Communicating with the FXCore

Overview

The FXCore operates in two primary modes: RUN and PROG. RUN mode is the normal operating mode, this mode will run one of the 16 programs from its internal FLASH memory. PROG mode allows users to write their program to one of the 16 locations in the FXCore or to send a program to the FXCore and have the FXCore execute it.

In RUN mode the FXCore monitors the program select pins, when it detects a change on one or more of them it sends 0 out the DAC data lines, clears the internal delay memory, reads the program from FLASH and places it in the EXEC_RAM block as FLASH is too slow to execute from and finally reads the register initialization data from FLASH and initializes the registers. Once all these steps are complete the program is executed and the DAC outputs are released.

PROG mode allows a user to download an assembled FXCore program along with the register preset values to the FXCore and either program one of the internal FLASH memory program locations with the program and presets or execute the program. FXCore can only write programs to the FLASH, it cannot read back from the FLASH for security. In order to ensure that the program was received and programmed to FLASH properly a checksum is sent along with program data and preset values. This checksum is compared to an internally generated one and if they match the data is saved awaiting FLASH programming or execution.

Note that once program and preset data are sent to the FXCore the user can only run OR write the program to flash but not both. For example a user may send a program and execute it but if they then want to write it to FLASH they will have to resend the program and preset data.

Entering Program Mode

The FXCore starts and normally operates in RUN mode, to enter PRG mode a special ENTR_PRG command must be sent via I2C to the FXCore. The FXCore can only act as an I2C slave device so all transactions must be initiated by an I2C master device.

NOTE: Wait at least 16384 sample periods prior to attempting to enter PRG mode. The FXCore throws away the first 16384 samples from the CODEC to allow it time to settle and start producing valid samples. The FXCore will ignore I2C transactions at this time and attempting to talk to it may cause the internal I2C buffer to overflow so the chip is left in an indeterminate state.

To enter PRG mode the master sends a 3-byte command to the FXCore in the form:
0xA5 0x5A 0x<FXCore I2C address>
If the FXCore had address 0x30 then the sequence would be:
0xA5 0x5A 0x30

The complete I2C communication would look like:
Master sets START condition
Master sends 7-bit I2C address of target FXCore
Master sends write bit
FXCore sends ACK
Master sends 0xA5
FXCore sends ACK
Master sends 0x5A
FXCore sends ACK
Master sends FXCore I2C address as 8-bit value, set MSB to 0
FXCore sends ACK
Master sets STOP condition

At this point the FXCore will be in PRG mode STATE0 waiting for register preset and program data.

It is assumed the reader is familiar with I2C communication and any following examples will not include all the I2C steps.

PROG mode States

While the FXCore is in PRG mode it will be in one of a number of different states. Initially it will be in STATE0 which is an idle state, it is waiting to be told what to do.

STATE0 : Idle state, the initial state entered by the FXCore in PRG mode waiting to be told what to do. When in this state it can accept any command.

STATE1 : Registers received, the FXCore enters this state after it has received the preset values for CREGs, MREGs or SFRs. In this state the FXCore can accept another register preset transfer command, a program transfer command or a RETURN_0 command.

STATE2 : Program received, the FXCore enters this state after it has received a program. In this state the FXCore can accept an EXEC_FROM_RAM, WRITE_PRG or RETURN_0

NOTE: The program must always be sent last as you cannot return to STATE1 from STATE2 and you can only perform an EXEC_FROM_RAM or WRITE_PRG from STATE2 which is after a program has been received. You can send the register preset values in any order and if you choose to not send preset values defaults will be used for the SFRs while the MREGs and CREGs will be set to 0x0. As a result you only need to transfer those registers sets which require presetting.

Command and Data Transfers

Commands and any associated data are transferred as two separate I2C transactions. First the command is sent and if there is data to be sent it is sent in the following transaction. This allows the FXCore to see what data it is about to receive and properly set up internally to receive it. In the cases of CREGs and SFRs all registers must be transferred, for MREGs only as many as necessary need to be transferred but they must start at MREG 0 and must be contiguous. Programs only need to send the actual program even if it is shorter than 1024 instructions. See details in the command section following.

Data is transferred in little-endian format (LS byte sent first) starting at CREG, MREG 0 or program instruction 0. SFRs have a slightly modified format.

A 2 byte checksum is added to the data transfer which is simply the sum of all bytes. This checksum is also sent little-endian.

Commands are sent big-endian or in the order listed in the below table (i.e. for XFER_CREG 0x01 is sent first)

Command Set

The FXCore understands a total of 12 commands:

Command	Hex codes	Available in states	Comments
XFER_CREG	0x01 0x0F	STATE0 STATE1	
XFER_SFR	0x02 0x0B	STATE0 STATE1	See notes on SFR transfer below
XFER_CREG_SFR	0x03 0x1B	STATE0 STATE1	Transfer both CREGs and SFRs in a single transaction. CREGs first, checksum is for all data
XFER_MREG	0x04 0xXX	STATE0 STATE1	Since only the number of MREGs that need to be preset are transferred the second byte contains N-1 the number of registers being transferred. I.e. if only 7 registers (0 – 6) are being transferred then 0xXX would be 0x06
XFER_PRG	0x08 0xXX to 0x0B 0xFF	STATE0 STATE1	This command is basically 0x0800 + (number of instructions – 1) so if a program is 233 (0xE9) long then we add 0x0800 + (0xE9 – 1) = 0x08E8 and

			the hex codes are 0x08 0xE8
WRITE_PRG	0x0C 0x0X	STATE2	Write the program to location X, valid range is 0 - F
EXEC_FROM_RAM	0x0D 0x00	STATE2	Execute the program, end program with a RETURN_0 command
RETURN_0	0x0E 0x00	STATE1 STATE2	Return to STATE0
EXIT_PRG	0x5A 0xA5	STATE0	Exit PRG mode and return to RUN mode
ENTER_PRG	0xA5 0x5A 0xXX	From RUN mode	Enters PRG mode, 0xXX is I2C address of the FXCore

SFR Data

While the CREG, MREG and program data is straight forward (send lowest byte of register 0 or instruction 0 first, etc.) SFRs are more complex. They are not contiguous in memory and have various sizes resulting in 50 bytes of data to transfer. The SFR data is there for formatted in the following manner starting at the LSBs of the lowest word as we are packing the data into 11 32-bit words:

Word	Bits 31 … 0	Notes
0	0:0:P5<4:0>:P4<4:0>:P3<4:0>:P2<4:0>:P1<4:0>:P0<4:0>	POT smoothing coefficients
1	LFO0 frequency coefficient	
2	LFO1 frequency coefficient	
3	LFO2 frequency coefficient	
4	LFO3 frequency coefficient	
5	RAMP 0 frequency coefficient	
6	RAMP 1 frequency coefficient	
7	MAX tap tempo count	
8	Starting tap tempo count	
9	Tap sticky count<15:0>: Tap debounce count<15:0>	
10	Switch debounce count<15:0> : Program debounce count<15:0>	
11	Overflow LED time<15:0>:00000000000000:USR1:USR0	

Typical Steps to Program a Location

Keep in mind that each of the following steps is a separate I2C transaction with start, address, etc.

Send an ENTER_PRG command with the I2C address of the FXCore in the command
Send a XFER_CREG command
Send the 16-CREGs plus the-2 byte checksum
Send a XFER_MREG command with the number of registers – 1 to transfer in the command

Send MREGs plus the 2-byte checksum
Send a XFER_SFR command
Send the 48-bytes of SFR data plus the 2-byte checksum
Send a XFER_PRG command with the number of instructions – 1 to transfer in the command
Send the instructions plus the 2-byte checksum

You can now send an EXEC_FROM_RAM to run the program, a WRITE_PRG command to write the program and preset values to a FLASH location or a RETURN_0 command to return to STATE0.

If you execute a WRITE_PRG you will need to wait until the write to FLASH completes before issuing any additional commands such as a RETURN_0. Wait at least 100mS, the FXCore assembler waits 200mS to be extra safe between WRITE_PRG and RETURN_0.

Status Word

The master I2C device can read a status word from the FXCore in state 0, 1 or 2 but not while executing an EXEC_FROM_RAM. The read will return 12-bytes formatted as 4 byte values, two 16-bit values and one 32-bit value. The 16-bit and 32-bit values are sent little-endian.

Byte 0 – Current transfer state:
Bits <7:5> are always 0
Bit <4> is 1 if a program was successfully received.
Bit <3> is 1 if at least 1 of the register presets was receive successfully
Bit <2> is 1 if the MREGs have been received
Bit <1> is 1 if the SFRs have been received
Bit <0> is 1 if the CREGs have been received

Byte 1 – Command status:
0xFF – Unknown command
0xFE – Command length error, all commands are 2 or 3 bytes
0xFD – Parameter out of range, generally from setting an invalid program slot number or count
0xFC – Command not allowed in current state
0x80 – Calculated checksum did not match received checksum
0x4X – Unknown program transfer error, state reset to STATE0
0x1F, 0x2F, 0x3F – FLASH erase error
0x1X, 0x2X, 0x3X (X any value but F) – FLASH write error
0x00 – Command was successful

Byte 2 CMDH and byte 3 CMDL – Last command received
Last command high and low bytes received from host

Bytes 4 and 5 – Program slot status
A 1 in a bit position indicates that slot has a program in it

Bytes 6 and 7 – Device ID
16-bit device ID

Bytes 8 – 11 – Device serial number
32-bit serial number, set at the fab in production

Additional Information

As in any communications document examples are a great help but trying to do a complete example within this document would be long and detailed task where a single typo could cause confusion. In place of this the IDE version of FXCore ASM tells the user in the left information window what it is doing, the number of bytes transferred and the calculated checksum.