSH_callback userFunction)*

描述	* This function is used to set a user listener function. The listener * function is responsible for handling all interrupts that are not * handled by the Driver Kit interrupt handler. This encompasses all

	* error interrupts, general calls, read requests, and receive full * when no receive buffer is available. There is no need to clear any * interrupts in the listener as this is handled automatically by the * Driver Kit interrupt handlers. * A listener must be setup up before using any of the other functions * of the Interrupt API. Note that if the FI2cPs_userIrqHandler * interrupt handler is being used, none of the other Interrupt API * functions can be used with it. This is because they are symbiotic * with the FI2cPs_irqHandler() interrupt handler.
参数	* @param * dev is i2c device handle. * listener is function pointer to user listener function.
返回值	* @return * None.

*46. int FI2cPs_masterBack2Back(FMSH_device *dev, uint16_t *txBuffer, unsigned txLength, uint8_t *rxBuffer, unsigned rxLength, FMSH_callback userFunction)*

描述	* This function initiates an interrupt-driven master back-to-back * transfer. To do this, the I2C must first be properly configured, * enabled and a transmit buffer must be setup which contains the * sequential reads and writes to perform. An associated receive * buffer of suitable size must also be specified when issuing the * transfer. As data is received, it is written to the receive buffer. * The callback function is called (if it is not NULL) when the final * byte is received and there is no more data to send.

	* * A transfer may be stopped at any time by calling FI2cPs_terminate(), * which returns the number of bytes that are sent before the transfer * is interrupted. A terminated transfer's callback function is never * called.
参数	* @param * dev is i2c device handle. * txBuffer is buffer from which to send data * txLength is length of transmit buffer/number of bytes to send * rxBuffer is buffer to write received data to * rxLength is length of receive buffer/number of bytes to receive * callback is function to call when transfer is complete
返回值	* @return * 0 -- if successful * -FMSH_EBUSY -- if the I2C is busy (transfer already in progress)

*47. int FI2cPs_masterTransmit(FMSH_device *dev, uint8_t *buffer, unsigned length, FMSH_callback userFunction)*

描述	* This function initiates an interrupt-driven master transmit * transfer. To do this, the I2C must first be properly configured and * enabled. This function configures a master transmit transfer and * enables the transmit interrupt to keep the transmit FIFO filled. * Upon completion, the callback function is called (if it is not * NULL). * * A transfer may be stopped at any time by calling FI2cPs_terminate(),

	* which returns the number of bytes that are sent before the transfer * is interrupted. A terminated transfer's callback function is never * called.
参数	* @param * dev is i2c device handle. * buffer is buffer from which to send data * length is length of transmit buffer/number of bytes to send * callback is function to call when transfer is complete
返回值	* @return * 0 -- if successful * -FMSH_EBUSY -- if the I2C is busy (transfer already in progress)

*48. int FI2cPs_slaveTransmit(FMSH_device *dev, uint8_t *buffer, unsigned length, FMSH_callback userFunction)*

描述	* This function initiates an interrupt-driven slave transmit transfer. * To do this, the I2C must first be properly configured, enabled and * must also receive a read request (I2c_irq_rd_req) from an I2C * master. This function fills the transmit FIFO and, if there is more * data to send, sets up and enables the transmit interrupts to keep * the FIFO filled. Upon completion, the callback function is called * (if it is not NULL). * * A transfer may be stopped at any time by calling FI2cPs_terminate(), * which returns the number of bytes that were sent before the transfer * was interrupted. A terminated transfer's callback function is never * called.

参数	* @param * dev is i2c device handle. * buffer is buffer from which to send data * length is length of transmit buffer/number of bytes to send * callback is function to call when transfer is complete
返回值	* @return * 0 -- if successful * -FMSH_EBUSY -- if the I2C is busy (transfer already in progress)

*49. int FI2cPs_slaveBulkTransmit(FMSH_device *dev, uint8_t *buffer,unsigned length, FMSH_callback userFunction)*

描述	* This function initiates an interrupt-driven slave transmit transfer. * To do this, the I2C must first be properly configured, enabled and * must also receive a read request (I2c_irq_rd_req) from an I2C * master. This function fills the transmit FIFO and, if there is more * data to send, sets up and enables the transmit interrupts to keep * the FIFO filled. Upon completion, the callback function is called * (if it is not NULL). * * A transfer may be stopped at any time by calling FI2cPs_terminate(), * which returns the number of bytes that were sent before the transfer * was interrupted. A terminated transfer's callback function is never * called.
参数	* @param * dev is i2c device handle. * buffer is buffer from which to send data

	* length is length of transmit buffer/number of bytes to send * callback is function to call when transfer is complete
返回值	* @return * 0 -- if successful * -FMSH_EPROTO -- if a read request was not received

*50. int FI2cPs_masterReceive(FMSH_device *dev, uint8_t *buffer, unsigned length, FMSH_callback userFunction)*

描述	* This function initiates an interrupt-driven master receive transfer. * To do this, the I2C must first be properly configured and enabled. * This function sets up the transmit FIFO to be loaded with read * commands. In parallel, this function sets up and enables the * receive interrupt to fill the buffer from the receive FIFO (the same * number of times as writes to the transmit FIFO). Upon completion, * the callback function is called (if it is not NULL). * * A transfer may be stopped at any time by calling FI2cPs_terminate(), * which returns the number of bytes that were received before the * transfer was interrupted. A terminated transfer's callback function * is never called.
参数	* @param * dev is i2c device handle. * buffer is buffer from which to send data * length is length of transmit buffer/number of bytes to send * callback is function to call when transfer is complete
返回值	* @return

	* 0 -- if successful * -FMSH_EBUSY -- if the I2C is busy (transfer already in progress)

*51. int FI2cPs_slaveReceive(FMSH_device *dev, uint8_t *buffer, unsigned length, FMSH_callback userFunction)*

描述	* This function initiates an interrupt-driven slave receive transfer. * To do this, the I2C must first be properly configured and enabled. * This function sets up and enables the receive interrupt to fill the * buffer from the receive FIFO. Upon completion, the callback * function is called (if it is not NULL). * * A transfer may be stopped at any time by calling FI2cPs_terminate(), * which returns the number of bytes that were received before the * transfer was interrupted. A terminated transfer's callback function * is never called.
参数	* @param * dev is i2c device handle. * buffer is buffer to write received data to * length is length of buffer/max number of bytes to receive * callback is function to call when transfer is complete
返回值	* @return * 0 -- if successful * -FMSH_EBUSY -- if the I2C is busy (transfer already in progress)

*52. int FI2cPs_terminate(FMSH_device *dev)*

描述	* This function terminates the current I2C interrupt-driven transfer

	* in progress, if any. This function must be called to end an * unfinished interrupt-driven transfer as driver instability would * ensue otherwise. * Any data received after calling this function is treated as a new * transfer by the driver. Therefore, it would be prudent to wait * until the next detected stop condition when receiving data in order * to avoid a misalignment between the device and driver.
参数	* @param * dev is i2c device handle.
返回值	* @return * The number of bytes sent/received during the interrupted transfer, * if any.

*53. void FI2cPs_unmaskIrq(FMSH_device *dev, FIicPs_Irq_T interrupts)*

描述	* Unmasks specified I2C interrupt(s).
参数	* @param * dev is i2c device handle. * interrupts is interrupt(s) to enable
返回值	* @return * None.

*54. void FI2cPs_maskIrq(FMSH_device *dev, FIicPs_Irq_T interrupts)*

描述	* Masks specified I2C interrupt(s).
参数	* @param * dev is i2c device handle.

	* interrupts is interrupt(s) to disable
返回值	* @return * none

*55. void FI2cPs_clearIrq(FMSH_device *dev, FIicPs_Irq_T interrupts)*

描述	* Clears specified I2C interrupt(s). Only the following interrupts * can be cleared in this fashion: rx_under, rx_over, tx_over, rd_req, * tx_abrt, rx_done, activity, stop_det, start_det, gen_call. Although * they can be specified, the tx_empty and rd_req interrupts cannot be * cleared using this function.
参数	* @param * dev is i2c device handle. * interrupts is interrupt(s) to clear
返回值	* @return * None

*56. bool FI2cPs_isIrqMasked(FMSH_device *dev, FIicPs_Irq_T interrupt)*

描述	* Returns whether the specified I2C interrupt is masked or not. Only * one interrupt can be specified at a time.
参数	* @param * dev is i2c device handle. * interrupts is interrupt to check
返回值	* @return * true -- interrupt is enabled

	* false -- interrupt is disabled

*57. uint32_t FI2cPs_getIrqMask(FMSH_device *dev)*

描述	* Returns the current interrupt mask. For each bitfield, a value of * '0' indicates that an interrupt is masked while a value of '1' * indicates that an interrupt is enabled.
参数	* @param * dev is i2c device handle.
返回值	* @return * The DW_apb_i2c interrupt mask.

*58. bool FI2cPs_isIrqActive(FMSH_device *dev, FIicPs_Irq_T interrupt)*

描述	* Returns whether an I2C interrupt is active or not, after the masking * stage.
参数	* @param * dev is i2c device handle. * interrupt is interrupt to check
返回值	* @return * true -- irq is active * false -- irq is inactive

*59. bool FI2cPs_isRawIrqActive(FMSH_device *dev, FIicPs_Irq_T interrupt)*

描述	* Returns whether an I2C raw interrupt is active or not, regardless of * masking.
参数	* @param

	* dev is i2c device handle. * interrupt is interrupt to check
返回值	* @return * true -- irq is active * false -- irq is inactive

*60. int FI2cPs_setDmaTxMode(FMSH_device *dev, enum FMSH_dmaMode mode)*

描述	* This function is used to set the DMA mode for transmit transfers. * Possible options are none (disabled), software or hardware * handshaking. For software handshaking, a transmit notifier function * (notifies the DMA that the I2C is ready to accept more data) must * first be set via the FI2cPs_setNotifier_destinationReady() function. * The transmitter empty interrupt is masked for hardware handshaking * and unmasked (and managed) for software handshaking or when the DMA * mode is set to none.
参数	* @param * dev is i2c device handle. * mode is DMA mode to set (none, hw or sw handshaking).
返回值	* @return * 0 -- if successful * -FMSH_ENOSYS -- if device does not have a DMA interface

*61. enum FMSH_dmaMode FI2cPs_getDmaTxMode(FMSH_device *dev)*

描述	* This function returns the current DMA mode for I2C transmit * transfers.

参数	* @param * dev is i2c device handle.
返回值	* @return * The current DMA transmit mode.

*62. int FI2cPs_setDmaRxMode(FMSH_device *dev, enum FMSH_dmaMode mode)*

描述	* This function is used to set the DMA mode for receive transfers. * Possible options are none (disabled), software or hardware * handshaking. For software handshaking, a receive notifier function * (notifies the DMA that the I2C is ready to accept more data) must * first be setup via the FI2cPs_setNotifier_sourceReady() function. * The receiver full interrupt is masked for hardware handshaking and * unmasked for software handshaking or when the DMA mode is set to * none.
参数	* @param * dev is i2c device handle. * mode is DMA mode to set (none, hw or sw handshaking)
返回值	* @return * 0 -- if successful * -FMSH_ENOSYS -- if device does not have a DMA interface

*63. enum FMSH_dmaMode FI2cPs_getDmaRxMode(FMSH_device *dev)*

描述	* This function returns the current DMA mode for I2C transmit * transfers.
参数	* @param

	* dev is i2c device handle.
返回值	* @return * The current DMA transmit mode.

*64. int FI2cPs_setDmaTxLevel(FMSH_device *dev, uint8_t level)*

描述	* This function sets the threshold level at which new data is * requested from the DMA. This is used for DMA hardware handshaking * mode only.
参数	* @param * dev is i2c device handle. * level is DMA request threshold level.
返回值	* @return * 0 -- if successful * -FMSH_ENOSYS -- if device does not have a DMA interface

*65. uint8_t FI2cPs_getDmaTxLevel(FMSH_device *dev)*

描述	* This functions gets the current DMA transmit data threshold level. * This is the FIFO level at which the DMA is requested to send more * data from the I2C.
参数	* @param * dev is i2c device handle.
返回值	* @return * The current DMA transmit data level threshold.

*66. int FI2cPs_setDmaRxLevel(FMSH_device *dev, uint8_t level)*

描述	* This function sets the threshold level at which the DMA is requested * to receive data from the I2C. This is used for DMA hardware * handshaking mode only.
参数	* @param * dev is i2c device handle. * level is DMA request threshold level
返回值	* @return * 0 -- if successful * -FMSH_ENOSYS -- if device does not have a DMA interface

*67. uint8_t FI2cPs_getDmaRxLevel(FMSH_device *dev)*

描述	* This functions gets the current DMA receive data threshold level. * This is the FIFO level at which the DMA is requested to receive from * the I2C.
参数	* @param * dev is i2c device handle.
返回值	* @return * The current DMA receive data level threshold.

*68. int FI2cPs_setNotifier_destinationReady(FMSH_device *dev, FMSH_dmaNotifierFunc funcptr, FDmaPs_T *dmac, unsigned channel)*

描述	* This function sets the user DMA transmit notifier function. This * function is required when the DMA transmit mode is software * handshaking. The I2C driver calls this function at a predefined

	* threshold to request the DMA to send more data to the I2C.
参数	* @param * dev is i2c device handle. * funcptr is called to request more data from the DMA * dmac is associated DW_ahb_dmac device handle * channel is channel number used for the transfer
返回值	* @return * 0 -- if successful * -FMSH_ENOSYS -- if device does not have a DMA interface

*69. int FI2cPs_setNotifier_sourceReady(FMSH_device *dev, FMSH_dmaNotifierFunc funcptr, FDmaPs_T *dmac, unsigned channel)*

描述	* This function sets the user DMA receive notifier function. This * function is required when the DMA receive mode is software * handshaking. The I2C driver calls this function at a predefined * threshold to inform the DMA that data is ready to be read from the * I2C.
参数	* @param * dev is i2c device handle. * funcptr is called to inform the DMA to fetch more data * dmac is associated DMA device handle * channel is channel number used for the transfer
返回值	* @return * 0 -- if successful * -FMSH_ENOSYS -- if device does not have a DMA interface

*70. int FI2cPs_irqHandler(FMSH_device *dev)*

描述	* This function handles and processes I2C interrupts. It works in * conjunction with the Interrupt API and a user listener function * to manage interrupt-driven transfers. When fully using the * Interrupt API, this function should be called whenever a DW_apb_i2c * interrupt occurs. There is an alternate interrupt handler * available, FI2cPs_userIrqHandler(), but this cannot be used in * conjunction with the other Interrupt API functions.
参数	* @param * dev is i2c device handle.
返回值	* @return * true -- an interrupt was processed * false -- no interrupt was processed

*71. int FI2cPs_userIrqHandler(FMSH_device *dev)*

描述	* This function identifies the current highest priority active * interrupt, if any, and forwards it to a user-provided listener * function for processing. This allows a user absolute control over * how each I2C interrupt is processed. * * None of the other Interrupt API functions can be used with this * interrupt handler. This is because they are symbiotic with the * FI2cPs_irqHandler() interrupt handler. All Command and Status API * functions, however, can be used within the user listener function. * This is in contrast to FI2cPs_irqHandler(), where FI2cPs_read(),

	* FI2cPs_write() and FI2cPs_issueRead() cannot be used within the user * listener function.
参数	* @param * dev is i2c device handle.
返回值	* @return * true -- an interrupt was processed * false -- no interrupt was processed

*72. int FI2cPs_flushRxHold(FMSH_device *dev)*

描述	* This functions virtually flushes any data in the hold variable to * the buffer (both in the i2c_Instance_T structure). The 'hold' * variable normally stores up to four data bytes before they are * written to memory (i.e. the user buffer) to optimize bus performace. * Flushing the * (instance->) hold variable only makes sense when the i2c is in * either master-receiver or slave-receiver mode.
参数	* @param * dev is i2c device handle.
返回值	* @return * 0 if successful * -EPERM if the i2c is not in a receive mode (master-rx/slave-rx)

*73. int FI2cPs_autoCompParams(FMSH_device *dev)*

描述	* This function attempts to automatically discover the hardware * component parameters, if this supported by the i2c in question.

	* This is usually controlled by the ADD_ENCODED_PARAMS coreConsultant * parameter.
参数	* @param * dev is i2c device handle.
返回值	* @return * 0 if successful * -ENOSYS function not supported

*74. void FI2cPs_resetInstance(FMSH_device *dev)*

描述	* This function resets/zeros all variables found in the * FIicPs_Instance_T structure, except for FMSH_i2c_statistics.
参数	* @param * dev is i2c device handle.
返回值	* @return * none