# Solid-State Drive Driver Package 4.18.0 Kernel-Based Linux

Version 01.01

Date 2019/12/11

## Introduction

* 1. The Solid-State Drive (SSD) Driver Package consists of a Linux block device driver and a lock program.
  2. The driver was built and tested on a Linux version 4.18.0-041800-generic x64 based Ubuntu System
  3. The driver is for use with WinSystems, Inc. Single Board Computers (SBC) that provide battery-backed SRAM. This driver will allow this memory to appear as a disk. This will allow a user to load the driver module, make a file system, mount it, and create a file.

The following WinSystems, Inc. products incorporate a battery-backed SRAM device:

PCM-VDX-512-2

EPX-C380-S/D

EBC-C413

PCM-C418

* 1. This driver is provided on an 'as-is' basis and no warranty as to usability or fitness of purpose is inferred or claimed.
  2. WinSystems, Inc. does not provide support for the modification of this driver. Customer application specific queries can be sent to: [support@winsystems.com](mailto:support@winsystems.com).
  3. This work is provided under the terms of the MIT General Public License (GPL).

## Installations and Build

* 1. The device driver and lock program are provided in source code form as a compressed zipped folder. This document assumes the source code is placed at the directory */usr/src*.
  2. Any user may build the code using the included *makefile*. Installation of the device drivers requires root privileges.
  3. The MAJOR device number for this device is allocated dynamically by the OS, however this can be overridden by editing the file ssd.c, and changing the value of the variable *ssd\_init\_major.*
  4. The SSD driver kernel module and the lock program can be built using the following steps:

1. Extract the contents of the device driver source code archive (TAR) into a directory (for purposes of this document, this will be in the */usr/src* directory). After extracting the contents of the archive file, the new directory *winsys\_ssd* will have been created (*/usr/src/winsys\_ssd*). The */usr/src/winsys\_ssd* directory will contain a pair of items: *ssd\_linux\_2.0.pdf*, and a directory named *source*.
2. Open a BASH terminal window in the directory */usr/src/winsys\_ssd/source*. From the command line of the BASH terminal window, execute the command: *sudo make all* [[1]](#footnote-1),[[2]](#footnote-2)
3. When the build completes, there will be a number of new files in the */usr/sr/winsys\_ssd/source* directory. The file *ssd.ko* is the Linux kernel module, which can be dynamically installed or removed from a running system. The file *lock* is a Linux executable program that is used to write protect the contents of the SSD.
4. Install the driver into it’s runtime location by executing the command: *sudo make install*
   1. The SSD driver kernel module may be installed into a running OS with the provided *ssd\_load* script, or it may be installed manually. In either case, the command for installing the SSD driver module is:

*modprobe ssd.ko io=<SSD\_IO\_ADDRESS>*

Where <SSD\_IO\_ADDRESS> is the IO port address for the SSD device. The table below contains the IO port addresses for the WinSystems products that support this SSD:

|  |  |
| --- | --- |
| **WinSystem Product** | **Port I/O Address** |
| PCM-VDX-512-2 | 0x220 |
| EPX-C380-*X* | 0x210 |
| EPX-C414 | 0x210 |

Please note that the *ssd\_load* script defaults to a port I/O address of 0x210. If *ssd\_load* is to be used on a PCM-VDX-512 system, the script must be edited to change the port I/O address from 0x210 to 0x220.

* 1. When the SSD module is installed, a new device will have been created in the system device directory */dev*. The new device will be device *ssd*, and the full path to it will be */dev/ssd*.

Before using the */dev/ssd* device, it must have a file system instantiated on it. To instantiate the file system on /dev/ssd, execute the following set of commands[[3]](#footnote-3):

*sudo /usr/src/winsys\_ssd/source/lock off*

*sudo mkfs –t <TYPE> /dev/ssd*

Where <TYPE> is the type of file system to be instantiated on the SSD device. Valid file system types include ext2, ext3, ext4, msdos. Please reference the man page for mkfs for a complete list of supported file system types.

* 1. After a file system type has been instantiated on the */dev/ssd* device, it will need to be mounted before it can be used by applications running on the platform. To mount the */dev/ssd* device, create a directory on the system where the user has the correct permission to create a directory. For purposes of this document, we will assume that the base directory is */usr/src/winsys\_ssd*, and the new directory is *mnt\_pt*. From a BASH terminal window, opened in the directory /usr/src/winsys\_ssd, execute the following command to mount */dev/ssd* to the directory *mnt\_pt*:

*sudo mount /dev/ssd mnt\_pt*

## 3 Usage

3.1 The device driver will create a block accessible disk device that maps the battery-backed SRAM to a disk type device. The operating system will be able to use this block device as if it were any other disk device. Files may be copied to the device, and application programs may open files on the SSD for read and write access using the Linux standard I/O libraries. No special application interface is required once the SSD drive is mounted[[4]](#footnote-4).

3.2 The provided “lock” application that is a deliverable from the build process is a simple program that allows the SSD device to be write protected. When the device driver loads, and the SSD file system is mounted, it will default to being in the write protected mode. While in the write-protect mode, the SSD device will permit read-only access.

To disable SSD write protect: *./lock off*

To enable SSD write protect: *./lock on*

## 4 Hardware Implementation

Access to the battery backed SRAM is through a set of 5 I/O port mapped addresses. Depending on the platform, the base address of these 4 registers is either 0x210 or 0x220.

|  |  |
| --- | --- |
| Register Name | Register Offset |
| MSB Address Register (A23:A16 of access address) | 0 |
| NSB Address Register (A15:A8 of access address) | 1 |
| Data Access Register A (D7:D0, auto address increment) | 2 |
| Data Access Register B (D7:D0, no auto address increment) | 3 |
| Write Protect Register | 4 |

In order to access a specific offset for read or write purposes in the battery backed SRAM array, it is necessary to write the upper 8 bits of the 24 bit offset to the MSB Address Register, and to write the middle 8 bits of the 24 bit offset to the NSB Address Register. When the NSB register is written, it automatically zeros the LSB Address Register (not directly accessible to the user). The LSB Address Register is then auto-incremented by reading Data Access Register A until the address formed by the MSB, NSB, and LSB are the desired offset. When the combination of the MSB, NSB, and LSB form the desired array offset, data may be written to Data Access Register A (the LSB will auto-increment), or the data may be written to Data Register B (the LSB will not auto-increment).

When using Data Address Register B to access data at a specified offset in the battery backed SRAM array, the LSB of the offset must be incremented to the specified offset by performing reads of Data Access Register A until the LSB contains the correct offset.

The Write Protect Register at offset 4 is used to enable and disable write access to the battery backed SRAM array. When Bit 0 of the Write Protect Register is clear, the array is write protected. When Bit 0 of the Write Protect Register is set, the array is not write protected.

# Appendix A Makefile Targets

This section of the manual will detail the make targets supported by the SSD makefile.

|  |  |
| --- | --- |
| Make Target | Make Result |
| install | Installs the kernel driver in the kernel module directory and creates driver dependancies |
| uninstall | Removes the driver from the kernel module directory |
| lock | Builds the *lock* application program |
| clean | Removes objects created by the build |
| spotless | Forcibly removes all build artifacts |

1. The all target for the make file will attempt to install the produced driver kernel module for the SSD. Installation of kernel modules requires root privileges. [↑](#footnote-ref-1)
2. The *makefile* contains additional targets. For instance, running *make* with no target will attempt to build the *default* target. Appendix A contains a list of the make targets. [↑](#footnote-ref-2)
3. The commands to instantiate a file system on the SSD device assume the driver module has already been loaded either by using the *ssd\_load* script, or manually. [↑](#footnote-ref-3)
4. The SSD block device driver was developed and implemented to be friendly to concurrent device usage by multiple SW threads, however it is strongly suggested that its use be limited to a single process/thread. [↑](#footnote-ref-4)