PRELIMINARY

IN-SYSTEM PROGRAMMING

INTRODUCTION

In-system-programming is the capability to program or re-program a chip in place after it has been mounted on a circuit board. The benefits of in-system programming have been well documented. It speeds up product development, minimizes the risk of over stocking of out-dated parts, and allows field upgrade and bug fix.

With ISP, developing a micro-controller system is simpler and faster. During program development, codes can be updated with the micro-controller in the actual system board. Without ISP, the micro-controller has to be removed from the board and plugged into a separate programmer before it can be programmed. The controller can be program and re-program many times, using in-system-programming feature, until the design functions correctly. It saves a lot of time. Also, the micro-controller can be soldered on the board just like the actual production system. No special sockets are needed. It makes debugging timing or noise sensitive designs a lot easier.

For manufacturing, there are many benefits too. System boards can be built with the controller mounted, without sockets, before the design is finalized to meet time-to-market requirement. Additional information can be added at end of production cycles, information that may not be available otherwise, such as, vendor ID. But, one of the major benefits is to reduce the risk of over stocking out-dated pre-programmed devices. Because the controller can be programmed or re-programmed while in system, the parts do not have to be pre-programmed. And pre-programmed parts can be re-programmed if revisions or upgrades become necessary. Finally, with ISP, the parts can be easily upgraded or fixed in the field.

The ISP solution from Scenix is a proprietary system but uses the two pins, clock input (OSC1) and clock output (OSC2), for crystal oscillator. It does not require an expensive JTAG tester and, in most applications, does not need any additional hardware to isolate the ISP circuitry from other hardware on the system board. The reason is that OSC1 and OSC2 are normally connected to passive components such as capacitors, resistors, or crystals and can not be damaged with high programming voltages and would not interfere with ISP programming signals. The Scenix solution does not use an expensive tester, does not add pins to the micro-controller and does not need extra hardware. It reduces component cost and real estate requirement on the circuit board.

In-System-Programming

The in-system-programming (ISP) mechanism allows SX to be programmed in the system board without removing the SX. In most cases, no modifications to the system board are needed. Only two pins, OSC1 and OSC2 are used. OSC1 is used to supply power (Vpp = 12.5V) for EEPROM programming. OSC2 is the serial data I/O for writing data to EEPROM and reading data from EEPROM.

There are three stages of in-system-programming: Entering ISP, Programming ISP, and Exiting ISP.

Entering ISP:

The method to enter ISP depends on the oscillator mode selected. SX has three oscillator modes: crystal (XTAL), external RC (XRC), and internal RC (IRC). The three methods are slightly different but the basic idea is the same: signal SX to shut off the oscillator at OSC1 so that a high voltage can be applied at OSC1 for EEPROM programming (Fig. 1).

OSC2 is the communication pin for ISP. To tell SX to shut off the oscillator, drive OSC2 low. To avoid the oscillator from being inadvertently shut off, a noise filtering logic is used. OSC2 has to be low for nine consecutive OSC1 clocks to signal the beginning of ISP.

Use the following steps to start ISP:

First, force OSC1 low. Then force OSC2 low. With OSC2 low, toggle OSC1 nine times. These nine OSC1 clocks, driven by the ISP programming module (e.g. ISP programming module from Parallax, Inc.), clears OSC_EN bit in the DEBUG register and disables the on chip oscillator circuit.

With the oscillator circuit disabled, OSC1 can be safely driven to Vpp level without any contention. The last step is to raise OSC1 to VPP. When VPP appears at OSC1, the internal RC oscillator (freq. = 128K) is enabled and becomes the clock for ISP state machine. The ISP logic is now ready for programming.

Programming ISP:

When ISP programming stage is entered, SX could be in the middle of executing a program. To prevent any accidental damage to the system board, the first thing the ISP logic does is to reset the chip. When reset is done, ISP logic executes the in-system-program protocol. It is a “self aligned” protocol. One pin, OSC2, is used for both serial data I/O and a “synchronization clock”. No separate clock pin is needed. OSC2 is implemented as open drain with an internal pull up.

The ISP programming protocol is made up of a stream of ISP frames at OSC2. In the current scheme, a frame has 17 cycles and a cycle has four clocks (Fig.2).

The first cycle of a frame is the synchronization cycle. It signals the beginning of a frame. It is followed by four cycles for entering ISP command. These are the command cycles. The command is shifted in serially and it specifies an ISP operation. For example, write data to EEPROM or read data from EEPROM. The command phase is followed by twelve cycles for data. These are the data cycles. Depending on whether the ISP command is to write data or read data, EEPROM data is either serially driven onto OSC2 by the external programming module (write case) or by the chip (read case). In either case, data is valid on the raising edge of fourth clock (clock 3).

There are four clocks in a cycle. On the first clock (clock0), nobody drives OSC2. Hence, the open drain OSC2 is pulled high by the internal pull up of OSC2. On second clock (clock 1), the chip drives OSC2 low. This is the synchronization pulse. The ISP programming module uses this pulse and the synchronization cycle to align data for writing to or reading from the chip. The third and fourth clocks (clock2 and clock3) are for data. Depending on the direction of data transfer, the external module or the chip can drive data during these two clocks. The data is valid and sampled on the rising edge of clock3.

One special note on the operation is that the encoding for the “no operation” command, NOP, must be all one’s. This is important because once entering ISP programming stage, the ISP logic on chip will start looking for command to execute. But, the external ISP programming module takes some time to align its internal clock to the synchronization cycle and synchronization pulses so that command can be driven onto OSC2 at the right cycles (command cycles) and at the right clocks (clock2 and clock3). Since OSC2 is open drain, ISP programming module, by not driving OSC2 (and thus no synchronization is needed), effectively drives the NOP command. When the ISP programming module aligned its internal clock with the synchronization pulses, it can safely and correctly drive the command and write or read data.

Currently, nine commands are defined:

Name	Code	Description
NOP	0xf	No operation.
ERASE	0x0	Erase ALL EEPROM locations.
READ DEV	0x1	Read SX features available for the device.
READ FuseX	0x2	Read FuseX.
PROG FuseX	0x3	Write FuseX. Need to execute the LOAD command first to specify the data to be programmed.
LOAD	0x4	Specify the data to be programmed.
PROG	0x5	Write data to EEPROM other that FuseX. Need to execute

		the LOAD command first to specify the data to be programmed.
READ	0x6	Read EEPROM data other than FuseX.
INC	0x7	Increment EEPROM address by one.

However, the scheme does not put any restriction on the number of commands. New commands can be added as needed.

The on chip ISP logic is consisted of three major blocks: command shift register, command decode and data shift register. Command is shifted in serially and decoded once all command bits are shifted in place. In a write data operation, data to be written is first loaded in the data register by shifting the data in serially, using the LOAD command. It is then written to the EEPROM in parallel using the PROG command. To control the length of data write time, simply execute the PROG command continuously for the desired number of times. In a read data operation, use the READ to load the data shift register with EEPROM data in parallel mode and then to shift the data out serially. For example, to program one location in EEPROM, following commands need to be executed:

1. ERASE (First, erase the memory. Issue the ERASE command several time to allow sufficient time to erase. For example, if the erase time is x msec and each frame takes 0.53msec (1/128k * 4 * 17), issue ERASE command x/0.53 times)
2. LOAD
3. PROG (Issue PROG command several times to allow sufficient time to program. For example, if the program time is x msec and each frame takes 0.53msec (1/128k * 4 * 17), issue PROG command x/0.53 times)

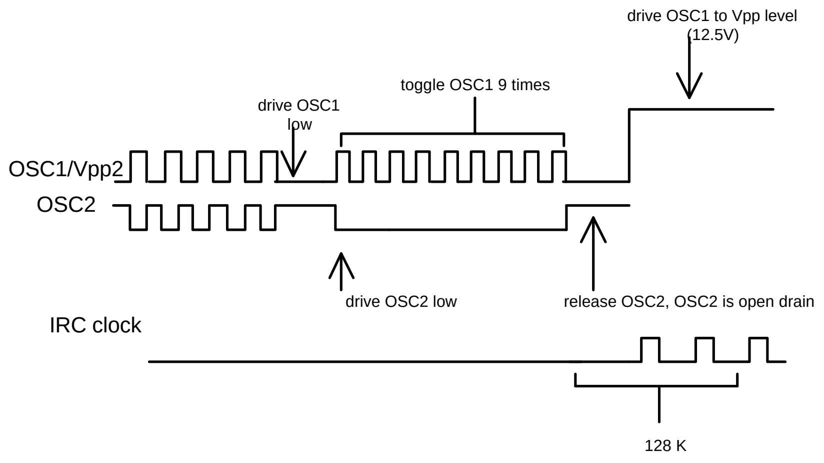
Exiting ISP:

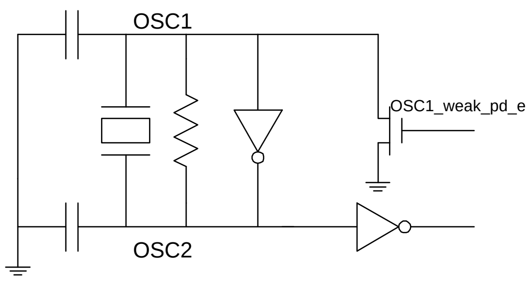
When ISP programming is done, a method is needed to safely exit the ISP mode without accidentally damaging the system board. A scheme is needed to let both the on chip ISP logic and the off chip ISP programming module know that it is safe to exit ISP.

The scheme chosen is for the off chip ISP programming module to drop OSC1 from Vpp to low. This signals its intention to exit ISP mode. The ISP mode is exited on the first rising clock edge after the synchronization cycle. At this time, the on chip ISP logic generates a reset pulse to reset the chip and terminate the ISP mode. Before this point, both on chip and off chip ISP logic need to observe all ISP protocol.

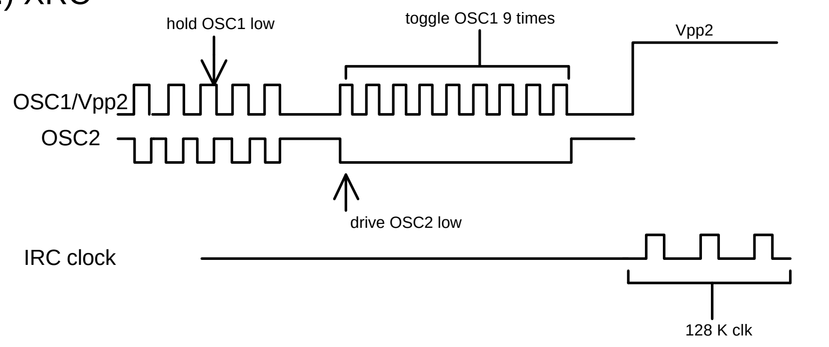
3 OSC modes: XTAL, XRC, IRC

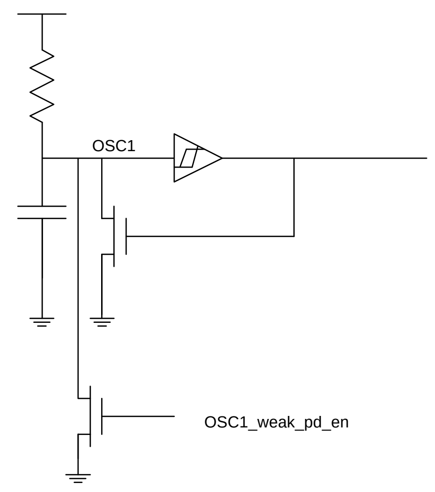
1.) XTAL





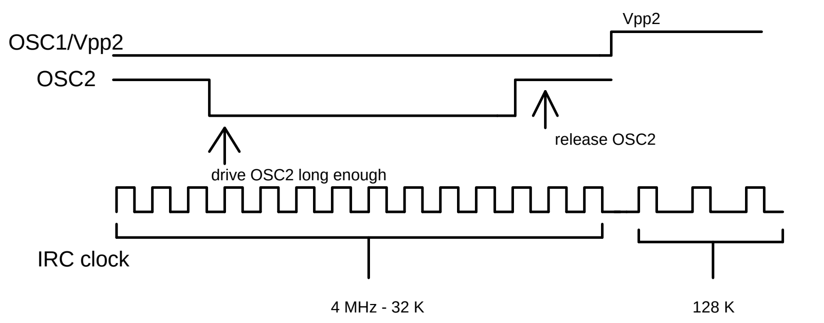
2.) XRC

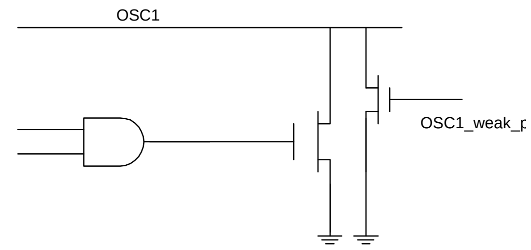




OSC2 (not used)

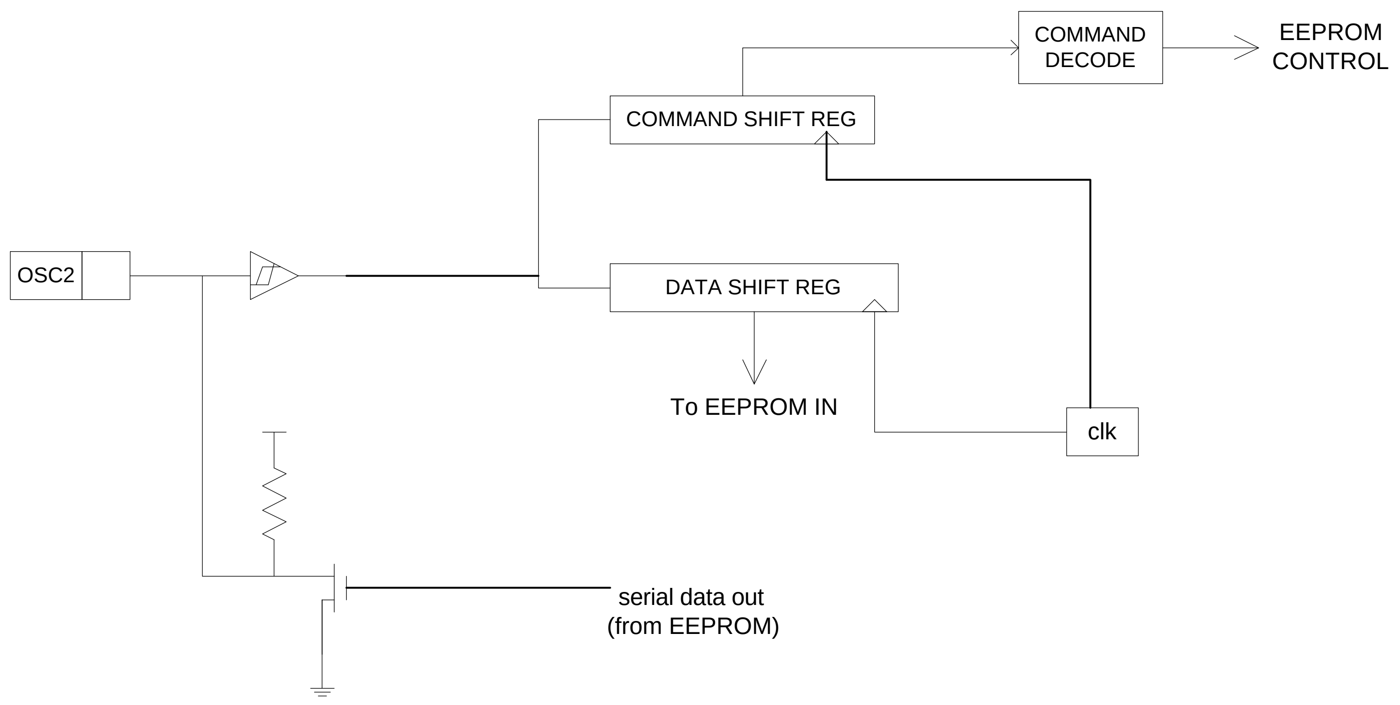
3.) IRC



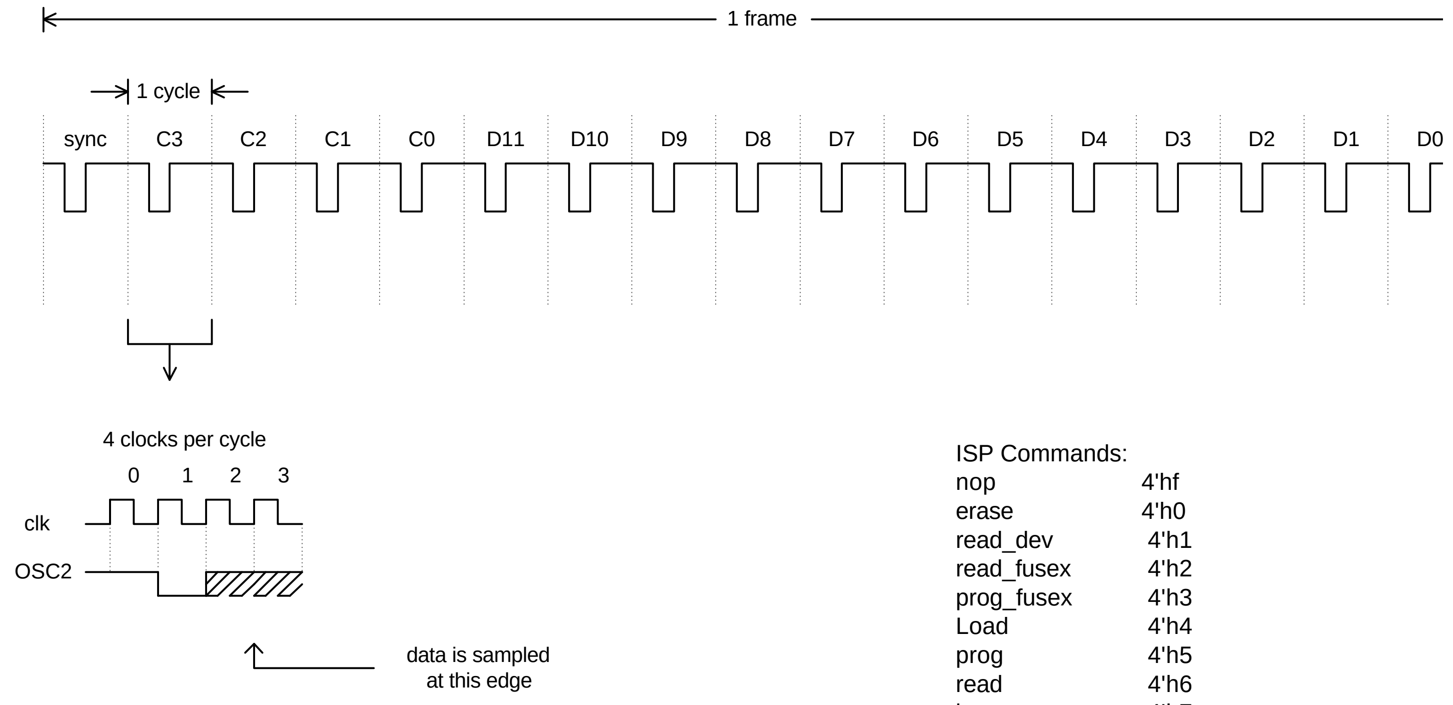


OSC2 (not used)

Fig. 1



In System Programming Protocol



ISP Commands:

nop	4'hf
erase	4'h0
read_dev	4'h1
read_fusex	4'h2
prog_fusex	4'h3
Load	4'h4
prog	4'h5
read	4'h6
inc	4'h7

In command phase, command is driven in.
Two cases in data phase:
a. write : EEPROM data is driven in
b. read : EEPROM data is driven out

Fig. 2