XLR8 User Manual

FPGA
DEVELOPMENT BOARD
Part Number: XLR8R22M08V5U0DI

February 14, 2018
<table>
<thead>
<tr>
<th>Version</th>
<th>DATE</th>
<th>AUTHOR</th>
<th>CHANGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>April 3, 2016</td>
<td>mwebber</td>
<td>Initial Version</td>
</tr>
<tr>
<td>1.1</td>
<td>Dec 14, 2016</td>
<td>jpeterson</td>
<td>Added Best Practices section for SPI</td>
</tr>
<tr>
<td>1.2</td>
<td>Feb 1, 2017</td>
<td>hjuedes</td>
<td>Updated 3.3 Reconfigurability Instructions</td>
</tr>
<tr>
<td>1.3</td>
<td>Feb 8, 2017</td>
<td>hjuedes</td>
<td>Updated Register Summary description</td>
</tr>
<tr>
<td>1.4</td>
<td>Apr 11, 2017</td>
<td>hjuedes</td>
<td>Updated board schematics</td>
</tr>
<tr>
<td>1.5</td>
<td>Feb 14, 2018</td>
<td>Hjuedes</td>
<td>Updated Register Summary and Schematics</td>
</tr>
</tbody>
</table>
# Table of Contents

1 Overview ........................................................................................................... 4

2 Usage ................................................................................................................ 4

3 Differences between XLR8 and Arduino Uno/Sparkfun RedBoard ................. 4

3.1 USB ................................................................................................................. 4

3.2 Xcelerator Blocks (XBs) .............................................................................. 5

3.3 Reconfigurability ......................................................................................... 5

3.3.1 Connecting to Ground ................................................................................ 5

3.4 I/O .................................................................................................................. 6

3.5 ADC .............................................................................................................. 7

3.6 Bootloader and ICSP header ....................................................................... 7

3.7 I2C / Two Wire Interface (TWI) ................................................................. 8

3.8 UART ............................................................................................................ 8

3.9 SPI ................................................................................................................ 8

3.9.1 SPI Best Practices ..................................................................................... 8

3.10 Analog Compare ....................................................................................... 9

3.11 EEPROM ................................................................................................... 10

3.12 Power ........................................................................................................ 10

3.13 Pin13 LED .................................................................................................. 10

3.14 JTAG Interface .......................................................................................... 10

4 ATmega328p features not implemented .......................................................... 10

4.1 Fuses .......................................................................................................... 11

4.2 Power Reduction ....................................................................................... 11

5 Xcelerator Blocks (XBs) .................................................................................. 11

5.1 Floating Point ............................................................................................. 12

5.2 Servo Control ............................................................................................. 12

5.3 NeoPixel Control ....................................................................................... 12

6 Register Summary ........................................................................................... 13

6.1 XLR8 and XB Register Descriptions ........................................................... 18

6.1.1 XLR8VERL, XLR8VERH, XLR8VERT – Version Number Registers ............ 18

6.1.2 FCFGCID – Chip ID Register .................................................................. 18

6.1.3 CLKSPD – Clock Speed Register ............................................................ 18

6.1.4 XICR, XIFR, XMSK - XLR8 IRQ Registers .............................................. 19

6.1.5 OX8ICR, OX8IFR, OX8MSK - OpenXLR8 Pin Change Interrupts .......... 19

6.1.6 FCFGDAT, FCFGSTS, FCFGCTL – FPGA Reconfiguration Registers ........ 20

6.1.7 SVPWH, SVPWL, SVCR – Servo XB Registers ..................................... 20

6.1.8 NEOO2, NEOO1, NEOO0, NEOCR – NeoPixel XB Registers .................... 21

6.1.9 XFCTRL, XFRSTAT, XFR0, XFR1, XFR2, XFR3 – Floating Point XB Registers 22

6.1.10 XLR8ADCR – XLR8 ADC Control Register ......................................... 22

6.1.11 XLR8PID – XLR8 PID ......................................................................... 22

6.1.12 XLR8Quad – XLR8 Quadrature ............................................................. 23

7 Schematics and Layout .................................................................................... 24

8 Credits ............................................................................................................ 26

---

Copyright 2018 Alorium Technology
1 Overview
XLR8 is an FPGA-based application accelerator and development board that has been specifically designed to look, feel, and act like a standard Arduino. It is programmed with the popular and easy to use Arduino IDE. The heart of XLR8 is an FPGA chip that is configured with an ATmega328 microcontroller clone as well as additional accelerator functions.

XLR8 provides an option for Arduino developers to achieve significantly improved performance in the same physical footprint and using the same tool chain as standard Arduino Uno and other similar Arduino compatible boards, even when incorporating custom hardware functions via accelerator blocks.

2 Usage
XLR8 can be programmed with Arduino sketches from the Arduino IDE by selecting “Arduino Uno” as the target board, in the same way as the Uno. The pin headers are compliant with the Arduino Uno R3 layout. It has the same analog pins, the same digital pins, and the same pins with PWM functionality as the Uno. So, using it as a clone of the Uno is a simple as using an Uno. The real fun starts after installing our Arduino Board package as described in the instructions at https://github.com/AloriumTechnology/Arduino_Boards, enabling advanced features such as doubling the microcontroller’s clock speed or utilizing Xcelerator Blocks (XBs). But you don’t need to do that until you are ready.

3 Differences between XLR8 and Arduino Uno/Sparkfun RedBoard

3.1 USB
Similar to the Sparkfun RedBoard, XLR8 uses a USB mini-B connector and an FTDI chip to do the USB-to-Serial conversion, which is slightly different than the Uno. If you haven’t already, you’ll likely need to install FTDI drivers. Note that this is not needed and should not be done if you are using Mac OS El Capitan or later. The OS will have the correct drivers.

Sparkfun has a great tutorial showing just how to do that https://learn.sparkfun.com/tutorials/how-to-install-ftdi-drivers). We have noticed for the Mac that Sparkfun’s tutorial doesn’t mention restarting your computer after installing the driver, but we've generally needed to do that and everything has worked great afterwards. We have also noticed that the FTDI driver in older Linux kernels (specifically, version 2.6 or older) does not appear to support the newer FTDI chips that are used on XLR8, Sparkfun’s RedBoard, and many other products. This can be fixed by upgrading your Linux kernel to version 3.10 or newer.
3.2 Xcelerator Blocks (XBs)

Xcelerator Blocks give XLR8 a performance advantage through a combination of custom hardware in the FPGA fabric along with a software library that is able to communicate with that hardware and make it easy for users to take advantage of the performance. The software libraries are available on our github site (https://github.com/AloriumTechnology), but it is easy to install them without even going to github. In the Arduino IDE, go to the menu Sketch -> Include Library -> Manage Libraries, which will open the Library Manager in a new window. Enter XLR8 in the search bar and you will find the entries for the various XLR8 libraries available. Click on the desired library and an Install button will appear for it.

3.3 Reconfigurability

One of the most awesome things about XLR8 is its re-configurability. XLR8 is able to hold two different FPGA images. Image 1 can be reconfigured from the Arduino IDE to take advantage of increased functionality as new XBs are introduced. The reconfiguration files are obtained by installing our Arduino Board package as described in the instructions at https://github.com/AloriumTechnology/Arduino_Boards. Image 0 is never changed and is typically unused unless the primary image 1 becomes corrupted. If necessary, a “factory reset” of XLR8 can be performed by grounding the FPGA side of R51 (cfg_sel) while applying power to the board. It only takes a momentary grounding to cause this to happen. The factory image is then loaded. Any loss of power to the board will still cause the corrupted image to be reloaded, however. Typically, the user will want to reburn a known-good image into image 1 before proceeding.

3.3.1 Connecting to Ground
3.4 I/O

The FPGA at the heart of XLR8 is a 3.3v device. However, unlike other 3.3v boards, we've made a concerted effort to make XLR8 as compatible as possible with the 5v Arduino Uno. We've done this by adding level shifting and protection circuitry to the I/O so that existing shields can be used without fear of damaging the board. The digital I/O’s (RX, TX, and D2-D13) can both drive and receive 5V signals providing great compatibility with all shields. The analog I/O’s (A0-A5) can tolerate both analog and digital inputs up to 5.0V, and as digital outputs can drive up to 3.3V and that should work fine for the majority of shields and other circuits. XLR8 provides 5.0V on the IOREF pin for newer shields that use that feature.

While ATmega328p inputs can enable or disable the internal weak pullup resistors, in XLR8, the pullups differ by pin as follows:

- RX, TX, D2-D13: Strong pullups on XLR8 board (1K ohms) are always enabled. The initial value of the PINB and PIND registers is therefore 0x3F and 0xFF respectively.
- A0-A5: No pullups
- SDA, SCL: Strong pullups on XLR8 board (1K ohms) are included. When using Arduino libraries, Wire.begin() enables the pullups for the I2C case (assuming the PUD bit of the MCUCR (0x35) register is not set) without any additional work required from the user. If desired, disabling the pullups in Arduino is as simple as doing both
digitalWrite(SDA, LOW); and digitalWrite(SCL, LOW). Outside of the Arduino environment the pullups are enabled when PUD is low, TWEN is high, and both PORTC4 and PORTC5 are high.

### 3.5 ADC

The ADC used in XLR8 is higher performing in both speed and resolution than the ADC found in the ATMEGA328p. By default, doing an analogRead will give the same speed and 10 bit resolution as an Arduino Uno. By using the XLR8ADC library ([https://github.com/AloriumTechnology/XLR8ADC](https://github.com/AloriumTechnology/XLR8ADC)) you can get 12 bit resolution. Further enhancements will be available in the future.

One ADC difference between XLR8 and Arduino Uno/Sparkfun RedBoard occurs when running from USB power. The USB may supply a voltage that is somewhat lower than 5V and because the Uno and RedBoard use this for the default analog reference it results in higher ADC readings compared both to what would be expected and to what is measured when powered from the barrel connector. The XLR8 board does not suffer from this issue; it will give the same ADC reading whether powered from the barrel or from USB. Some shields, however, can be deceiving. If an analog voltage that the shield is creating drops along with the 5V USB power, it will appear on the Uno/Redboard that the ADC readings stay somewhat constant while on XLR8 the ADC readings will be reduced. If it is desired to have XLR8 vary the reference voltage with the USB power similar to an Uno/Redboard, this can be accomplished quite easily by wiring from the 5V header pin to the AREF header pin and using Arduino’s analogReference(EXTERNAL);

While the ATMEGA328p is able to switch its ADC reference voltage between internal 5V, internal 1.1V, and external references, XLR8 uses circuitry on the board to make this selection. From the user’s point of view, there shouldn’t be a noticeable difference. To ensure that XLR8 can tolerate ADC inputs up to 5V, the analog inputs run through an op-amp and voltage divider circuit before entering the FPGA. Again, from the user’s point of view, there shouldn’t be a noticeable.

While the ADC channel selection is the same for measuring the primary six analog inputs (ADMUX=0000 through 0101), there are differences in the other channels. Channel 6 (ADMUX=0110) reads from the voltage divider created by R30 and R59, and should be around the max ADC value (1023 when in 10 bit mode) if the 5V supply is at or above 5.0V, but may be slightly less if the 5V supply is lower. Channel 7, and the temperature sensor are not implemented (ADMUX=0111 and 1000). Using the ADC to read the bandgap (ADMUX=1110) does not actually do a measurement but returns a calculated value equivalent to 1.1/Aref. Using the ADC to read ground (ADMUX=1111) does not actually do a measurement and instead returns a fixed value of 0.

### 3.6 Bootloader and ICSP header

XLR8 uses the Optiboot bootloader that is used by the Arduino Uno, with a slight modification that allows it to run correctly at different CPU speeds. As an extra indication of XLR8’s current CPU speed, you may notice when running at 32MHz that the beginning of an upload sequence
blinks the pin 13 LED three times rather quickly just like an Arduino Uno, while at 16MHz you get two slightly slower blinks. This bootloader is hardcoded into the design and cannot be changed by the user. If you have a need for something different in the bootloader, we’d be interested to hear about it. Consistent with using Optiboot, the BOOTRST “fuse” is hardcoded to zero (programmed) and the IVSEL and IVCE bits of the MCUCR (0x35) register are hardcoded to zero.

XLR8 is programmed only via the USB-UART path. While the XLR8 board includes the ICSP header it is useful only as a convenient connector for the SPI interface; it can’t be used to load programs/sketches into the design.

3.7 I2C / Two Wire Interface (TWI)

Although the Arduino Uno’s Rev3 layout has separate SDA and SCL pins for I2C/TWI, they are still connected on the Uno board to the original A4/A5 pins. This means that when using the I2C interface, two of your analog inputs are no longer available. On XLR8, I2C/TWI is by default only available on the SDA/SCL pins and the A4/A5 pins can still be used as analog or digital inputs (but not outputs). If you have existing projects that need to use A4/A5 as the I2C pins, you can reconnect them to SDA/SCL by populating 0ohm resistors at R60 and R61.

Another difference between the I2C/TWI on XLR8 and Arduino that you will likely never notice is related to the TWBR register. While Arduino allows running with non-standard SCL clock frequencies, XLR8 is optimized for 100KHz and 400KHz operation and setting TWBR and the prescaler bits of TWSR for other speeds will not be operational.

3.8 UART

XLR8 has a Universal Asynchronous serial Receiver and Transmitter (UART) block similar to the ATmega328p USART. However, XLR8’s UART does not implement the Synchronous UART (USART) in SPI Modes that are available on the ATmega328p. Therefore, the UMSEL01, UMSEL00, and UCPOL0 bits of the UCSR0C (0x) register, while implemented, have no effect and the UART block always runs in “Asynchronous UART” mode. We have not yet seen a case where anything else is needed in an Arduino environment; if you have one, please let us know.

3.9 SPI

The SPI interface on XLR8 should operate the same as the SPI interface on an Uno or Redboard. The only item to note is that we’ve seen some SPI examples where the XLR8/Uno/Redboard is the SPI slave and instead of being driven from the SPI master, the SS pin is left floating. Although a sloppy design practice, an Uno or Redboard will often still work in this arrangement. Our XLR8 board, due to its I/O pullups, needs to have SS driven and not floating.

3.9.1 SPI Best Practices

In rare cases, we have observed an issue with the SPI clock being properly sampled by a shield. We suggest adding a 100-200 ohm pulldown resistor to the SPI clock pin D13 for more robust operation.
3.10 Analog Compare

XLR8 does not have the Analog Compare function that is found in the ATmega328p. The ACME bit and analog compare triggering (ADTS=001) of the ADCSRB (0x7B) register, the ACSR (0x30) register, and the DIDR1 (0x7F) register are not implemented. If an analog compare function is desired, pins 6 and 7 on the (unpopulated) JTAG connector are wired to MAX10 pins DIFFIO_TX_RX_B1P/N which is a differential I/O. Using the OpenXLR8 platform, a user could implement an analog compare function that is very similar to the ATMega328’s, although the pin voltage would need to be limited to 3.3V.
3.11 EEPROM

XLR8 does not have the EEPROM memory that is found in the ATmega328p. However, there is an SOIC-8 location that could be populated with an EEPROM and using the OpenXLR8 platform, a user could implement an EEPROM function that is very similar to the one found in the ATMega328P. The SOIC-8 pinout is:

<table>
<thead>
<tr>
<th>FPGA pin</th>
<th>Typical Use</th>
<th>SOIC-8 pin</th>
<th>SOIC-8 pin</th>
<th>Typical Use</th>
<th>FPGA pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>C12</td>
<td>Addr0</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>Vcc</td>
</tr>
<tr>
<td>B13</td>
<td>Addr1</td>
<td>2</td>
<td>7</td>
<td>WP</td>
<td>D11</td>
</tr>
<tr>
<td>C11</td>
<td>Addr2</td>
<td>3</td>
<td>6</td>
<td>SCL</td>
<td>D12</td>
</tr>
<tr>
<td>Gnd</td>
<td>Vss</td>
<td>4</td>
<td>5</td>
<td>SDA</td>
<td>E13</td>
</tr>
</tbody>
</table>

3.12 Power

The Arduino Uno and the Sparkfun RedBoard are only able to supply 50mA from the 3.3V pin header. XLR8 has a 1.5A regulator creating its 3.3V power (an LT1963EST-3.3) and the Max10 FPGA is estimated to draw less than 100mA, leaving a significant amount of 3.3V power available to the user for shields or other circuits. (Keep in mind that USB power is limited to about 500mA.)

3.13 Pin13 LED

Digital pin 13 is used for both the on-board LED as well as the SPI clock, SCK. While the Arduino Uno R3 adds circuitry to prevent the LED and its pulldown resistor from affecting SCK, the Sparkfun Redboard does not take those precautions. On XLR8 we’ve decided to follow Arduino’s lead and keep SCK separated from the LED.

3.14 JTAG Interface

The JTAG pins on the board are used for MFG loading of the original XLR8 image, as well as MFG testing. All available XLR8 images and OpenXLR8 images must be loaded through the IDE.

Using the JTAG pins to load the MAX10 FPGA with any other image will wipe out the XLR8 image and not allow a recovery back to the factory image. If the user would like to load their own FPGA images, they will need 100-ohm pull-down resistor on the TCK pin to do the load.

4 ATmega328p features not implemented

The ATmega328p microcontroller includes several features that are seldom or never used in an Arduino environment. In XLR8, these features are hard coded to match the Arduino usage. Therefore, when comparing to the ATmega328p specification the following differences may be noted, however they do not impact functionality in an Arduino environment. One exception is that Arduino Uno allows programming via the SPI interface while, as mentioned in section 3.6, XLR8 has not implemented this feature (SPIEN fuse).
4.1 Fuses

The Arduino Uno runs with the following fuse settings.

- **Low Fuse 0xFF (all unprogrammed)**
  - CKDIV8: Default to clock division factor of 1.
  - CKOUT: Don’t output system clock on port B pin 0
  - SUT1/0: Start up time
  - CKSEL3/2/1/0
    - Always use external clock (supplied by 16MHz) crystal.
    - XLR8 does not have the internal 128kHz RC Oscillator or the Calibrated Internal RC Oscillator that exists in ATmega328p. The OSCCAL (0x66) register is not implemented.
    - Because ATmega328 shares the TOSC1/2 pins with the XTAL1/2 pins, and XTAL1/2 are used to bring in the external 16MHz clock, the asynchronous mode of timer/counter 2 is not available and the ASSR register (0xB6) is not implemented.

- **High Fuse 0xDE in Arduino (SPIEN and BOOTRST programmed), 0xFE in XLR8 (BOOTRST programmed)**
  - RSTDISBL: External reset is always enabled. This is how the microcontroller returns to the bootloader to upload a new sketch from the Arduino IDE to program memory
  - DWEN: The debug wire feature is not enabled in Arduino and is not implemented in XLR8.
  - SPIEN: While Arduino programs this bit, XLR8 instead functions as if it were not programmed (see section 3.6)
  - WDTON: Watchdog always on not enabled
  - EESAVE: EEPROM not preserved through chip erase. Not applicable on XLR8.
  - BOOTSZ1/0: Bootloader is 256 words starting at 0x3F00
  - BOOTRST: Reset starts at the bootloader address

- **Extended Fuse 0x05 in Arduino (BODLEVEL1 programmed), 0x07 in XLR8 (all unprogrammed)**
  - BODLEVEL2/1/0: In XLR8 the brown out detection circuitry is disabled and not implemented. The BODS and BOSE bits of the MCUCR (0x35) register are not implemented, as well as the BORF bit of the MCUSR (0x34) register.

4.2 Power Reduction

Because we don’t see it being used very often (or at all) in Arduino projects, XLR8 has not implemented the ATmega328p’s Power Reduction Register PRR (0x64), Sleep Mode Control Register SMCR (0x33), or Clock Prescale Register CLKPR (0x61). They could be added if customer feedback indicates that would be a good thing to do.

5 Xcelerator Blocks (XBs)

Xcelerator Blocks are custom hardware blocks implemented within the XLR8 FPGA chip and are tightly integrated with the ATmega328 clone that is also implemented inside the FPGA chip. These custom hardware blocks can implement almost any functionality you can dream up, and can then
be loaded into the XLR8 as with the Arduino toolset. Since an FPGA can be reprogrammed many times, a single XLR8 can be reconfigured to incorporate different XB depending on the project requirements.

XLR8 ships with three sample XBs: Floating Point, NeoPixel, and Servo Control. As mentioned in section 3.2, the software libraries are delivered as .zip files from our github site ([https://github.com/AloriumTechnology](https://github.com/AloriumTechnology)). They are installed like other Arduino .zip libraries as described here ([https://www.arduino.cc/en/Guide/Libraries - toc4](https://www.arduino.cc/en/Guide/Libraries - toc4)).

Note: There is one thing to be aware of with all Arduino libraries. If you download an updated version and simply rename the directory of the old version (perhaps you think you might want to go back to it or maybe you are just feeling nostalgic), Arduino very likely will still end up using the old version as it simply looks for the first place it can find the #include header file name, regardless of what directory it is in. If you want to save an old version of a library, move it to someplace outside of your Arduino/libraries folder.

5.1 Floating Point
As an 8 bit microcontroller, the ATmega328p struggles with floating point math. The Floating Point XB provides functions that will give you floating point results in about ¼ the time that it takes software floating point to get the same answer. Available functions include add, subtract, multiply, and divide.

5.2 Servo Control
It is common for the standard Servo.h library to cause jitter in the servo control due to timing uncertainties caused by interrupt processing. The Servo Control XB completely eliminates this jitter by putting a dedicated hardware timer behind all 20 digital and analog pins. The XLR8Servo.h library is a drop-in replacement for the standard Servo.h library, so taking advantage of this XB is as simple as changing one line in your sketch from

```c
#include <Servo.h>
```

to

```c
#include <XLR8Servo.h>
```

5.3 NeoPixel Control
The standard Arduino struggles to meet the timing requirements of NeoPixels. With the XLR8 NeoPixel hardware and library, interrupts remain enabled, data memory is saved, and pixel color information does not need to be rewritten when brightness is lowered and then brought back up. The XLR8NeoPixel library can be a drop-in replacement for the standard Adafruit_NeoPixel library. It is common for other libraries, such as Adafruit_NeoMatrix, to build on the Adafruit_NeoPixel library and with just three lines added to the top of your sketch, it is possible to get the XLR8 advantages used by those libraries as well. For example,

```c
// 3 new lines added
#include <XLR8NeoPixel.h>
#define Adafruit_NeoPixel XLR8NeoPixel
#define ADAFRUIT_NEOPIXEL_H
// Existing lines kept
#include <Adafruit_GFX.h>
```
#include <Adafruit_NeoMatrix.h>
#include <Adafruit_NeoPixel.h>

As explanation, working from the bottom up:

- Including Adafruit_NeoPixel.h remains because other library files that we are using but are not modifying (Adafruit_NeoMatrix.h in this case) have #include <Adafruit_NeoPixel.h>. The way Arduino handles compiling and linking generally requires that lower level #includes also be included in the sketch.
- Including Adafruit_NeoMatrix remains because that is the library that this particular sketch is using.
- Including Adafruit_GFX remains for the same reason that it was there initially. It is included by Adafruit_NeoMatrix, and the way Arduino handles compiling and linking generally requires that lower level #includes also be included in the sketch.
- Even though we are still including the Adafruit_NeoPixel.h, adding #define ADAFRUIT_NEOPIXEL_H causes that header file to appear empty instead of having the actual Adafruit library code.
- By adding #define Adafruit NeoPixel XLR8NeoPixel, any place in your sketch, or in the libraries that your sketch is using, where an Adafruit_NeoPixel string is instantiated, it will instantiate an XLR8NeoPixel string instead.
- And of course, the XLR8NeoPixel library is included.

And designed to become even more powerful in the future. Watch for updates.

6 Register Summary

The registers used in XLR8 are listed below. Those with a grey background have identical function to the equivalent ATmega328p register. Those with just a touch of grey are reserved in both XLR8 and ATmega328p but are noted because they are used in the ATmega328pb. Those with green background function like the equivalent ATmega328p register, but with some differences as noted. Those with blue background are new registers added for XLR8 functions. Those with white background are ATmega328p registers that do not exist in XLR8.

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0xFF)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>(0xFE)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>(0xFD)</td>
<td>SVPWH</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Servo Pulse Width High Register 6.1.7</td>
</tr>
<tr>
<td>(0xFC)</td>
<td>SVPWL</td>
<td>Servo Pulse Width Low Register 6.1.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xFB)</td>
<td>SVCR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NeoPixel Data 2 register, function depends on NEOCMD 6.1.8</td>
</tr>
<tr>
<td>(0xFA)</td>
<td>NEOD2</td>
<td>NeoPixel Data 2 register, function depends on NEOCMD 6.1.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xF9)</td>
<td>NEOCH</td>
<td>NeoPixel Data 1 register, function depends on NEOCMD, often high half of pixel address/length 6.1.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xF8)</td>
<td>NEOD0</td>
<td>NeoPixel Data 0 register, function depends on NEOCMD, often low half of pixel address/length 6.1.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xF7)</td>
<td>NEOCR</td>
<td>NEOPIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xF6)</td>
<td>PID_OP_L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Low Byte output 6.1.11</td>
</tr>
<tr>
<td>(0xF5)</td>
<td>PID_OP_H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>High Byte output 6.1.11</td>
</tr>
<tr>
<td>(0xF4)</td>
<td>PID_PV_L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Process variable low byte 6.1.11</td>
</tr>
</tbody>
</table>

Copyright 2018 Alorium Technology
<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x3)</td>
<td>PID_PV_H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.11</td>
</tr>
<tr>
<td>(0x2)</td>
<td>PID_SP_L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.11</td>
</tr>
<tr>
<td>(0x1)</td>
<td>PID_SP_H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.11</td>
</tr>
<tr>
<td>(0xF0)</td>
<td>PID_KP_L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.11</td>
</tr>
<tr>
<td>(0xEF)</td>
<td>PID_KP_H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.11</td>
</tr>
<tr>
<td>(0xEE)</td>
<td>PID_KI_L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.11</td>
</tr>
<tr>
<td>(0xED)</td>
<td>PID_KI_H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.11</td>
</tr>
<tr>
<td>(0xEC)</td>
<td>PID_KD_L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.11</td>
</tr>
<tr>
<td>(0xEB)</td>
<td>PID_KD_H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.11</td>
</tr>
<tr>
<td>(0xEA)</td>
<td>PIDCR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.11</td>
</tr>
<tr>
<td>(0xE9)</td>
<td>QERAT3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.12</td>
</tr>
<tr>
<td>(0xE8)</td>
<td>QERAT2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.12</td>
</tr>
<tr>
<td>(0xE7)</td>
<td>QERAT1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.12</td>
</tr>
<tr>
<td>(0xE6)</td>
<td>QERAT0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.12</td>
</tr>
<tr>
<td>(0xE5)</td>
<td>QECNT3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.12</td>
</tr>
<tr>
<td>(0xE4)</td>
<td>QECNT2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.12</td>
</tr>
<tr>
<td>(0xE3)</td>
<td>QECNT1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.12</td>
</tr>
<tr>
<td>(0xE2)</td>
<td>QECNT0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.12</td>
</tr>
<tr>
<td>(0xE1)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xE0)</td>
<td>QECR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.12</td>
</tr>
<tr>
<td>(0xDF)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xDE)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xDD)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xDC)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xDB)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xDA)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xD9)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xD8)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xD7)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xD6)</td>
<td>XLR8VERT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.1</td>
</tr>
<tr>
<td>(0xD5)</td>
<td>XLR8VERH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.1</td>
</tr>
<tr>
<td>(0xD4)</td>
<td>XLR8VERL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.1</td>
</tr>
<tr>
<td>(0xD3)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xD2)</td>
<td>FCFGDAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.6</td>
</tr>
<tr>
<td>(0xD1)</td>
<td>FCFGSTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.6</td>
</tr>
<tr>
<td>(0xD0)</td>
<td>FCFGCTRL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.6</td>
</tr>
<tr>
<td>(0xCF)</td>
<td>FCFGID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.2</td>
</tr>
<tr>
<td>(0xCE)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xCD)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xCC)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xCB)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xCA)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xC9)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xC8)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xC7)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xC6)</td>
<td>UDR0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1.2</td>
</tr>
<tr>
<td>(0xC5)</td>
<td>UBRROH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xC4)</td>
<td>UBRROL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xC3)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xC2)</td>
<td>UCSROC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.8</td>
</tr>
<tr>
<td>(0xC1)</td>
<td>UCSR0B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xC0)</td>
<td>UCSR0A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xBF)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright 2018 Alorium Technology
<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0xEB)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>(0xDB)</td>
<td>TWAMR</td>
<td>TWAM6</td>
<td>TWAM5</td>
<td>TWAM4</td>
<td>TWAM3</td>
<td>TWAM2</td>
<td>TWAM1</td>
<td>TWAM0</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>(0xCB)</td>
<td>TWCRTWINT</td>
<td>TWEA</td>
<td>TWSTA</td>
<td>TWSTO</td>
<td>TWWC</td>
<td>TWEN</td>
<td>–</td>
<td>TWIE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xBB)</td>
<td>TWDR</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>2-wire Serial Interface Data Register</td>
</tr>
<tr>
<td>(0xBA)</td>
<td>TWAR</td>
<td>TWA6</td>
<td>TWA5</td>
<td>TWA4</td>
<td>TWA3</td>
<td>TWA2</td>
<td>TWA1</td>
<td>TWA0</td>
<td>TWGCE</td>
<td></td>
</tr>
<tr>
<td>(0xBA)</td>
<td>TW5R</td>
<td>TW57</td>
<td>TW56</td>
<td>TW55</td>
<td>TW54</td>
<td>TW53</td>
<td>–</td>
<td>TWPS1</td>
<td>TWPS0</td>
<td></td>
</tr>
<tr>
<td>(0xBB)</td>
<td>TWBR</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>2-wire Serial Interface Bit Rate Register</td>
</tr>
<tr>
<td>(0xB7)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>(0xB6)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>ASSR 4.1</td>
</tr>
<tr>
<td>(0xB5)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>(0xB4)</td>
<td>OCR2B</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Timer/Counter2 Output Compare Register B</td>
</tr>
<tr>
<td>(0xB3)</td>
<td>OCR2A</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Timer/Counter2 Output Compare Register A</td>
</tr>
<tr>
<td>(0xB2)</td>
<td>TCNT2</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Timer/Counter2 (8-bit)</td>
</tr>
<tr>
<td>(0xB1)</td>
<td>TCCR2B</td>
<td>FOC2A</td>
<td>FOC2B</td>
<td>–</td>
<td>–</td>
<td>WGM22</td>
<td>CS22</td>
<td>CS21</td>
<td>CS20</td>
<td></td>
</tr>
</tbody>
</table>
| (0xBA)  | TCCR2A                | COM2A1| COM2A0| COM2B1| COM2B0| –     | –     | –     | –     | TCC
| (0xAF)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     |                             |
| (0xAE)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | SPDR1                       |
| (0xAD)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | SPSR1                       |
| (0xAC)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | SPCR1                       |
| (0xAB)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | OCR4BH                      |
| (0xAA)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | OCR4BL                      |
| (0xA9)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | OCR4AH                      |
| (0xA8)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | OCR4AL                      |
| (0xA7)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | ICR4H                       |
| (0xA6)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | ICR4L                       |
| (0xA5)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | TCNT4H                      |
| (0xA4)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | TCNT4L                      |
| (0xA3)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     |                             |
| (0xA2)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | TCCR4C                      |
| (0xA1)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | TCCR4B                      |
| (0xA0)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | TCCR4A                      |
| (0x9F)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     |                             |
| (0x9E)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     |                             |
| (0x9D)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     |                             |
| (0x9C)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     |                             |
| (0x9B)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | OCR3BH                      |
| (0x9A)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | OCR3BL                      |
| (0x99)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | OCR3AH                      |
| (0x98)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | OCR3AL                      |
| (0x97)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | ICR3H                       |
| (0x96)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | ICR3L                       |
| (0x95)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | TCNT3H                      |
| (0x94)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | TCNT3L                      |
| (0x93)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     |                             |
| (0x92)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | TCCR3C                      |
| (0x91)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | TCCR3B                      |
| (0x90)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     | TCCR3A                      |
| (0x8F)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     |                             |
| (0x8E)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     |                             |
| (0x8D)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     |                             |
| (0x8C)  | Reserved              | –     | –     | –     | –     | –     | –     | –     | –     |                             |
| (0x8B)  | OCR1BH                | –     | –     | –     | –     | –     | –     | –     | –     | Timer/Counter1 – Output Compare Register B High Byte |
| (0x8A)  | OCR1BL                | –     | –     | –     | –     | –     | –     | –     | –     | Timer/Counter1 – Output Compare Register B Low Byte |

**Address:** 8-bit wired address. **Name:** Register name. **Bit 7-0:** Bit positions within the register. **Notes:** Additional information about the register.
<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x89)</td>
<td>OCR1AH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Timer/Counter1 – Output Compare Register A High Byte</td>
</tr>
<tr>
<td>(0x88)</td>
<td>OCR1AL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Timer/Counter1 – Output Compare Register A Low Byte</td>
</tr>
<tr>
<td>(0x87)</td>
<td>ICR1H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Timer/Counter1 – Input Capture Register High Byte</td>
</tr>
<tr>
<td>(0x86)</td>
<td>ICR1L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Timer/Counter1 – Input Capture Register Low Byte</td>
</tr>
<tr>
<td>(0x85)</td>
<td>TCNT1H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Timer/Counter1 – Counter Register High Byte</td>
</tr>
<tr>
<td>(0x84)</td>
<td>TCNT1L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Timer/Counter1 – Counter Register Low Byte</td>
</tr>
<tr>
<td>(0x83)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>(0x82)</td>
<td>TCCR1C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>(0x81)</td>
<td>TCCR1B</td>
<td>ICIN1</td>
<td>ICES1</td>
<td>WGM13</td>
<td>WGM12</td>
<td>CS12</td>
<td>CS11</td>
<td>CS10</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>(0x80)</td>
<td>TCCR1A</td>
<td>COM1A1</td>
<td>COM1A0</td>
<td>COM1B1</td>
<td>COM1B0</td>
<td>–</td>
<td>–</td>
<td>WGM11</td>
<td>WGM10</td>
<td></td>
</tr>
<tr>
<td>(0x7F)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>(0x7E)</td>
<td>DIDR0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>(0x7D)</td>
<td>XLR8ADCR</td>
<td>AD12EN</td>
<td>–</td>
<td>–</td>
<td>ADC5D</td>
<td>ADC4D</td>
<td>ADC3D</td>
<td>ADC2D</td>
<td>ADC1D</td>
<td>ADC0D</td>
</tr>
<tr>
<td>(0x7C)</td>
<td>ADMUX</td>
<td>REFS1</td>
<td>REFS0</td>
<td>ADBLR</td>
<td>–</td>
<td>MUX3</td>
<td>MUX2</td>
<td>MUX1</td>
<td>MUX0</td>
<td>3.5</td>
</tr>
<tr>
<td>(0x7B)</td>
<td>ADCSRB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>6.1.10</td>
</tr>
<tr>
<td>(0x7A)</td>
<td>ADCSRA</td>
<td>ADEN</td>
<td>ADSC</td>
<td>ADATE</td>
<td>ADIF</td>
<td>ADIE</td>
<td>ADPS2</td>
<td>ADPS1</td>
<td>ADPS0</td>
<td>ACME 3.9</td>
</tr>
<tr>
<td>(0x79)</td>
<td>ADCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ADC Data Register High byte</td>
</tr>
<tr>
<td>(0x78)</td>
<td>ADCL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ADC Data Register Low byte</td>
</tr>
<tr>
<td>(0x77)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>(0x76)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>(0x75)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>(0x74)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>(0x73)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>(0x72)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>(0x71)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>(0x70)</td>
<td>TIMSK2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>(0x6F)</td>
<td>TIMSK1</td>
<td>–</td>
<td>–</td>
<td>ICIE1</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Reserved</td>
</tr>
<tr>
<td>(0x6E)</td>
<td>TIMSK0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Reserved</td>
</tr>
<tr>
<td>(0x6D)</td>
<td>PCMSK2</td>
<td>PCINT23</td>
<td>PCINT22</td>
<td>PCINT21</td>
<td>PCINT20</td>
<td>PCINT19</td>
<td>PCINT18</td>
<td>PCINT17</td>
<td>PCINT16</td>
<td>Reserved</td>
</tr>
<tr>
<td>(0x6C)</td>
<td>PCMSK1</td>
<td>PCINT14</td>
<td>PCINT13</td>
<td>PCINT12</td>
<td>PCINT11</td>
<td>PCINT10</td>
<td>PCINT9</td>
<td>PCINT8</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>(0x6B)</td>
<td>PCMSK0</td>
<td>PCINT7</td>
<td>PCINT6</td>
<td>PCINT5</td>
<td>PCINT4</td>
<td>PCINT3</td>
<td>PCINT2</td>
<td>PCINT1</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>(0x6A)</td>
<td>OX8MSK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OX8MSK Pin Control Interrupt Mask</td>
</tr>
<tr>
<td>(0x69)</td>
<td>EICRA</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>ISC11</td>
<td>OX8WP1</td>
</tr>
<tr>
<td>(0x68)</td>
<td>PCICR</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>ISC0</td>
<td>OX8WP0</td>
</tr>
<tr>
<td>(0x67)</td>
<td>OX8IFR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OX8IFR Pin Control Interrupt Flag Register</td>
</tr>
<tr>
<td>(0x66)</td>
<td>OX8ICR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OX8ICR Pin Control Interrupt Register</td>
</tr>
<tr>
<td>(0x65)</td>
<td>XACK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XLR8 IRQ Acknowledge Register</td>
</tr>
<tr>
<td>(0x64)</td>
<td>PRR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>XLR8 IRQ Acknowledge Register</td>
</tr>
<tr>
<td>(0x63)</td>
<td>XMSC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XLR8 IRQ Mask Register</td>
</tr>
<tr>
<td>(0x62)</td>
<td>XIFR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XLR8 IRQ Flag Register</td>
</tr>
<tr>
<td>(0x61)</td>
<td>XICR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XLR8 IRQ Control Register</td>
</tr>
<tr>
<td>(0x60)</td>
<td>WDTCSR</td>
<td>WEIF</td>
<td>WDEIE</td>
<td>WDEIE</td>
<td>WDP3</td>
<td>WDECE</td>
<td>WDE</td>
<td>WDP2</td>
<td>WDP1</td>
<td>WDP0</td>
</tr>
<tr>
<td>0x3F(0x5F)</td>
<td>SREG</td>
<td>I</td>
<td>T</td>
<td>H</td>
<td>S</td>
<td>V</td>
<td>N</td>
<td>Z</td>
<td>C</td>
<td>6.1.9</td>
</tr>
<tr>
<td>0x3E(0x5E)</td>
<td>SPH</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>SP11</td>
<td>SP10</td>
<td>SP9</td>
<td>SP8</td>
<td>6.1.9</td>
</tr>
<tr>
<td>0x3D(0x5D)</td>
<td>SPL</td>
<td>SP7</td>
<td>SP6</td>
<td>SP5</td>
<td>SP4</td>
<td>SP3</td>
<td>SP2</td>
<td>SP1</td>
<td>SP0</td>
<td>6.1.9</td>
</tr>
<tr>
<td>0x3C(0x5C)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>6.1.9</td>
</tr>
<tr>
<td>0x3B(0x5B)</td>
<td>XFR3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XLR8 Function (floating point) 32 bit Result High Byte</td>
</tr>
<tr>
<td>0x3A(0x5A)</td>
<td>XFR2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XLR8 Function (floating point) 32 bit Result Byte</td>
</tr>
<tr>
<td>0x39(0x59)</td>
<td>XFR1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XLR8 Function (floating point) 32 bit Result Byte</td>
</tr>
<tr>
<td>0x38(0x58)</td>
<td>XFR0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XLR8 Function (floating point) 32 bit Result Low Byte</td>
</tr>
<tr>
<td>0x37(0x57)</td>
<td>SPMCSR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SPMCSR</td>
</tr>
<tr>
<td>0x36(0x56)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Reserved</td>
</tr>
<tr>
<td>Address</td>
<td>Name</td>
<td>Bit 7</td>
<td>Bit 6</td>
<td>Bit 5</td>
<td>Bit 4</td>
<td>Bit 3</td>
<td>Bit 2</td>
<td>Bit 1</td>
<td>Bit 0</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>0x35(0x55)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>MCUCR 4.1.3.4.3.6</td>
</tr>
<tr>
<td>0x34(0x54)</td>
<td>MCUSR</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>EXTRF PORF BORF 4.1</td>
</tr>
<tr>
<td>0x33(0x53)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>SMCR 4.2</td>
</tr>
<tr>
<td>0x32(0x52)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>DWDR</td>
</tr>
<tr>
<td>0x31(0x51)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>ACSR 3.9</td>
</tr>
<tr>
<td>0x30(0x50)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>ACSR B</td>
</tr>
<tr>
<td>0x2F(0x4F)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>ACSR B</td>
</tr>
<tr>
<td>0x2E(0x4E)</td>
<td>SPRD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SPI Data Register</td>
<td></td>
</tr>
<tr>
<td>0x2D(0x4D)</td>
<td>SPRR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>–</td>
<td>SPI2X</td>
</tr>
<tr>
<td>0x2C(0x4C)</td>
<td>SPCR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SPIE</td>
<td>SPE DORD MSTR CPOL CPHA SPR1 SPR0</td>
</tr>
<tr>
<td>0x2B(0x4B)</td>
<td>GPIOR2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>–</td>
<td>General Purpose I/O Register 2</td>
</tr>
<tr>
<td>0x2A(0x4A)</td>
<td>GPIOR1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>–</td>
<td>General Purpose I/O Register 1</td>
</tr>
<tr>
<td>0x29(0x49)</td>
<td>CLKSPD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>–</td>
<td>Clock speed programming used by XLR8 bootloader 6.1.3</td>
</tr>
<tr>
<td>0x28(0x48)</td>
<td>OCR0B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OCRB</td>
<td>Timer/Counter0 Output Compare Register B</td>
</tr>
<tr>
<td>0x27(0x47)</td>
<td>OCRA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>–</td>
<td>Timer/Counter0 Output Compare Register A</td>
</tr>
<tr>
<td>0x26(0x46)</td>
<td>TCF0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>–</td>
<td>Timer/Counter0 (8-bit)</td>
</tr>
<tr>
<td>0x25(0x45)</td>
<td>TCR0B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>–</td>
<td>WGM02 CS02 CS01 CS00</td>
</tr>
<tr>
<td>0x24(0x44)</td>
<td>TCR0A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>–</td>
<td>WGM01 WGM0</td>
</tr>
<tr>
<td>0x23(0x43)</td>
<td>GTCCR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>–</td>
<td>PSRASY PSRSYNC</td>
</tr>
<tr>
<td>0x22(0x42)</td>
<td>EEARH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>–</td>
<td>EEPROM Address Register High Byte</td>
</tr>
<tr>
<td>0x21(0x41)</td>
<td>EEARL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>–</td>
<td>EEPROM Address Register Low Byte</td>
</tr>
<tr>
<td>0x20(0x40)</td>
<td>EEDR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>–</td>
<td>EEPROM Data Register</td>
</tr>
<tr>
<td>0x1F(0x3F)</td>
<td>EECR</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>EEPM1 EEPM0 EERIE EEMPE EEPE EERE</td>
</tr>
<tr>
<td>0x1E(0x3E)</td>
<td>GPIOR0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>–</td>
<td>General Purpose I/O Register 0</td>
</tr>
<tr>
<td>0x1D(0x3D)</td>
<td>EIMSK</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>INT1 INT0</td>
</tr>
<tr>
<td>0x1C(0x3C)</td>
<td>EIFR</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>INTF1 INTF0</td>
</tr>
<tr>
<td>0x1B(0x3B)</td>
<td>PCIFR</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>PCI10 PCI11</td>
</tr>
<tr>
<td>0x1A(0x3A)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>TIFR4</td>
</tr>
<tr>
<td>0x19(0x39)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>TIFR3</td>
</tr>
<tr>
<td>0x18(0x38)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>TIFR4</td>
</tr>
<tr>
<td>0x17(0x37)</td>
<td>TIFR2</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>OCF2B OCF2A TOV2</td>
</tr>
<tr>
<td>0x16(0x36)</td>
<td>TIFR1</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>OCF1B OCF1A TOV1</td>
</tr>
<tr>
<td>0x15(0x35)</td>
<td>TIFR0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>OCF0B OCF0A TOV0</td>
</tr>
<tr>
<td>0x14(0x34)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>0x13(0x33)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>0x12(0x32)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>0x11(0x31)</td>
<td>XFSTAT</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>6.1.9</td>
</tr>
<tr>
<td>0x10(0x30)</td>
<td>XFDONE</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>XFCMD</td>
</tr>
<tr>
<td>0x0F(0x2F)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>0x0E(0x2E)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>328PB PORTE</td>
</tr>
<tr>
<td>0x0D(0x2D)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>328PB DDRE</td>
</tr>
<tr>
<td>0x0C(0x2C)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>328PB PINE</td>
</tr>
<tr>
<td>0x0B(0x2B)</td>
<td>PORTD</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>pullup 3.4</td>
</tr>
<tr>
<td>0x0A(0x2A)</td>
<td>DDRD</td>
<td>DDD7</td>
<td>DDD6</td>
<td>DDD5</td>
<td>DDD4</td>
<td>DDD3</td>
<td>DDD2</td>
<td>DDD1</td>
<td>DDD0</td>
<td>pullup 3.4</td>
</tr>
<tr>
<td>0x09(0x29)</td>
<td>PIND</td>
<td>PIN7</td>
<td>PIN6</td>
<td>PIN5</td>
<td>PIN4</td>
<td>PIN3</td>
<td>PIN2</td>
<td>PIN1</td>
<td>PIN0</td>
<td>reset 3.4</td>
</tr>
<tr>
<td>0x08(0x28)</td>
<td>PORTC</td>
<td>–</td>
<td>DDC6</td>
<td>DDC5</td>
<td>DDC4</td>
<td>DDC3</td>
<td>DDC2</td>
<td>DDC1</td>
<td>DDC0</td>
<td>pullup 3.4</td>
</tr>
<tr>
<td>0x07(0x27)</td>
<td>DDRC</td>
<td>–</td>
<td>DDC6</td>
<td>DDC5</td>
<td>DDC4</td>
<td>DDC3</td>
<td>DDC2</td>
<td>DDC1</td>
<td>DDC0</td>
<td>pullup 3.4</td>
</tr>
<tr>
<td>0x06(0x26)</td>
<td>PINC</td>
<td>PIN6</td>
<td>PIN5</td>
<td>PIN4</td>
<td>PIN3</td>
<td>PIN2</td>
<td>PIN1</td>
<td>PIN0</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>0x05(0x25)</td>
<td>PORTB</td>
<td>–</td>
<td>DDC6</td>
<td>DDC5</td>
<td>DDC4</td>
<td>DDC3</td>
<td>DDC2</td>
<td>DDC1</td>
<td>DDC0</td>
<td>pullup 3.4</td>
</tr>
<tr>
<td>0x04(0x24)</td>
<td>DDRB</td>
<td>DDB7</td>
<td>DDB6</td>
<td>DDB5</td>
<td>DDB4</td>
<td>DDB3</td>
<td>DDB2</td>
<td>DDB1</td>
<td>DDB0</td>
<td>pullup 3.4</td>
</tr>
<tr>
<td>0x03(0x23)</td>
<td>PINB</td>
<td>PIN7</td>
<td>PIN6</td>
<td>PIN5</td>
<td>PIN4</td>
<td>PIN3</td>
<td>PIN2</td>
<td>PIN1</td>
<td>PIN0</td>
<td>reset 3.4</td>
</tr>
<tr>
<td>0x02(0x22)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

Copyright 2018 Alorium Technology  Page 17
### 6.1 XLR8 and XB Register Descriptions

#### 6.1.1 XLR8VERL, XLR8VERH, XLR8VERT – Version Number Registers

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01(0x21)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>= unchanged from ATmega328p</td>
</tr>
<tr>
<td>0x00(0x20)</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>= ATmega328p registers not implemented in XLR8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>= Some differences in XLR8 compared to ATmega328p</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>= new registers for XLR8 Blocks</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>= Reserved registers that are best not used for XLR8 blocks because ATmega328PB uses them</td>
</tr>
</tbody>
</table>

The version number register provides the FPGA design revision, while the version flags register indicates if the build had a mixed or modified version. The registers have a constant value for a particular design, but the value changes for each version. The easiest way to use these registers is with the XLR8Info library [https://github.com/AloriumTechnology/XLR8Info](https://github.com/AloriumTechnology/XLR8Info).

#### 6.1.2 FCFGCID – Chip ID Register

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xC6(0x46)</td>
<td>FCFGCID</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>write-reset</td>
</tr>
</tbody>
</table>

The chip ID register is a read-only register that provides chip ID information. Multiple bytes of chip ID information are available and each read presents the next byte. Writing the register (with any value) resets the read pointer back to the beginning (and does not store the write data in any way).

#### 6.1.3 CLKSPD – Clock Speed Register

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x29(0x49)</td>
<td>CLKSPD</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>Clock speed programming used by XLR8 bootloader</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>0x64</td>
<td>PRR</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The clock speed register holds a constant value that represents the value to be programmed into the UBRR0L register to run the UART at a baud rate of 115200. It is used by the modified
bootloader mentioned in section 3.6 to allow it to run correctly regardless of whether XLR8 is running 16MHZ, 32MHz, or some other speed.

XLR8 includes an on-chip oscillator that currently isn’t being used, but a divide-by-1024 version of it can be output to digital pin 8 by writing bit 0 of the CLKSPD register high. This is a write-only operation, it does not change the value that is read from the CLKSPD register. The internal oscillator can be turned off entirely by setting the PRINTOSC bit of the PRR register. As described in section 3.12, the other bits of this register are currently unused.

6.1.4 XICR, XIFR, XMSK - XLR8 IRQ Registers

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x61)</td>
<td>XICR</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>OX8</td>
</tr>
<tr>
<td>Read/Write</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>BI</td>
</tr>
<tr>
<td>Initial Value</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>0, 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x62)</td>
<td>XIFR</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>OX8</td>
</tr>
<tr>
<td>Read/Write</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>BI</td>
</tr>
<tr>
<td>Initial Value</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>0, 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x63)</td>
<td>XMSK</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>OX8</td>
</tr>
<tr>
<td>Read/Write</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>BI</td>
</tr>
<tr>
<td>Initial Value</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>0, 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x65)</td>
<td>XACK</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>OX8</td>
</tr>
<tr>
<td>Read/Write</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>BI</td>
</tr>
<tr>
<td>Initial Value</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>0, 0</td>
</tr>
</tbody>
</table>

For a description of the XLR8 IRQ Registers, see the Pin Change Interrupt Register Definition in the Atmel 8-bit AVR Microcontroller Datasheet.

There are some differences of note. The XACK register prevents the XIFR bit from getting set after the ISR has been invoked. Before exiting the ISR, the XACK bit should be cleared.

Bit 0 is for Built-In interrupt sources, such as extra GPIO interface pin change interrupts on the Sno and Hinx boards.

Bit 1 is for interrupts defined by the user in the OpenXLR8 block.

6.1.5 OX8ICR, OX8IFR, OX8MSK - OpenXLR8 Pin Change Interrupts

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x66)</td>
<td>OX8ICR</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>--</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>--</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x67)</td>
<td>OX8IFR</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>--</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>--</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x6A)</td>
<td>OX8MSK</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>--</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>--</td>
</tr>
</tbody>
</table>

For a description of the OpenXLR8 Pin Change Interrupt registers, see the Pin Change Interrupt Register Definition in the Atmel 8-bit AVR Microcontroller Datasheet.

The interrupt sources for the OpenXLR8 interrupts are defined by the user when designing the OpenXLR8 block.
### 6.1.6  FCFGDAT, FCFGSTS, FCFGCTL – FPGA Reconfiguration Registers

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0xD2)</td>
<td>FCFGDAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>FPGA Reconfiguration Data Register</td>
</tr>
<tr>
<td>Read/Write</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(0xD1)</td>
<td>FCFGSTS</td>
<td>FCFGDN</td>
<td>FCFGOK</td>
<td>FCFGFAIL</td>
<td>FCFGRDY</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W1C</td>
<td>R/W1C</td>
<td>R/W1C</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(0xD0)</td>
<td>FCFGCTL</td>
<td>–</td>
<td>FCFGSEC</td>
<td>–</td>
<td>FCFGCMD</td>
<td>FCFGEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

These registers are used during reconfiguration of the FPGA and are not intended for customer use. FCFGEN auto-clears after a reconfiguration is complete. The Data register is a write-only register.

### 6.1.7  SVPWH, SVPWL, SVCR – Servo XB Registers

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0xFD)</td>
<td>SVPWH</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Servo Pulse Width High Register</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(0xFC)</td>
<td>SVPWL</td>
<td>Servo Pulse Width Low Register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(0xFA)</td>
<td>SVCR</td>
<td>SVEN</td>
<td>SVDIS</td>
<td>SVUP</td>
<td>SVCHAN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>W</td>
<td>W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The servo data registers SVPWH and SVPWL represent the desired servo pulse width in microseconds. The value is programmed to the channel selected by SVCHAN when the SVCR register is written with the update (SVUP) bit set. The channel can be enabled to begin at the same time by also setting the enable (SVEN) bit. A channel is disabled by writing SVCR with the desired channel in the SVCHAN field, the SVEN bit clear and the SVDIS set. The pulse width of a channel can be changed without changing its enabled/disabled status by leaving the SVEN and SVDIS bits clear when writing the SVCR register. SVDIS and SVUP are strobes and will always read zero. Reading SVEN will give the current enabled/disabled status of the channel read in the SVCHAN field. The value of SVCHAN corresponds to the Arduino pin to use (i.e. 0=RX, 1=TX, 2=D2, ..., 14=A0, etc.). Multiple pins can be driven simultaneously, each with a different pulse width...with a small limitation. The 32 possible values of SVCHAN directly alias to the 16 available timers (e.g. channels 1 and 17 could both be enabled, but they would always have the same pulse width of whichever one was programmed most recently). The easiest way to use these registers is with the XLR8Servo library ([https://github.com/AloriumTechnology/XLR8Servo](https://github.com/AloriumTechnology/XLR8Servo)).
## 6.1.8 NEOD2, NEOD1, NEOD0, NEOCR – NeoPixel XB Registers

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0xF7)</td>
<td>NEOD2</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>NeoPixel Data 2 register, function depends on NEOCMD.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Read/Write</td>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(0xF6)</td>
<td>NEOD1</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>NeoPixel Data 1 register, function depends on NEOCMD, often high half of pixel address/length.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Read/Write</td>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(0xF5)</td>
<td>NEOD0</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>NeoPixel Data 0 register, function depends on NEOCMD, often low half of pixel address/length.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Read/Write</td>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(0xF4)</td>
<td>NEOCR</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>NEOCR, NEOPIN, NEOCMD.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Read/Write</td>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The NeoPixel register usage varies depending on the value of the NEOCMD field in the NeoPixel control register (NEOCR).

**NEOPIN**: For the show command, this specifies which pin to send the data to. For no-op a pin number or other ID can be used. For other commands this field is ignored and not stored. For show the value clears to zero when the command completes and thereby can be used as a busy indication. While the hardware doesn’t provide any locks, this may help software manage having multiple objects share this hardware nicely. The outputs are indexed starting at 1 because using the clear to zero to indicate busy/notbusy wouldn’t work with pin 0.

**NEO CMD**:

- 0000 = no-op, but set PinNum/BusyID (won’t autoclear until after show)
- 0001 = get memsize. D2=cmd buf size (entries), D1/D0= pixel mem size (bytes)
- 0010 = show WS2812. D2=starting cmd buffer D1/D0=length (bytes)
- 0011 = show WS2811. D2=starting cmd buffer D1/D0=length (bytes)
- 0100-0111 = Reserved.
- 1000 = set color. D2=color value, D1/D0=memory address (autoincrement)
- 1001 = set cmd buf entry-addr. D2=cmd buf addr, D1/D0=section start addr
- 1010 = set cmd buf entry-length. D2=cmd buf addr, D1/D0=section length
- 1011 = set cmd buf entry-bright. D2=cmd buf addr, D1=reserved, D0=brightness
- 1100 = get color. D2=color value, D1/D0=memory address (autoincrement)
- 1101 = get cmd buf entry-addr. D2=cmd buf addr, D1/D0=section start addr
- 1110 = get cmd buf entry-length. D2=cmd buf addr, D1/D0=section length
- 1111 = get cmd buf entry-bright. D2=cmd buf addr, D1=reserved, D0=brightness

**NEOD2/D1/D0**: 8b data registers used as described above

For set, and show operations, D2/D1/D0 are loaded before doing the set/show cmd in the CNTL register (except possibly doing a no-op to set the busy ID).

For get color, D1/D0 are set first, then CNTL, then the color value can be read from D2.

For get cmd buf, D2 is set first, then CNTL, then the cmd buf info can be read from D1/D0.

For get memsize, CNTL is written, and then the memsize can be read from D2/D1/D0.

The easiest way to use these registers is with the XLR8NeoPixel library ([https://github.com/AloriumTechnology/XLR8NeoPixel](https://github.com/AloriumTechnology/XLR8NeoPixel)).
### 6.1.9 XFCTRL, XFSTAT, XFR0, XFR1, XFR2, XFR3—Floating Point XB Registers

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x38(0x58)</td>
<td>XFR3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XLR8 Function (floating point) 32 bit Result High Byte</td>
</tr>
<tr>
<td>0x3A(0x5A)</td>
<td>XFR2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XLR8 Function (floating point) 32 bit Result Byte</td>
</tr>
<tr>
<td>0x39(0x59)</td>
<td>XFR1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XLR8 Function (floating point) 32 bit Result Byte</td>
</tr>
<tr>
<td>0x3B(0x5B)</td>
<td>XFR0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XLR8 Function (floating point) 32 bit Result Low Byte</td>
</tr>
</tbody>
</table>

- **Read/Write:** R/W
- **Initial Value:** 0

A floating-point calculation is started by writing the XFSTART bit in the XFCTRL register, along with the desired operation in the XFCMD field (1=add, 2=multiply, 3=divide). Operands come directly from the AVR’s general-purpose register file (using our library ensures they will be in the right place). When the operation is done, the result appears in the XFR0/1/2/3 registers and the XFDONE status bit is set. If an unsupported XFCMD is used, the XFERR bit is also set, allowing software to revert to using a software-based calculation. The XFSTAT register auto-clears when it is read, or when the next operation is started via writing the XFSTART bit.

The easiest way to use these registers is with the XLR8Float library ([https://github.com/AloriumTechnology/XLR8Float](https://github.com/AloriumTechnology/XLR8Float)).

### 6.1.10 XLR8ADCR – XLR8 ADC Control Register

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7D</td>
<td>XLR8ADCR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Read/Write:** R/W
- **Initial Value:** 0

The AD12EN bit enables the ADC to run in 12 bit mode. The results reported in the ADCL and ADCH registers when running with ADLAR=0 can range from 0-4095, and when running with ADLAR=1, bits 5:4 of ADCL will include the least significant bits of the 12 bit ADC result. When running in 10 bit mode (the default to match Arduino), the result is truncated (not rounded) from the 12 bit result.

### 6.1.11 XLR8PID – XLR8 PID

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x6</td>
<td>PID_OP_L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Low Byte output</td>
</tr>
</tbody>
</table>

- **Read/Write:** R/W
- **Initial Value:** 0

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x5</td>
<td>PID_OP_H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>High Byte output</td>
</tr>
</tbody>
</table>

- **Read/Write:** R/W
- **Initial Value:** 0

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x3</td>
<td>PID_PV_L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Process variable low byte</td>
</tr>
</tbody>
</table>

- **Read/Write:** R/W
- **Initial Value:** 0

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2</td>
<td>PID_PV_H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Process variable high byte</td>
</tr>
</tbody>
</table>

- **Read/Write:** R/W
- **Initial Value:** 0
To start a channel typically the channel is reset first, then the control register with the desired channel indicated and both the enable and update bits set.

### 6.1.12 XLR8Quad – XLR8 Quadrature

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0xE9)</td>
<td>QERAT3</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>Upper 8 bits of quadrature rate data</td>
</tr>
<tr>
<td>Read/Write</td>
<td></td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(0xE8)</td>
<td>QERAT2</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>Upper-middle 8 bits of quadrature rate data</td>
</tr>
<tr>
<td>Read/Write</td>
<td></td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(0xE7)</td>
<td>QERAT1</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>Lower-middle 8 bits of quadrature rate data</td>
</tr>
<tr>
<td>Read/Write</td>
<td></td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(0xE6)</td>
<td>QERAT0</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>Lower 8 bits of quadrature rate data</td>
</tr>
<tr>
<td>Read/Write</td>
<td></td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(0xE5)</td>
<td>QECNT3</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>Upper 8 bits of quadrature count data</td>
</tr>
<tr>
<td>Read/Write</td>
<td></td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(0xE4)</td>
<td>QECNT2</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>Upper-middle 8 bits of quadrature count data</td>
</tr>
<tr>
<td>Read/Write</td>
<td></td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(0xE3)</td>
<td>QECNT1</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>Lower-middle 8 bits of quadrature count data</td>
</tr>
<tr>
<td>Read/Write</td>
<td></td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Name</td>
<td>Bit 7</td>
<td>Bit 6</td>
<td>Bit 5</td>
<td>Bit 4</td>
<td>Bit 3</td>
<td>Bit 2</td>
<td>Bit 1</td>
<td>Bit 0</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>------------------------</td>
</tr>
<tr>
<td>0xE2</td>
<td>QECNT0</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>Lower 8 bits of quadrature count data</td>
</tr>
<tr>
<td></td>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0xE0</td>
<td>QECR</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QEEN</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QEDIS</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QECLR</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QERATE</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QECHAN</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

To start a channel typically the channel is reset first, then the control register with the desired channel indicated and both the enable and update bits set.

## 7 Schematics and Layout


**Note:**

_Schematic released under the Creative Commons Attribution Share-Alike 3.0 License_

[https://creativecommons.org/licenses/by-sa/3.0](https://creativecommons.org/licenses/by-sa/3.0)
8 Credits

Some code is used and modified from the AVR core written by Ruslan Lepetenok (lepetenokr@yahoo.com) that is available at http://opencores.com/project,avr_core. Ruslan’s AVR core does not contain copyright or license notices, but we certainly wish to recognize its contribution to this project.

The I2C module builds upon the I2C core written by Richard Herveille (richard@asics.ws) that is available at http://opencores.org/project,i2c. The I2C core was released under BSD license with the following copyright statement:

```
Copyright (C) 2001 Richard Herveille
richard@asics.ws

This source file may be used and distributed without restriction provided that this copyright statement is not removed from the file and that any derivative work contains the original copyright notice and the associated disclaimer

THIS SOFTWARE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

All other portions of XLR8 were designed and verified for Alorium Technology by the team at Superion Technology.