Sno Edge 50™

Intel MAX 10 FPGA System on Module

User's Manual and Release Notes

April 11, 2022

Version	Date	Author	Changes
0.7	April 6, 2022	Steve Phillips	Initial Version
1.0	April 11, 2022	Steve Phillips	First official release

Table of Contents

1 Introduction

Sno Edge 50 is is an Intel MAX 10 FPGA System on Module (SOM) that includes an 8-bit AVR compatible microcontroller integrated on the FPGA for easy programmability and optimized access to the FPGA fabric for custom hardware functionality.

Based on Alorium Technology's very popular embeddable Snō FPGA module, the Sno Edge 50 enhances the powerful features and functionality of Snō with significantly increased digital I/O, additional ADCs, and more FPGA logic gates for custom Xcelerator Block development.

All of this functionality is packaged in a 200-pin SODIMM form factor for the ultimate in low-profile physical integration.

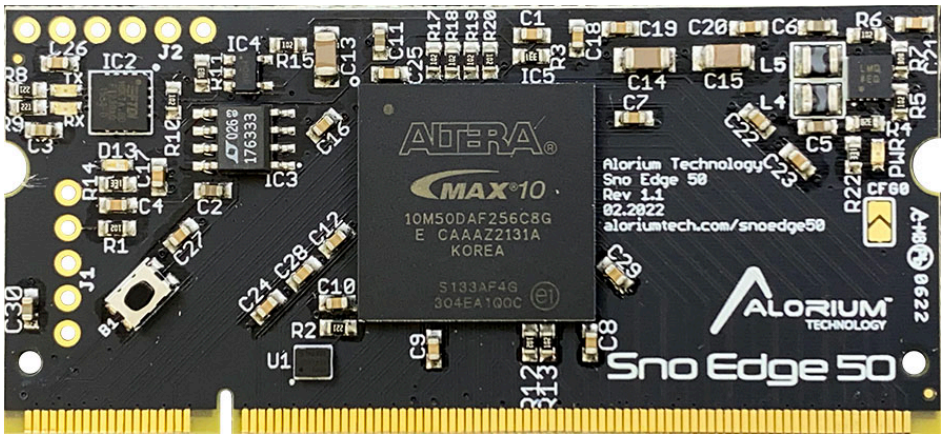


Figure 1: Sno Edge 50 Board

Note:

Sno Edge 50 is the first release in a planned roadmap of "Sno Edge" boards and is named based on the fact that it has a 50K LE MAX 10 FPGA. Additional variations of the design may be produced based on customer demand and FPGA availability.

For the remainder of this document, Sno Edge 50 is referred to as simply as "Sno Edge", and this label can be considered synonomous for the purposes of this manual.

2 Programming Sno Edge

2.1 Microcontroller Programming

The embedded microcontroller on Sno Edge is easily programmable with the Aruino IDE. Refer to the Appendix in Section 10 of this document for Arduino IDE installation instructions if you don't already have it installed on your development machine.

Other programming tools such as Atmel Studio, PlatformIO/VSCode, and others may also work for programming Sno Edge. However, Arduino is the only officially supported programming environment for Sno Edge.

2.1.1 USB Programming

Sno Edge is designed to be programmed via a USB connection.

There is an on-board FTDI USB-to-Serial translator chip that converts USB signals from the edge connector pins to serial UART commands used for programming the microcontroller.

Note: There is NO USB connector directly on Sno Edge. The physical USB connection will be made on the carrier board that is being used with Sno Edge.

For example, this image shows the USB connector on Alorium's Sno Edge test breakout board:

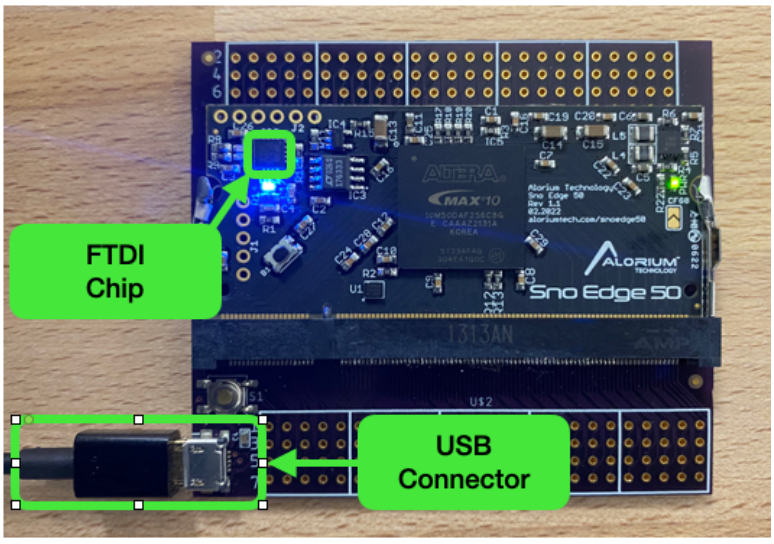


Figure 2: USB Connections

2.1.2 FTDI Programming

Sno Edge also has a 6-pin FTDI header at the top of Sno Edge that is used for initial microcontroller programming and test during the manufacturing process.

Figure 3: FTDI Vias

The FTDI interface can be used for general serial programming of microcontroller, as well; however, it does require using A USB-to-FTDI adapter of some kind. One of our favorites is the SparkFun Beefy 3 Basic FTDI Breakout.

2.2 FPGA Programming

The FPGA on Sno Edge comes pre-programmed with an image that includes the microcontroller as well as a pre-configured set of Xcelerator Blocks developed by Alorium Technology.

Alternate images can be uploaded directly through the Arduino IDE or accessed via our GitHub repo and flashed to the FPGA using a command-line program as described below.

2.2.1 Updating the FPGA Image

Sno Edge ships with a standard FPGA image that includes the 8-bit microcontroller and a small set of built-in Xcelerator Blocks.

This image can be updated with other images provided by Alorium Technology by using the "Burn Bootloader" command in the Arduino or by running a standalone command-line program.

Video Demonstration Examples

NOTE:
The following videos were originally created for our XLR8 board as demonstrations for how to upload new FPGA images from the Arduino IDE or with our command line program. The process for Sno Edge 50 can be accomplished by selecting Sno Edge 50 as the board, instead of XLR8.

Demonstration videos using Sno Edge will be available soon on our YouTube channel, and this manual will be updated to reflect the new tutorials.

2.2.1.1 Flashing A New FPGA Image via Arduino IDE

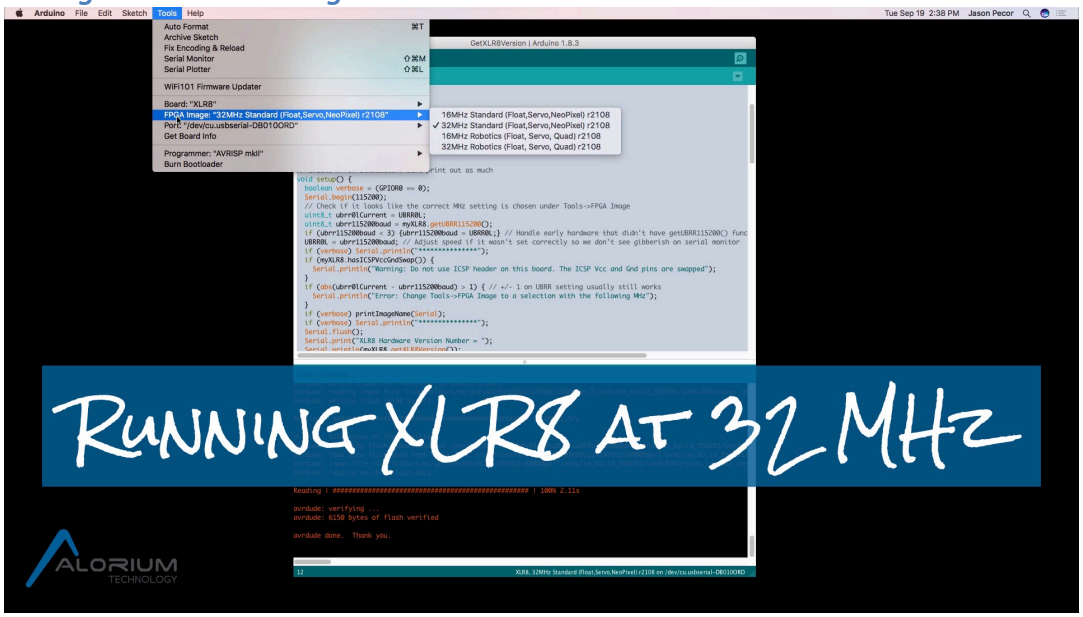


Figure 4: Arduino IDE Video

2.2.1.2 Flashing A New FPGA Image Using the Command-Line

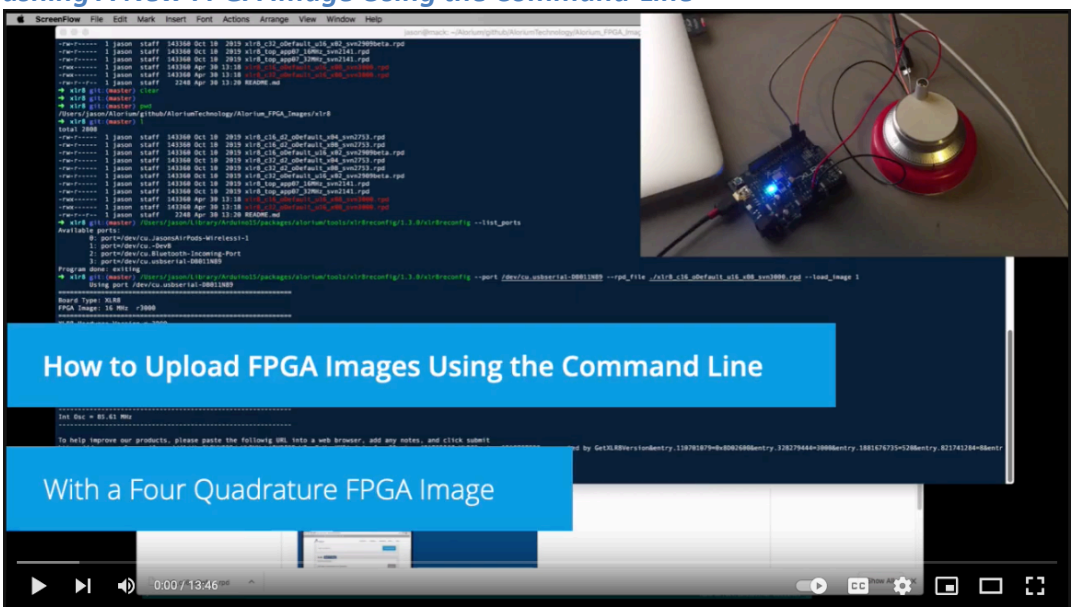


Figure 5: Command Line Video

2.2.2 Restoring Factory FPGA Image

The FPGA on Sno Edge can hold two different FPGA images. One of those, the User Image, can be reconfigured with new images to take advantage of increased functionality as new features are

introduced and released. The other image, the Factory Image, is never changed, and is typically unused unless the primary image 1 becomes corrupted.

If necessary, a "factory reset" of Sno Edge can be performed by bridging the two sides of the split CFG0 pad while applying power to the board. It only takes a momentary grounding to cause this to happen. See Figure 6 for the location of the CFG0 pad.

After power-up, the Factory Image will be loaded. However, any loss of power to the board will result in the corrupted image being reloaded. Therefore, the user will want to flash a known-good image into the User Image before proceeding.

2.2.2.1 Locating CFG0

Figure 6: CFG0 Location

2.2.3 Creating Custom FPGA Images with OpenXLR8

As with all of our products, the FPGA can be programmed with your own custom FPGA image by using our OpenXLR8 FPGA process. OpenXLR8 is the methodology that allows users of all our XLR8 products to develop their own custom Xcelerator Blocks and integrate them into the FPGA.

You can learn more about how to use OpenXLR8 here:

> **Introduction to OpenXLR8** <https://aloriumtech.com/openxlr8/>

2.2.4 Bare-Metal FPGA Programming

For advanced FPGA users, Sno Edge does have a JTAG header that can be used for creating bare-metal FPGA designs and directly flashing a new image to the FPGA.

Use of the JTAG interface will require that user has the appropriate JTAG programming hardware such as the JTAG Blaster programmer from Intel.

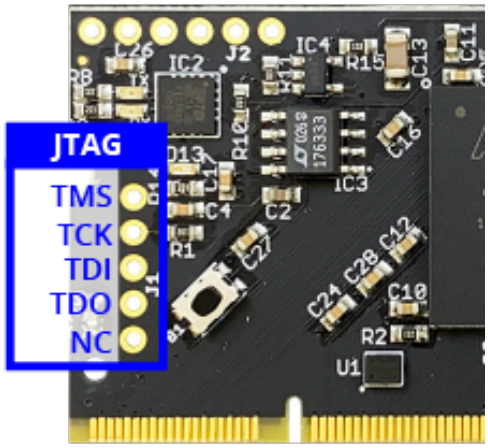


Figure 7: JTAG Vias

IMPORTANT NOTE!!

If the JTAG interface is used to load the MAX10 FPGA with a custom image, it is possible to erase the production Sno Edge functionality, deleting the factory production image and the integrated 8-bit microcontroller subsystem. In this scenario, loading images through the Arduino IDE would no longer be possible.

The Sno Edge FPGA has been designed to be modified and extended by using Alorium's OpenXLR8 Methodology. This flow provides a path to create custom XBs in the FPGA fabric that can easily interface to the on-chip microcontroller and preserve full factory functionality.

Learn More about OpenXLR8 here: https://www.aloriumtech.com/openxlr8

3 General Technical Specifications

3.1 3.3V I/O

Sno Edge is a 3.3V device, and users are cautioned to only connect to other 3.3V devices.

The Sno Edge does not come equipped with pull-up resistors so the user is required to add them physically as needed, except for the dedicated SDA and SCL pins which have 1K ohm pull-ups. .

3.2 ADC

Sno Edge supports 16 ADC inputs via two eight input ADC modules. The ADCSRB register now supports the MUX5 bit (bit 5) which is used to select which ADC input is read.

The exact mapping of the ADC inputs is show in Figure 8.

ADC Input	ADCSRB[5] MUX5	ADMUX[2:0] MUX[2:0]	FPGA Input	Edge Connector	Arduino Label
0	0	0	ADC1[1]	26	A0
1	0	1	ADC1[2]	24	A1
2	0	2	ADC1[3]	31	A2
3	0	3	ADC1[4]	29	A3
4	0	4	ADC1[5]	25	A4
5	0	5	ADC1[6]	23	A5
6	0	6	ADC1[7]	19	A6
7	0	7	ADC1[8]	17	A7
8	1	0	ADC2[1]	16	A8
9	1	1	ADC2[2]	13	A9
10	1	2	ADC2[3]	20	A10
11	1	3	ADC2[4]	22	A11
12	1	4	ADC2[5]	7	A12
13	1	5	ADC2[6]	5	A13
14	1	6	ADC2[7]	11	A14
15	1	7	ADC2[8]	14	A15

Figure 8: Dual ADC Input Mapping

An Arduino variant has been defined for the Sno Edge so the user does not need to be concerned with the mapping of ADC inputs to ADCSRB/ADMUX register fields. The user can simply reference the Arduino Label in their sketch. For example: `analogRead(A12)`, to read the ADC input on Edge Connector pin 7.

Sno Edge is only able to measure against an internal 3.3V reference. The ADC inputs themselves are limited to a max input voltage of 2.5 volts.

The temperature sensor are not implemented (ADMUX=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.

3.3 Analog Compare

Sno Edge does not support 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, 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.

3.4 Power

There are 3 ways to power the Sno Edge module:

- Connect a 3.3V FTDI breakout board or cable to the FTDI interface
- Supply 5V via pin 2 of the SODIMM connector (this is intended to come from a USB connector on the system board). This will use an on-board 3.3V regulator to supply power to the FPGA and other components on the board, and is limited to 500mA
- Supply 3.3V via the dedicated power pins on the SODIMM connector (this is the most robust way to power the Sno Edge board)

3.5 Pin 13 LED

As with many other Arduino-compatible boards, digital pin 13 is used for both the on-board LED as well as the SPI clock, SCK. On Sno Edge, SCK and the LED are driven from separate FPGA pins which are logically equivalent but physically separate, in order to avoid the extra loading the LED can cause.

4 Xcelerator Blocks (XBs)

Xcelerator Blocks are custom hardware blocks implemented within the Sno Edge 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 Sno Edge with the Arduino toolset. Since an FPGA can be reprogrammed many times, a single Sno Edge can be reconfigured to incorporate different XBs depending on the project requirements.

Sno Edge ships with three sample XBs: Floating Point, Quadrature, and Servo Control. The software libraries are delivered as .zip files from our github site (https://github.com/AloriumTechnology). They are installed like other Arduino .zip libraries as described here (https://www.arduino.cc/en/Guide/Libraries).

4.1 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.

4.2 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 Port K 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 `#include <Servo.h>` to `#include <XLR8Servo.h>`

The servos are connected to the physical pins starting with Port KA, pin 0, going through port KD, pin 7, with each servo connected to the one sequential pins in order. So, servo 0 is tied to KA[0], sevo 1 is tied to KA[1], servo 31 is tied to KD[7], etc. You can instantiate an array like this: `Servo servo[32];`

4.3 Quadrature

The Sno Edge builtin Quadrature XB provides up to 16 Quadrature encoders. These are connected to Port J, which is the concatination of ports {JD,JC,JB,JA}.

As quadrature objects are instantiated, they are created sequentially. I.e., the first quadrature object will control quadrature 0 in the fabric, the second will control quadrature 1, etc., through quadrature 16.

The quadratures are connected to the physical pins starting with Port JA, pin 0, going through port JD, pin 7, with each quadrature connected to the two sequential pins in order. So, quadrature 0 is tied to JA[1:0], quadrature 15 is tied to JD[7:6], etc. The simplest way to manage multiple quadratures in an application is to create an array of quadrature objects. You can instantiate an array like this:

```
Quadrature quadratures[16];
```

The XLR8Quadrature library is included with the line

```
#include <XLR8Quadrature.h>
```

Once you instantiate an quadrature object, the quadrature is enabled by default. The software library then allows you to disable & re-enable the quadratures, and read the count and rate values of the quadrature. By default, the quadrature samples every 200ms to get the rate, but can be set to sample every 20ms instead.

5 Pin Mapping

With a handful of exceptions, the pins are arranged into 18 ports, each of which can be up to 8 bits wide. They can be organized into four groups:

1. First there are the standard Arduino Uno ports: D, B, and C. Note that ports B and C are only 6 bits wide. In the case of Port C this creates a gap in the "D" pin numbering since there is no C[7:6] which would correspond to D pins [21:20].
2. Then there are the standard Sno extended ports A, E, and G. All boards in the Sno family implement these three extended ports.
3. Following the Sno extended ports are the J and K ports. There are four J ports (JA, JB, JC, and JD) and four K ports (KA, KB, KC, and KD). These can be treated in software either as 32 bit ports or as four 8 bit ports. The Sno Edge variant provides support for both.
4. Finally, the PL port provide four pins that can also be used as two differential PLL inputs

Aside from the port pins, there are various non-port pins that provide specific functionality:

- Clock
- Reset
- ADC
- ADC Reference
- I2C
- Serial

In the following figures a Color Key is used to indicate how the various types of pins are organized:

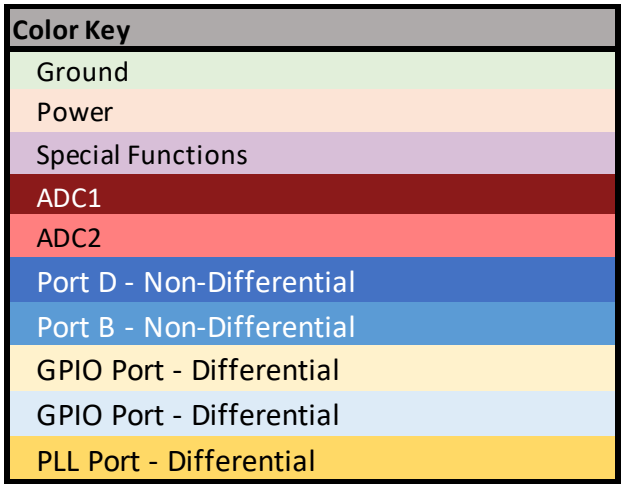


Figure 9: Pin Map Color Key

In Figure 10 the ports are enumerated and the bit ranges are specified. Of special interest is the numbering gap in the D Nums column at [21:20], as discussed above. The XB Busses, which are used in the OpenXLR8 module, do not have a gap at [21:20] and so are offset by 2 from the D Nums for pins above 19.

The Int Bit column indicates which bit in the SPCIFR register will get set when there is a pin change interrupt for that port.

The GPIO column indicates whether the pins in that port are differential pairs or not. A “D” indicates a differential pair port. By default all pins are normal non-differential GPIO pins in these ports, but it is possible in the OpenXLR8 methodology to change the configuration of those pins to be differential.

Port Name	Port Bits	D Nums	XB Busses	Int Bit	GPIO
D	8	7: 0	[7:0]		S
B	6	13: 8	[13:8]		S
C	6	19: 14	[19:14]		D
A	6	27: 22	[25:20]	0	D
E	6	33: 28	[31:26]	1	D
G	8	41: 34	[39:32]	2	D
JA	8	49: 42	[47:40]	3	D
JB	8	57: 50	[55:48]	3	D
JC	8	65: 58	[63:56]	3	D
JD	8	73: 66	[71:64]	3	D
KA	8	81: 74	[79:72]	4	D
KB	8	89: 82	[87:80]	4	D
KC	8	97: 90	[95:88]	4	D
KD	8	105: 98	[103:96]	4	D
PL	4	109: 106	[107:104]	5	D

Figure 10: Port Numbering

In Figure 10, the mapping between the FPGA pins and the Sno Edge connector pins is shown. The table is split two halves representing the odd side and the even side of the connector. For each connector pin, the following information is shown:

Column	Description
Edge Connector Pin	The pin number of the edge connector
FPGA D Pin	The “D” pins are numbered from 0 up to 109. These numbers can be used directly to specify pins in functions such as digitalWrite()
FPGA Port Bit	The pins are all arranged into ports of up to 8 bits. The Port Bit indicats which pin in a port
FPGA Pad	The FPGA Pad specifies which physical pin on the FPGA the corresponding signal is using.
FPGA Pin Type	The Pin Type indicates any important type desciptor for FPGA Pad, such as Diff pair, GND, or VCC
Special Function	Special Function uiindicates any special note about that pin

Figure 11: Edge Connector Table Information

In Figure 12, each differential pair is indicated by a box around the two pins. For instance, edge pins 30 and 32 are a differential pair.

Odd Side of Connector						Even Side of Connector					
Special Function	FPGA Pin Type	FPGA PAD	FPGA Port Bit	FPGA D Bit	Edge Pin	Edge Pin	FPGA D Bit	FPGA Port Bit	FPGA PAD	FPGA Pin Type	Special Function
ADC_REF	ADC_REF	E4	ADC_REF	na	1	2				5.0V	
	GND				3	4				USB-	Serial
ADC2in6	L6P	B2	ADC2[6]	na	5	6				USB+	Serial
ADC2in5	L6N	C2	ADC2[5]	na	7	8				GND	
	GND				9	10	na	RESET_N	B10	RESETN	RESET_N
ADC2in7	L8N	B1	ADC2[7]	na	11	12				GND	
ADC2in2	L8P	C1	ADC2[2]	na	13	14	na	ADC2[8]	C3	L2P	ADC2in8
	GND				15	16	na	ADC2[1]	C4	L2N	ADC2in1
ADC1in8	L7P	D1	ADC1[8]	na	17	18				GND	
ADC1in7	L7N	E1	ADC1[7]	na	19	20	na	ADC2[3]	E3	L4N	ADC2in3
	GND				21	22	na	ADC2[4]	F2	L4P	ADC2in4
ADC1in6	L5P	F1	ADC1[6]	na	23	24	na	ADC1[2]	F4	L1P	ADC1in2
ADC1in5	L5N	G2	ADC1[5]	na	25	26	na	ADC1[1]	F5	L1N	ADC1in1
	GND				27	28				GND	
ADC1in4	L3P	G5	ADC1[4]	na	29	30	14	C[0]	J2	L22P	
ADC1in3	L3N	H5	ADC1[3]	na	31	32	15	C[1]	J3	L22N	
	GND				33	34				GND	
	L20P	H6	C[2]	16	35	36	18	C[4]	K2	L29P	
	L20N	J5	C[3]	17	37	38	19	C[5]	L1	L29N	
	GND				39	40				GND	
	Connector Gap									Connector Gap	
	GND				41	42				GND	
	L36N	J6	A[0]	22	43	44	24	A[2]	K5	L41N	
	L36P	K6	A[1]	23	45	46	25	A[3]	L6	L41P	
	GND				47	48				GND	
	L37P	L2	A[4]	26	49	50	na	Clock	L3	FPGA_CLK_P	CLK
	L37N	M2	A[5]	27	51	52	na	Clock_n	M3	FPGA_CLK_N	CLK
	GND				53	54				GND	
	B2P	R1	E[0]	28	55	56	30	E[2]	P1	L38P	
	B2N	P2	E[1]	29	57	58	31	E[3]	N2	L38N	
	GND				59	60				GND	
	B6P	T2	E[4]	32	61	62	34	G[0]	R2	B4P	
	B6N	T3	E[5]	33	63	64	35	G[1]	R3	B4N	
	GND				65	66				GND	
PLL	L59N	N3	PL[0]	106	67	68	36	G[2]	R4	B5N	
PLL	L59P	N4	PL[1]	107	69	70	37	G[3]	P5	B5P	
	GND				71	72				GND	
	B1N	P4	G[4]	38	73	74	40	G[6]	T4	B14P	
	B1P	N5	G[5]	39	75	76	41	G[7]	T5	B14N	
	GND				77	78				GND	
no diff pair	B13P	R5	D[0]	0	79	80	1	D[1]	R6	B13N	no diff pair
	3.3V				81	82				3.3V	
	B16P	P6	J[0]	42	83	84	44	J[2]	R8	B17P	
	B16N	R7	J[1]	43	85	86	45	J[3]	T8	B17N	
	3.3V				87	88				3.3V	
no diff pair	B15P	L8	D[2]	2	89	90	3	D[3]	M7	B15N	no diff pair
	B3N	M6	J[4]	46	91	92	48	J[6]	M8	B20N	
	B3P	L7	J[5]	47	93	94	49	J[7]	M9	B20P	
	3.3V				95	96				3.3V	
no diff pair	B18P	P8	D[4]	4	97	98	5	D[5]	P9	B18N	no diff pair
	B36P	L9	J[8]	50	99	100	52	J[10]	R9	B19P	
	B36N	M10	J[9]	51	101	102	53	J[11]	T9	B19N	
	3.3V				103	104				3.3V	
	B22P	R10	J[12]	54	105	106	56	J[14]	P10	B34N	
	B22N	T11	J[13]	55	107	108	57	J[15]	P11	B34P	
SDA	B57N	M11	SDA	na	109	110	na	SCL	L10	B57P	SCL
	3.3V				111	112				3.3V	
	B35N	R11	J[16]	58	113	114	60	J[18]	P12	B37P	
	B35P	R12	J[17]	59	115	116	61	J[19]	P13	B37N	
	3.3V				117	118				3.3V	
	R2P	T14	D[6]	6	119	120	7	D[7]	T15	R2N	no diff pair
	GND				121	122				GND	
	R1N	R14	J[20]	62	123	124	64	J[22]	N14	R27P	
	R1P	P14	J[21]	63	125	126	65	J[23]	P15	R27N	
	GND				127	128				GND	
	R29N	P16	J[24]	66	129	130	8	B[0]	L12	R25N	no diff pair
	R29P	N16	J[25]	67	131	132				GND	
	GND				133	134	68	J[26]	M14	R28N	
	R33P	M16	J[28]	70	135	136	69	J[27]	M15	R28P	
	R33N	L16	J[29]	71	137	138				GND	
	GND				139	140	72	J[30]	K14	R32P	
	R30P	K11	K[0]	74	141	142	73	J[31]	L15	R32N	
	R30N	K12	K[1]	75	143	144				GND	
	GND				145	146	76	K[2]	J11	R38P	
no diff pair	R39N	K15	B[1]	9	147	148	77	K[3]	J12	R38N	
	GND				149	150				GND	
	R40P	J15	K[4]	78	151	152	80	K[6]	H15	R41P	
	R40N	J16	K[5]	79	153	154	81	K[7]	H16	R41N	
	GND				155	156				GND	
	R44P	H11	K[8]	82	157	158	84	K[10]	G11	R50P	
	R44N	H12	K[9]	83	159	160	85	K[11]	G12	R50N	
	GND				161	162				GND	
pair w/170	R46N	G15	B[2]	10	163	164	86	K[12]	G16	R47P	
	GND				165	166	87	K[13]	F16	R47N	
	R51P	F14	K[14]	88	167	168				GND	
	R51N	E14	K[15]	89	169	170	11	B[3]	G14	R46P	pair w/163
	GND				171	172				GND	
	R52P	E15	K[16]	90	173	174	108	PL[2]	D14	R69P	PLL
	R52N	E16	K[17]	91	175	176	109	PL[3]	C14	R69N	PLL
	GND				177	178				GND	
	R42P	D16	K[18]	92	179	180	94	K[20]	D15	R70P	
	R42N	C16	K[19]	93	181	182	95	K[21]	C15	R70N	
	GND				183	184				GND	
no diff pair	T17N	F12	B[4]	12	185	186	96	K[22]	C13	T2P	
	GND				187	188	97	K[23]	C12	T2N	
	T30P	B13	K[24]	98	189	190				GND	
	T30N	A13	K[25]	99	191	192	100	K[26]	D12	T1P	
	GND				193	194	101	K[27]	E11	T1N	
	T28P	F10	K[28]	102	195	196				GND	
	T28N	E10	K[29]	103	197	198	104	K[30]	B12	T41P	
no diff pair	T43P	A11	B[5]	13	199	200	105	K[31]	B11	T41N	

Figure 12: Pin Mapping

6 Extended Interrupts

The Sno Edge extends the AVR architecture to implement additional interrupts for extended GPIO pin change interrupts and for user-defined interrupts in the OpenXLR8 methodology.

6.1 GPIO Port Pin Change Detection

The Sno Edge extended GPIO ports support pin change detection in a way similar to the standard ports. Port pins are monitored and if a pin change is detected, an interrupt can be generated.

Each GPIO port has a PCMSK that can be used to enable pin change interrupts on a per-pin basis. The Sno Edge extended GPIO PCMSKs are:

- PCMSKA
- PCMSKE
- PCMSKG
- PCMSKPL
- PCMSKJA
- PCMSKJB
- PCMSKJC
- PCMSKJD
- PCMSKJA
- PCMSKJB
- PCMSKJC
- PCMSKJD

The PCMSK register contains a bit for each pin in the port. A PCMSK bit value of zero will prevent a pin change on the corresponding port pin from causing an interrupt signal to be generated. The PCMSK does not support bit set or bit clear operations, so a read-modify-write operation should be used to change individual bits.

6.2 Pin Change Interrupts

Pin Change notifications from the ports are collected and controlled by three registers:

Register	Description
SPCIFR	Sno Pin Change Interrupt Flag Register
SPCICR	Sno Pin Change Interrupt Control Register
SPCIMSK	Sno Pin Change Interrupt Mask Register

Figure 13: Sno Pin Change Interrupt Registers

The bits in the above registers correspond to the ports in the following way. Notice that the four Jx ports and the four Kx ports are combined into single bits:

Bit	Interrupt Source
0	Port A
1	Port E
2	Port G
3	Port J (JA or JB or JC or JD)
4	Port K (KA or KB or KC or KD)
5	Port PL

Figure 14: Sno Pin Change Interrupt Fields

A bit in the Flag register (SPCIFR) will be set when a pin change notification is received if the corresponding bit in the Mask register (SPCIMSK) is set.

A bit in the Flag register is cleared via software by writing a one to the bit.

A bit set in the Flag register will cause an IRQ to be generated if the corresponding bit in the Control register (SPCICR) is set.

Neither the Mask register nor the Control register support bit operations, so a read-modify-write operation should be used to change individual bits.

6.3 OpenXLR8 Interrupts

Interrupts from XBs instantiated within the OpenXLR8 Module are collected and saved in the `xlr8_pcint` module.

Register	Description
OX8IFR	Sno Pin Change Interrupt Flag Register
OX8ICR	Sno Pin Change Interrupt Control Register
OX8MSK	Sno Pin Change Interrupt Mask Register

Figure 15: OpenXLR8 Interrupt Registers

The bits in the above registers are defined by the OpenXLR8 developer and are specific to that particular implementation.

A bit in the Flag register (OX8IFR) will be set when a pin change notification is received if the corresponding bit in the Mask register (OX8MSK) is set.

A bit in the Flag register is cleared via software by writing a one to the bit.

A bit set in the Flag register will cause an IRQ to be generated if the corresponding bit in the Control register (OX8ICR) is set.

Neither the Mask register nor the Control register support bit operations, so a read-modify-write operation should be used to change individual bits.

6.4 Extended IRQs

The IRQs from the GPIO pin change interrupts and the OpenXLR8 interrupts are managed by the following registers

Register	Description
XIFR	eXtended IRQ Flag Register
XICR	eXtended IRQ Control Register
XMSK	eXtended IRQ Mask Register
XACK	eXtended IRQ Acknowledge Register

Figure 16: Extended IRQ Registers

The bits in the above registers correspond to the interrupt sources in the following way:

Bit	Interrupt Source	IRQ Num	AVR Name	XLR8/Sno Alias
0	SPCIFR	23	EE_READY_vect	XGPIO_vect, BIXB_vect
1	OX8IFR	24	ANALOG_COMP_vect	OPENXLR8_vect
7:2	Unused			

Figure 17: Extended IRQ Fields

The AVR supports a specified set of IRQ vectors, specified by integers. The Sno Edge, and XLR8 boards in general, reassign two of the defined IRQ vectors to support the new extended GPIO pin change interrupts and the OpenXLR8 interrupts. Those reassigned vectors are indicated above.

A bit in the Flag register (XIFR) will be set when an IRQ is received if the corresponding bit in the Mask register (XMSK) is set and the corresponding bit in the Acknowledge (XACK) register is not set.

A bit in the Flag register is cleared either by the corresponding bit in the Acknowledge register being set, or by the source of the IRQ being cleared.

A bit set in the Flag register will cause an IRQ to be generated if the corresponding bit in the Control register (XICR) is set.

When an IRQ is generated to the AVR core it will respond by setting a bit in the acknowledge register. This will block the corresponding bit in the Flag register from being set, preventing further IRQs of that type from being sent to the AVR core. The bit in the Acknowledge must be cleared by software once the interrupt has been serviced and control is returned to the original program. Bits in the Acknowledge register can be cleared by writing a one to the corresponding bit location in the Acknowledge register

Neither the Mask register nor the Control register support bit operations, so a read-modify-write operation should be used to change individual bits.

6.5 Setup and Usage

The default values for the interrupt related registers are all zeros. This disables all interrupts. In order to enable interrupts for a pin they must be configured:

1. Set the mask bits for the pin and port that is to be enabled by writing PCMSKxx for that port.
2. Set the SPCICR enable bit that corresponds to the port that is being enabled.
3. Set the SPCIMSK mask bit that corresponds to the port that is being enabled.
4. Set the XICR enable bit that corresponds to the port that is being enabled.
5. Set the XMSK mask bit that corresponds to the port that is being enabled.

When an IRQ is received by the AVR core it will trigger an Interrupt Service Routine (ISR) associated with the interrupt to be called. It will also set the bit in XACK corresponding to the interrupt vector.

After the interrupt has been handled by the ISR the interrupts should be re-enabled by:

1. Clear the SPCIFR bit or OX8IFR bit by writing a one to it
2. Clear the XACK bit by writing a one to it

Interrupt Service Routine functions can be specified using the XLR8 IRQ aliases specified in Figure 17. Simply specify the desired XLR8 IRQ alias name in the `ISR()` function call.

```
ISR(XGPIO_vect) {      // Extended GPIO Port Pin Change Interrupts
  // Enter ISR code
}

ISR(OPENXLR8_vect) {   // OpenXLR8 Interrupts
  // Enter ISR code
}
```

6.6 Example Interrupt Sketch

The following example sketch sets up Port G, Pin 0, for a pin change interrupt. To test this in hardware simply start the sketch and then simply ground Port G, Pin 0 momentarily. This should cause a pin change interrupt and the sketch will print `" loop():  Interrupt detected... "` each time the pin changes.

```
////////////////////////////////////////////////////////////////
//=============================================================
//  Copyright(c) Alorium Technology Inc., 2022
//  ALL RIGHTS RESERVED
//=============================================================
//
// File name:   : snoedge_int_example.ino
// Author       : support@aloriumtech.com
```

```
// Description : Demonstrate pin change interrupts on Sno Edge
//               extended GPIO port
//
////////////////////////////////////////////////////////////////

// Variables to use in the ISR routine. Use volatile to make sure
// value is maintained across ISR calls
volatile bool isr_found = false;

void setup() {

  Serial.begin(115200);
  Serial.println("========== Start snoedge_int_example.ino ==========");
  Serial.println(" Enter setup(): Configure pin change interrupt on Port G, Pin 0");

  // Enable Port G, pin 0 for pin change interrupts
  PCMSKG  |= (1 << MSKG0);

  // Enable the SPCIFR bit for Port G to be enabled for pin change interrupts
  SPCIMSK |= (1 << SPCIPG); // Set the bit for Port G in the mask reg
  SPCICR  |= (1 << SPCIPG); // Set the bit for Port G in the control reg

  // Enable the XIFR bit for pin change interrupts
  XMSK    |= (1 << XIGPIO); // Set the bit for Pin Change IRQ in the mask reg
  XICR    |= (1 << XIGPIO); // Set the bit for Pin Change IRQ in the control reg
  Serial.println(" Enter loop():  Check for interrupt and reset after response");
}

void loop() {
  // Check for ISR
  if (isr_found) {
    Serial.print(" loop():        Interrupt detected... ");

    // Code for interrupt response goes here
    //    ...
    // After Interrupt response, clear interrupt flag and re-enable
    // IRQ by clearing the XACK bit
    SPCIFR = (1 << SPCIPG); // Write one to flag bit for Port G
    XACK   = (1 << XIGPIO); // Write one to clear the ACK

    // Reset the isr_found flag so that we break out of the loop
    isr_found = false;
    Serial.println(" Interrupt handled");
  }

  // Wait a bit before checking the isr_found flag again
  delay(1);
}

ISR(XGPIO_vect) { // Extended GPIO Port Pin Change Interrupts
  // This ISR will be involked when a pin change interrupt
  // is triggered. Keep the interrupt service routine short
  // by just setting a flag and returning. The flag will be
  // checked in the main loop() function.
  isr_found = true;
}
```

Sample output:

```
========== Start snoedge_int_example.ino ==========
```

```
Enter setup(): Configure pin change interrupt on Port G, Pin 0
Enter loop():  Check for interrupt and reset after response
loop():        Interrupt detected...  Interrupt handled"
loop():        Interrupt detected...  Interrupt handled"
```

7 Register Summary

The registers used in Sno Edge are listed below. The table is color coded to indicate whether the registers are as defined for the ATmega328p, or whether they have been changed in some way. The color key can be found at the bottom of the table.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
(0xFF)	XDINFO	XBINFOAD								**Error! Reference source not found.**
(0xFE)	Reserved	–	–	–	–	–	–	–	–	
(0xFD)	SVPWH	–	–	–	–	Servo Pulse Width High Register				7.1.17
(0xFC)	SVPWL	Servo Pulse Width Low Register								7.1.17
(0xFB)	SVCR	SVEN	SVDIS	SVUP	SVCHAN					7.1.17
(0xFA)	Reserved		–	–	–	–	–	–	–	
(0xF9)	Reserved	–	–	–	–	–	–	–	–	
(0xF8)	Reserved	–	–	–	–	–	–	–	–	
(0xF7)	Reserved	–	–	–	–	–	–	–	–	
(0xF6)	PID_OP_L	Low Byte output								7.1.16
(0xF5)	PID_OP_H	High Byte output								7.1.16
(0xF4)	PID_PV_L	Process variable low byte								7.1.16
(0xF3)	PID_PV_H	Process variable high byte								7.1.16
(0xF2)	PID_SP_L	Set point low byte								7.1.16
(0xF1)	PID_SP_H	Set point high byte								7.1.16
(0xF0)	PID_KP_L	KP coefficient low byte								7.1.16
(0xEF)	PID_KP_H	KD coefficient high byte								7.1.16
(0xEE)	PID_KI_L	KI coefficient low byte								7.1.16
(0xED)	PID_KI_H	KI coefficient high byte								7.1.16
(0xEC)	PID_KD_L	KD coefficient low byte								7.1.16
(0xEB)	PID_KD_H	KD coefficient high byte								7.1.16
(0xEA)	PIDCR	PEDEN	PIDDIS	PIDUPD	PIDCHAN					7.1.16
(0xE9)	QERAT3	Upper 8 bits of quadrature rate data								7.1.15
(0xE8)	QERAT2	Upper-middle 8 bits of quadrature rate data								7.1.15
(0xE7)	QERAT1	Lower-middle 8 bits of quadrature rate data								7.1.15
(0xE6)	QERAT0	Lower 8 bits of quadrature rate data								7.1.15
(0xE5)	QECNT3	Upper 8 bits of quadrature count data								7.1.15
(0xE4)	QECNT2	Upper-middle 8 bits of quadrature count data								7.1.15
(0xE3)	QECNT1	Lower-middle 8 bits of quadrature count data								7.1.15
(0xE2)	QECNT0	Lower 8 bits of quadrature count data								7.1.15
(0xE1)	Reserved	–	–	–	–	–	–	–	–	
(0xE0)	QECR	QEEN	QEDIS	QECLR	QERATE	QECHAN				7.1.15
(0xDF)	Reserved	–	–	–	–	–	–	–	–	
(0xDE)	Reserved	–	–	–	–	–	–	–	–	
(0xDD)	Reserved	–	–	–	–	–	–	–	–	TWAMR1
(0xDC)	Reserved	–	–	–	–	–	–	–	–	TWCR1
(0xDB)	Reserved	–	–	–	–	–	–	–	–	TWDR1
(0xDA)	Reserved	–	–	–	–	–	–	–	–	TWAR1
(0xD9)	Reserved	–	–	–	–	–	–	–	–	TWSR!
(0xD8)	Reserved	–	–	–	–	–	–	–	–	TWBR1
(0xD7)	Reserved	–	–	–	–	–	–	–	–	
(0xD6)	XLR8VERT	XLR8 Version Number Flags								7.1.14
(0xD5)	XLR8VERH	XLR8 Version Number Register High Byte								7.1.14

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
(0xD4)	XLR8VERL	XLR8 Version Number Register Low Byte								7.1.14
(0xD3)	Reserved	–	–	–	–	–	–	–	–	
(0xD2)	FCFGDAT	FPGA Reconfiguration Data Register								7.1.13
(0xD1)	FCFGSTS	FCFGDN	0	FCFGFM	FCFGRDY	–	–	–	–	7.1.13
(0xD0)	FCFGCTL	–	FCFGSEC			–	FCFGCMD		FCFGEN	7.1.13
(0xCF)	FCFGCID	Chip ID register								7.1.13
(0xCE)	Reserved	–	–	–	–	–	–	–	–	UDR1
(0xCD)	Reserved	–	–	–	–	–	–	–	–	UBBR1H
(0xCC)	Reserved	–	–	–	–	–	–	–	–	UBBR1L
(0xCB)	Reserved	–	–	–	–	–	–	–	–	UCSR1D
(0xCA)	Reserved	–	–	–	–	–	–	–	–	UCSR1C
(0xC9)	Reserved	–	–	–	–	–	–	–	–	UCSR1B
(0xC8)	Reserved	–	–	–	–	–	–	–	–	UCSR1A
(0xC7)	Reserved	–	–	–	–	–	–	–	–	
(0xC6)	UDR0	USART I/O Data Register								
(0xC5)	UBRR0H	–	–	–	–	USART Baud Rate Register High				
(0xC4)	UBRR0L	USART Baud Rate Register Low								
(0xC3)	Reserved	–	–	–	–	–	–	–	–	UCSR0D
(0xC2)	UCSR0C	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01/ UDORD0	UCSZ00/ UCPHA0	UCPOL0	
(0xC1)	UCSR0B	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80	
(0xC0)	UCSR0A	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0	
(0xBF)	Reserved	–	–	–	–	–	–	–	–	
(0xBE)	Reserved	–	–	–	–	–	–	–	–	
(0xBD)	TWAMR	TWAM6	TWAM5	TWAM4	TWAM3	TWAM2	TWAM1	TWAM0	–	
(0xBC)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	
(0xBB)	TWDR	2-wire Serial Interface Data Register								
(0xBA)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	
(0xB9)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	
(0xB8)	TWBR	2-wire Serial Interface Bit Rate Register								
(0xB7)	Reserved	–	–	–	–	–	–	–		
(0xB6)	Reserved	–	–	–	–	–	–	–		ASSR
(0xB5)	Reserved	–	–	–	–	–	–	–	–	
(0xB4)	OCR2B	Timer/Counter2 Output Compare Register B								
(0xB3)	OCR2A	Timer/Counter2 Output Compare Register A								
(0xB2)	TCNT2	Timer/Counter2 (8-bit)								
(0xB1)	TCCR2B	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	
(0xB0)	TCCR2A	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	
(0xAF)	PCMSKKD	PCMSKKD7	PCMSKKD6	PCMSKKD5	PCMSKKD4	PCMSKKD3	PCMSKKD2	PCMSKKD1	PCMSKKD0	7.1.4
(0xAE)	PORTKD	PORTKD7	PORTKD6	PORTKD5	PORTKD4	PORTKD3	PORTKD2	PORTKD1	PORTKD0	7.1.4
(0xAD)	DDRKD	DDRKD7	DDRKD6	DDRKD5	DDRKD4	DDRKD3	DDRKD2	DDRKD1	DDRKD0	7.1.4
(0xAC)	PINKD	PINKD7	PINKD6	PINKD5	PINKD4	PINKD3	PINKD2	PINKD1	PINKD0	7.1.4
(0xAB)	PCMSKKC	PCMSKKC7	PCMSKKC6	PCMSKKC5	PCMSKKC4	PCMSKKC3	PCMSKKC2	PCMSKKC1	PCMSKKC0	7.1.4
(0xAA)	PORTKC	PORTKC7	PORTKC6	PORTKC5	PORTKC4	PORTKC3	PORTKC2	PORTKC1	PORTKC0	7.1.4
(0xA9)	DDRKC	DDRKC7	DDRKC6	DDRKC5	DDRKC4	DDRKC3	DDRKC2	DDRKC1	DDRKC0	7.1.4
(0xA8)	PINKC	PINKC7	PINKC6	PINKC5	PINKC4	PINKC3	PINKC2	PINKC1	PINKC0	7.1.4
(0xA7)	PCMSKKB	PCMSKKB7	PCMSKKB6	PCMSKKB5	PCMSKKB4	PCMSKKB3	PCMSKKB2	PCMSKKB1	PCMSKKB0	7.1.4
(0xA6)	PORTKB	PORTKB7	PORTKB6	PORTKB5	PORTKB4	PORTKB3	PORTKB2	PORTKB1	PORTKB0	7.1.4
(0xA5)	DDRKB	DDRKB7	DDRKB6	DDRKB5	DDRKB4	DDRKB3	DDRKB2	DDRKB1	DDRKB0	7.1.4
(0xA4)	PINKB	PINKB7	PINKB6	PINKB5	PINKB4	PINKB3	PINKB2	PINKB1	PINKB0	7.1.4
(0xA3)	PCMSKKA	PCMSKKA7	PCMSKKA6	PCMSKKA5	PCMSKKA4	PCMSKKA3	PCMSKKA2	PCMSKKA1	PCMSKKA0	7.1.4
(0xA2)	PORTKA	PORTKA7	PORTKA6	PORTKA5	PORTKA4	PORTKA3	PORTKA2	PORTKA1	PORTKA0	7.1.4
(0xA1)	DDRKA	DDRKA7	DDRKA6	DDRKA5	DDRKA4	DDRKA3	DDRKA2	DDRKA1	DDRKA0	7.1.4
(0xA0)	PINKA	PINKA7	PINKA6	PINKA5	PINKA4	PINKA3	PINKA2	PINKA1	PINKA0	7.1.4

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
(0x9F)	PCMSKJD	PCMSKJD7	PCMSKJD6	PCMSKJD5	PCMSKJD4	PCMSKJD3	PCMSKJD2	PCMSKJD1	PCMSKJD0	7.1.3
(0x9E)	PORTJD	PORTJD7	PORTJD6	PORTJD5	PORTJD4	PORTJD3	PORTJD2	PORTJD1	PORTJD0	7.1.3
(0x9D)	DDRJD	DDRJD7	DDRJD6	DDRJD5	DDRJD4	DDRJD3	DDRJD2	DDRJD1	DDRJD0	7.1.3
(0x9C)	PINJD	PINKJD7	PINKJD6	PINKJD5	PINKJD4	PINKJD3	PINKJD2	PINKJD1	PINKJD0	7.1.3
(0x9B)	PCMSKJC	PCMSKJC7	PCMSKJC6	PCMSKJC5	PCMSKJC4	PCMSKJC3	PCMSKJC2	PCMSKJC1	PCMSKJC0	7.1.3
(0x9A)	PORTJC	PORTJC7	PORTJC6	PORTJC5	PORTJC4	PORTJC3	PORTJC2	PORTJC1	PORTJC0	7.1.3
(0x99)	DDRJC	DDRJC7	DDRJC6	DDRJC5	DDRJC4	DDRJC3	DDRJC2	DDRJC1	DDRJC0	7.1.3
(0x98)	PINJC	PINKJC7	PINKJC6	PINKJC5	PINKJC4	PINKJC3	PINKJC2	PINKJC1	PINKJC0	7.1.3
(0x97)	PCMSKJB	PCMSKJB7	PCMSKJB6	PCMSKJB5	PCMSKJB4	PCMSKJB3	PCMSKJB2	PCMSKJB1	PCMSKJB0	7.1.3
(0x96)	PORTJB	PORTJB7	PORTJB6	PORTJB5	PORTJB4	PORTJB3	PORTJB2	PORTJB1	PORTJB0	7.1.3
(0x95)	DDRJB	DDRJB7	DDRJB6	DDRJB5	DDRJB4	DDRJB3	DDRJB2	DDRJB1	DDRJB0	7.1.3
(0x94)	PINJB	PINJB7	PINJB6	PINJB5	PINJB4	PINJB3	PINJB2	PINJB1	PINJB0	7.1.3
(0x93)	PCMSKJA	PCMSKJA7	PCMSKJA6	PCMSKJA5	PCMSKJA4	PCMSKJA3	PCMSKJA2	PCMSKJA1	PCMSKJA0	7.1.3
(0x92)	PORTJA	PORTJA7	PORTJA6	PORTJA5	PORTJA4	PORTJA3	PORTJA2	PORTJA1	PORTJA0	7.1.3
(0x91)	DDRJA	DDRJA7	DDRJA6	DDRJA5	DDRJA4	DDRJA3	DDRJA2	DDRJA1	DDRJA0	7.1.3
(0x90)	PINJA	PINJA7	PINJA6	PINJA5	PINJA4	PINJA3	PINJA2	PINJA1	PINJA0	7.1.3
(0x8F)	PCMSKPL	–	–	–	–	PCMSKPL3	PCMSKPL2	PCMSKPL1	PCMSKPL0	7.1.5
(0x8E)	PORTPL	–	–	–	–	PORTPL3	PORTPL2	PORTPL1	PORTPL0	7.1.5
(0x8D)	DDRPL	–	–	–	–	DDRPL3	DDRPL2	DDRPL1	DDRPL0	7.1.5
(0x8C)	PINPL	–	–	–	–	PINPL3	PINPL2	PINPL1	PINPL0	7.1.5
(0x8B)	OCR1BH	Timer/Counter1 – Output Compare Register B High Byte								
(0x8A)	OCR1BL	Timer/Counter1 – Output Compare Register B Low Byte								
(0x89)	OCR1AH	Timer/Counter1 – Output Compare Register A High Byte								
(0x88)	OCR1AL	Timer/Counter1 – Output Compare Register A Low Byte								
(0x87)	ICR1H	Timer/Counter1 – Input Capture Register High Byte								
(0x86)	ICR1L	Timer/Counter1 – Input Capture Register Low Byte								
(0x85)	TCNT1H	Timer/Counter1 – Counter Register High Byte								
(0x84)	TCNT1L	Timer/Counter1 – Counter Register Low Byte								
(0x83)	Reserved	–	–	–	–	–	–	–	–	
(0x82)	TCCR1C	FOC1A	FOC1B	–	–	–	–	–	–	
(0x81)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	
(0x80)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	
(0x7F)	Reserved	–	–	–	–	–	–	–	–	DIDR1
(0x7E)	DIDR0	–	–	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	
(0x7D)	XLR8ADCR	AD12EN	–	–	–	–	–	–	–	7.1.11
(0x7C)	ADMUX	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	
(0x7B)	ADCSRB	–	–	MUX5	–	–	ADTS2	ADTS1	ADTS0	ACME
(0x7A)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	
(0x79)	ADCH	ADC Data Register High byte								
(0x78)	ADCL	ADC Data Register Low byte								
(0x77)	Reserved	–	–	–	–	–	–	–	–	
(0x76)	Reserved	–	–	–	–	–	–	–	–	
(0x75)	Reserved	–	–	–	–	–	–	–	–	
(0x74)	Reserved	–	–	–	–	–	–	–	–	
(0x73)	SPCIMSK	–	–	SPCIPL	SPCIPK	SPCIPJ	SPCIPG	SPCIPE	SPCIPA	7.1.10
(0x72)	SPCIFR	–	–	SPCIPL	SPCIPK	SPCIPJ	SPCIPG	SPCIPE	SPCIPA	7.1.10
(0x71)	SPICR	–	–	SPCIPL	SPCIPK	SPCIPJ	SPCIPG	SPCIPE	SPCIPA	7.1.10
(0x70)	TIMSK2	–	–	–	–	–	OCIE2B	OCIE2A	TOIE2	
(0x6F)	TIMSK1	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	
(0x6E)	TIMSK0	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	
(0x6D)	PCMSK2	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	
(0x6C)	PCMSK1	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	
(0x6B)	PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	
(0x6A)	OX8MSK	OX8I7	OX8I6	OX8I5	OX8I4	OX8I3	OX8I2	OX8I1	OX8I0	7.1.9

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
(0x69)	EICRA	–	–	–	–	ISC11	ISC10	ISC01	ISC00	
(0x68)	PCICR	–	–	–	–	–	PCIE2	PCIE1	PCIE0	
(0x67)	OX8IFR	OX8I7	OX8I6	OX8I5	OX8I4	OX8I3	OX8I2	OX8I1	OX8I0	7.1.9
(0x66)	OX8ICR	OX8I7	OX8I6	OX8I5	OX8I4	OX8I3	OX8I2	OX8I1	OX8I0	7.1.9
(0x65)	XACK	-	-	-	-	-	-	XIOX8	XIGPIO	7.1.8
(0x64)	PRR	–	–	–	PRINTOSC	–	–	–	–	
(0x63)	XMSK	-	-	-	-	-	-	XIOX8	XIGPIO	7.1.8
(0x62)	XIFR	-	-	-	-	-	-	XIOX8	XIGPIO	7.1.8
(0x61)	XICR	-	-	-	-	-	-	XIOX8	XIGPIO	7.1.8
(0x60)	WDTCSR	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	
0x3F(0x5F)	SREG	I	T	H	S	V	N	Z	C	
0x3E(0x5E)	SPH	–	–	–	–	SP11	SP10	SP9	SP8	
0x3D(0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	
0x3C(0x5C)	Reserved	–	–	–	–	–	–	–	–	
0x3B(0x5B)	XFR3	XLR8 Function (floating point) 32 bit Result High Byte								7.1.6
0x3A(0x5A)	XFR2	XLR8 Function (floating point) 32 bit Result Byte								7.1.6
0x39(0x59)	XFR1	XLR8 Function (floating point) 32 bit Result Byte								7.1.6
0x38(0x58)	XFR0	XLR8 Function (floating point) 32 bit Result Low Byte								7.1.6
0x37(0x57)	SPMCSR	SPMIE	RWWSB	SIGRD	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	
0x36(0x56)	Reserved	–	–	–	–	–	–	–	–	
0x35(0x55)	Reserved	–	–	–	–	–	–	–	–	
0x34(0x54)	MCUSR	–	–	–	–	WDRF	–	EXTRF	PORF	
0x33(0x53)	PCMSKG	PCMSKG7	PCMSKG6	PCMSKG5	PCMSKG4	PCMSKG3	PCMSKG2	PCMSKG1	PCMSKG0	7.1.2
0x32(0x52)	PCMSKE	PCMSKE7	PCMSKE6	PCMSKE5	PCMSKE4	PCMSKE3	PCMSKE2	PCMSKE1	PCMSKE0	7.1.2
0x31(0x51)	PCMSKA	PCMSKA7	PCMSKA6	PCMSKA5	PCMSKA4	PCMSKA3	PCMSKA2	PCMSKA1	PCMSKA0	7.1.2
0x30(0x50)	Reserved	–	–	–	–	–	–	–	–	ACSR
0x2F(0x4F)	Reserved	–	–	–	–	–	–	–	–	ACSRB
0x2E(0x4E)	SPDR	SPI Data Register								
0x2D(0x4D)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	
0x2C(0x4C)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	
0x2B(0x4B)	GPIOR2	General Purpose I/O Register 2								
0x2A(0x4A)	GPIOR1	General Purpose I/O Register 1								
0x29(0x49)	CLKSPD	Clock speed programming used by XLR8 bootloader								7.1.7
0x28(0x48)	OCR0B	Timer/Counter0 Output Compare Register B								
0x27(0x47)	OCR0A	Timer/Counter0 Output Compare Register A								
0x26(0x46)	TCNT0	Timer/Counter0 (8-bit)								
0x25(0x45)	TCCR0B	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	
0x24(0x44)	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	
0x23(0x43)	GTCCR	TSM	–	–	–	–	–	PSRASY	PSRSYNC	
0x22(0x42)	EEARH	EEPROM Address Register High Byte								
0x21(0x41)	EEARL	EEPROM Address Register Low Byte								
0x20(0x40)	EEDR	EEPROM Data Register								
0x1F(0x3F)	EECR	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	
0x1E(0x3E)	GPIOR0	General Purpose I/O Register 0								
0x1D(0x3D)	EIMSK	–	–	–	–	–	–	INT1	INT0	
0x1C(0x3C)	EIFR	–	–	–	–	–	–	INTF1	INTF0	
0x1B(0x3B)	PCIFR	–	–	–	–	–	PCIF2	PCIF1	PCIF0	
0x1A(0x3A)	Reserved	–	–	–	–	–	–	–	–	
0x19(0x39)	Reserved	–	–	–	–	–	–	–	–	TIFR4
0x18(0x38)	Reserved	–	–	–	–	–	–	–	–	TIFR3
0x17(0x37)	TIFR2	–	–	–	–	–	OCF2B	OCF2A	TOV2	
0x16(0x36)	TIFR1	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	
0x15(0x35)	TIFR0	–	–	–	–	–	OCF0B	OCF0A	TOV0	
0x14(0x34)	PORTG	PORTG7	PORTG6	PORTG5	PORTG4	PORTG3	PORTG2	PORTG1	PORTG0	7.1.2

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
0x13(0x33)	DDRG	DDG7	DDG6	DDG5	DDG4	DDG3	DDG2	DDG1	DDG0	7.1.2
0x12(0x32)	PING	PING7	PING6	PING5	PING4	PING3	PING2	PING1	PING0	7.1.2
0x11(0x31)	XFSTAT	XFDONE	XFERR	–	–	–	–	–	–	7.1.6
0x10(0x30)	XFCTRL	–	XFSTART	–	–	–	XFCMD			7.1.6
0x0F(0x2F)	Reserved	–	–	–	–	–	–	–	–	
0x0E(0x2E)	PORTE	–	–	PORTE5	PORTE4	PORTE3	PORTE2	PORTE1	PORTE0	7.1.2
0x0D(0x2D)	DDRE	–	–	DDE5	DDE4	DDE3	DDE2	DDE1	DDE0	7.1.2
0x0C(0x2C)	PINE	–	–	PINE5	PINE4	PINE3	PINE2	PINE1	PINE0	7.1.2
0x0B(0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	
0x0A(0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	
0x09(0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	
0x08(0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	
0x07(0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	
0x06(0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	
0x05(0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	
0x04(0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	
0x03(0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	
0x02(0x22)	PORTA	–	–	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	7.1.2
0x01(0x21)	DDRA	–	–	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	7.1.2
0x0(0x20)	PINA	–	–	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	7.1.2
		= unchanged from ATmega328p								
		= Reserved registers that are best not used for XLR8 blocks because ATmega328PB uses them								
		= ATmega328p registers not implemented in XLR8								
		= Some differences in XLR8 compared to ATmega328p								
		= new registers for XLR8 Blocks								
		= Built-in XB registers. Not reserved in OpenXLR8 and can be used for OpenXLR8 registers								
		= Sno Edge specific registers								

Figure 18: Sno Edge Register Summary

7.1 Sno Edge and XB Register Descriptions

7.1.1 Register Access Definitions

In Figure 19: Register Access Definitions, the abbreviations used in the following CSR definitions are defined.

Abbreviation	Meaning
RW	Read and Write Access
R	Read Only
W	Write Only
RW1C	Read and Write, Write 1 to Clear
RW1CS	Read and Write, Write 1 to Clear, Sticky
RWS	Read and Write, Sticky

Figure 19: Register Access Definitions

Sticky bits are not initialized or modified by hot reset or function level reset.

7.1.2 Ports A, E and G

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
0x31(0x51)	PCMSKA	PCMSKA7	PCMSKA6	PCMSKA5	PCMSKA4	PCMSKA3	PCMSKA2	PCMSKA1	PCMSKA0	
Read/Write		N/A	N/A	RW	RW	RW	RW	RW	RW	
Initial Value		N/A	N/A	0	0	0	0	0	0	
0x02(0x22)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	
Read/Write		N/A	N/A	RW	RW	RW	RW	RW	RW	
Initial Value		N/A	N/A	0	0	0	0	0	0	
0x01(0x21)	DDRA	DDRA7	DDRA6	DDRA5	DDRA4	DDRA3	DDRA2	DDRA1	DDRA0	
Read/Write		N/A	N/A	RW	RW	RW	RW	RW	RW	
Initial Value		N/A	N/A	0	0	0	0	0	0	
0x0(0x20)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	
Read/Write		N/A	N/A	RW	RW	RW	RW	RW	RW	
Initial Value		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

0x32(0x52)	PCMSKE	PCMSKE7	PCMSKE6	PCMSKE5	PCMSKE4	PCMSKE3	PCMSKE2	PCMSKE1	PCMSKE0	
Read/Write		N/A	N/A	RW	RW	RW	RW	RW	RW	
Initial Value		N/A	N/A	0	0	0	0	0	0	
0x0E(0x2E)	PORTE	PORTE7	PORTE6	PORTE5	PORTE4	PORTE3	PORTE2	PORTE1	PORTE0	
Read/Write		N/A	N/A	RW	RW	RW	RW	RW	RW	
Initial Value		N/A	N/A	0	0	0	0	0	0	
0x0D(0x2D)	DDRE	DDRE7	DDRE6	DDRE5	DDRE4	DDRE3	DDRE2	DDRE1	DDRE0	
Read/Write		N/A	N/A	RW	RW	RW	RW	RW	RW	
Initial Value		N/A	N/A	0	0	0	0	0	0	
0x0C(0x2C)	PINE	PINE7	PINE6	PINE5	PINE4	PINE3	PINE2	PINE1	PINE0	
Read/Write		N/A	N/A	RW	RW	RW	RW	RW	RW	
Initial Value		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

0x33(0x53)	PCMSKG	PCMSKG7	PCMSKG6	PCMSKG5	PCMSKG4	PCMSKG3	PCMSKG	PCMSKG	PCMSKG0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x14(0x34)	PORTG	PORTG7	PORTG6	PORTG5	PORTG4	PORTG3	PORTG	PORTG	PORTG0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x13(0x33)	DDRG	DDRG7	DDRG6	DDRG5	DDRG4	DDRG3	DDRG	DDRG	DDRG0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x12(0x32)	PING	PING7	PING6	PING5	PING4	PING3	PING	PING	PING0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

7.1.3 Ports JA, JB, JC, and JD

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
0x93	PCMSKJA	PCMSKJA7	PCMSKJA6	PCMSKJA5	PCMSKJA4	PCMSKJA3	PCMSKJA2	PCMSKJA1	PCMSKJA0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
Initial Value		0	0	0	0	0	0	0	0	
0x92	PORTJA	PORTJA7	PORTJA6	PORTJA5	PORTJA4	PORTJA3	PORTJA2	PORTJA1	PORTJA0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x81	DDRJA	DDRJA7	DDRJA6	DDRJA5	DDRJA4	DDRJA3	DDRJA2	DDRJA1	DDRJA0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x90	PINJA	PINJA7	PINJA6	PINJA5	PINJA4	PINJA3	PINJA2	PINJA1	PINJA0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

0x97	PCMSKJB	PCMSKJB7	PCMSKJB6	PCMSKJB5	PCMSKJB4	PCMSKJB3	PCMSKJB2	PCMSKJB1	PCMSKJB0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x96	PORTJB	PORTJB7	PORTJB6	PORTJB5	PORTJB4	PORTJB3	PORTJB2	PORTJB1	PORTJB0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x95	DDRJB	DDRJB7	DDRJB6	DDRJB5	DDRJB4	DDRJB3	DDRJB2	DDRJB1	DDRJB0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x94	PINJB	PINJB7	PINJB6	PINJB5	PINJB4	PINJB3	PINJB2	PINJB1	PINJB0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

0x9B	PCMSKJC	PCMSKJC7	PCMSKJC6	PCMSKJC5	PCMSKJC4	PCMSKJC3	PCMSKJC2	PCMSKJC1	PCMSKJC0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x9A	PORTJC	PORTJC7	PORTJC6	PORTJC5	PORTJC4	PORTJC3	PORTJC2	PORTJC1	PORTJC0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x99	DDRJC	DDRJC7	DDRJC6	DDRJC5	DDRJC4	DDRJC3	DDRJC2	DDRJC1	DDRJC0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x98	PINJC	PINJC7	PINJC6	PINJC5	PINJC4	PINJC3	PINJC2	PINJC1	PINJC0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

0x9F	PCMSKJD	PCMSKJD7	PCMSKJD	PCMSKJD	PCMSKJD4	PCMSKJD3	PCMSKJD2	PCMSKJD1	PCMSKJD0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x9E	PORTJD	PORTJD7	PORTJD	PORTJD	PORTJD4	PORTJD3	PORTJD2	PORTJD1	PORTJD0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x9D	DDRJD	DDRJD7	DDRJD	DDRJD	DDRJD4	DDRJD3	DDRJD2	DDRJD1	DDRJD0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x9C	PINJD	PINJD7	PINJD	PINJD	PINJD4	PINJD3	PINJD2	PINJD1	PINJD0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

7.1.4 Ports KA, KB, KC, and KD

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
0x93	PCMSKKA	PCMSKKA7	PCMSKKA6	PCMSKKA5	PCMSKKA4	PCMSKKA3	PCMSKKA2	PCMSKKA1	PCMSKKA0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x92	PORTKA	PORTKA7	PORTKA6	PORTKA5	PORTKA4	PORTKA3	PORTKA2	PORTKA1	PORTKA0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x81	DDRKA	DDRKA7	DDRKA6	DDRKA5	DDRKA4	DDRKA3	DDRKA2	DDRKA1	DDRKA0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x90	PINKA	PINKA7	PINKA6	PINKA5	PINKA4	PINKA3	PINKA2	PINKA1	PINKA0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

0x97	PCMSKKB	PCMSKKB7	PCMSKKB6	PCMSKKB5	PCMSKKB4	PCMSKKB3	PCMSKKB2	PCMSKKB1	PCMSKKB0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x96	PORTKB	PORTKB7	PORTKB6	PORTKB5	PORTKB4	PORTKB3	PORTKB2	PORTKB1	PORTKB0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x95	DDRKB	DDRKB7	DDRKB6	DDRKB5	DDRKB4	DDRKB3	DDRKB2	DDRKB1	DDRKB0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x94	PINKB	PINKB7	PINKB6	PINKB5	PINKB4	PINKB3	PINKB2	PINKB1	PINKB0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

0x9B	PCMSKKC	PCMSKKC7	PCMSKKC6	PCMSKKC5	PCMSKKC4	PCMSKKC3	PCMSKKC2	PCMSKKC1	PCMSKKC0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x9A	PORTKC	PORTKC7	PORTKC6	PORTKC5	PORTKC4	PORTKC3	PORTKC2	PORTKC1	PORTKC0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x99	DDRKC	DDRKC7	DDRKC6	DDRKC5	DDRKC4	DDRKC3	DDRKC2	DDRKC1	DDRKC0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x98	PINKC	PINKC7	PINKC6	PINKC5	PINKC4	PINKC3	PINKC2	PINKC1	PINKC0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

0x9F	PCMSKKD	PCMSKKD7	PCMSKKD6	PCMSKKD5	PCMSKKD4	PCMSKKD3	PCMSKKD2	PCMSKKD1	PCMSKKD0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x9E	PORTKD	PORTKD7	PORTKD6	PORTKD5	PORTKD4	PORTKD3	PORTKD2	PORTKD1	PORTKD0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x9D	DDRKD	DDRKD7	DDRKD6	DDRKD5	DDRKD4	DDRKD3	DDRKD2	DDRKD1	DDRKD0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x9C	PINKD	PINKD7	PINKD6	PINKD5	PINKD4	PINKD3	PINKD2	PINKD1	PINKD0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	

Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

7.1.5 Port PL

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
0x8F	PCMSKPL	PCMSKPL7	PCMSKPL6	PCMSKPL5	PCMSKPL4	PCMSKPL3	PCMSKPL2	PCMSKPL1	PCMSKPL0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x8E	PORTPL	PORTPL7	PORTPL6	PORTPL5	PORTPL4	PORTPL3	PORTPL2	PORTPL1	PORTPL0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x8D	DDRPL	DDRPL7	DDRPL6	DDRPL5	DDRPL4	DDRPL3	DDRPL2	DDRPL1	DDRPL0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
0x8C	PINPL	PINPL7	PINPL6	PINPL5	PINPL4	PINPL3	PINPL2	PINPL1	PINPL0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

7.1.6 XFCTRL, XFSTAT, XFR0, XFR1, XFR2, XFR3– Floating Point XB Registers

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
0x3B(0x5B)	XFR3	XLR8 Function (floating point) 32 bit Result High Byte								
0x3A(0x5A)	XFR2	XLR8 Function (floating point) 32 bit Result Byte								
0x39(0x59)	XFR1	XLR8 Function (floating point) 32 bit Result Byte								
0x38(0x58)	XFR0	XLR8 Function (floating point) 32 bit Result Low Byte								
Read/Write		R	R	R	R	R	R	R	R	
Initial Value		0	0	0	0	0	0	0	0	
0x11(0x31)	XFSTAT	XFDONE	XFERR	–	–	–	–	–	–	
Read/Write		R	R	R	R	R	R	R	R	
Initial Value		0	0	0	0	0	0	0	0	
0x10(0x30)	XFCTRL	–	XFSTART	–	–	–	XFCMD			
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	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 sets, 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).

7.1.7 CLKSPD – Clock Speed Register

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
0x29(0x49)	CLKSPD	Clock speed programming used by XLR8 bootloader								
Read/Write		R	R	R	R	R	R	R	R	
Initial Value		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	
0x29(0x49)	CLKSPD	–	–	–	–	–	–	–	OSCOUT	

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
Read/Write		W	W	W	W	W	W	W	W	
Initial Value		N/A	N/A	N/A	N/A	N/A	N/A	N/A	0	
(0x64)	PRR	-	–	–	–	PRINTOSC	–	–	–	
Read/Write		R	R	R	R	RW	R	R	R	
Initial Value		0	0	0	0	0	0	0	0	

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 to allow it to run correctly regardless of whether Sno Edge is running 16MHZ, 32MHz, or some other speed.

Sno Edge 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. The other bits of this register are currently unused.

7.1.8 XICR, XIFR, XMSK, XACK – Extended IRQ

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
(0x65)	XACK	-						XIOX8	XIGPIO	
Read/Write		R	R	R	R	R	R	RW1C	RW1C	
Initial Value		0	0	0	0	0	0	0	0	
(0x63)	XMSK	-						XIOX8	XIGPIO	
Read/Write		R	R	R	R	R	R	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
(0x62)	XIFR	-						XIOX8	XIGPIO	
Read/Write		R	R	R	R	R	R	RW1C	RW1C	
Initial Value		0	0	0	0	0	0	0	0	
(0x61)	XICR	-						XIOX8	XIGPIO	
Read/Write		R	R	R	R	R	R	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	

A bit in the Flag register (XIFR) will be set when an IRQ is received if the corresponding bit in the Mask register (XMSK) is set and the corresponding bit in the Acknowledge (XACK) register is not set.

A bit in the Flag register is cleared either by the corresponding bit in the Acknowledge register being set, or by the source of the IRQ being cleared.

A bit set in the Flag register will cause an IRQ to be generated if the corresponding bit in the Control register (XICR) is set.

When an IRQ is generated to the AVR core it will respond by setting a bit in the acknowledge register. This will block the corresponding bit in the Flag register from being set, preventing further IRQs of that type from being sent to the AVR core. The bit in the Acknowledge must be cleared by software once the interrupt has been serviced and control is returned to the original program. Bits in the Acknowledge register can be cleared by writing a one to the corresponding bit location in the Acknowledge register

7.1.9 OX8ICR, OX8IFR, OX8MSK – OpenXLR8 Interrupts

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
(0x6A)	OX8MSK	OX8I7	OX8I6	OX8I5	OX8I4	OX8I3	OX8I2	OX8I1	OX8I0	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
(0x67)	OX8IFR	OX8I7	OX8I6	OX8I5	OX8I4	OX8I3	OX8I2	OX8I1	OX8I0	
Read/Write		RW1C	RW1C	RW1C	RW1C	RW1C	RW1C	RW1C	RW1C	
Initial Value		0	0	0	0	0	0	0	0	
(0x66)	OX8ICR	OX8I7	OX8I6	OX8I5	OX8I4	OX8I3	OX8I2	OX8I1	OX8I0	
Read/Write		RW	RW	RW	RW	RW	RW	RW1C	RW1C	
Initial Value		0	0	0	0	0	0	0	0	

A bit in the Flag register (OX8IFR) will be set when a pin change notification is received if the corresponding bit in the Mask register (OX8MSK) is set.

A bit in the Flag register is cleared via software by writing a one to the bit.

A bit set in the Flag register will cause an IRQ to be generated if the corresponding bit in the Control register (OX8ICR) is set.

7.1.10 SPICR, SPIFR, SPIMSK – Sno Pin Change Interrupts

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
(0x73)	SPIMSK	-	-	SPCIPL	SPCIPK	SPCIPJ	SPCIPG	SPCIPE	SPCIPA	
Read/Write		R	R	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
(0x72)	SPIFR	-	-	SPCIPL	SPCIPK	SPCIPJ	SPCIPG	SPCIPE	SPCIPA	
Read/Write		R	R	RW1C	RW1C	RW1C	RW1C	RW1C	RW1C	
Initial Value		0	0	0	0	0	0	0	0	
(0x71)	SPICR	-	-	SPCIPL	SPCIPK	SPCIPJ	SPCIPG	SPCIPE	SPCIPA	
Read/Write		R	R	RW	RW	RW	RW	RW1C	RW1C	
Initial Value		0	0	0	0	0	0	0	0	

The bits in the above registers correspond to the ports in the following way. Notice that in the figure above and in Figure 14: Sno Pin Change Interrupt Fields, the four Jx ports and the four Kx ports are combined into single bits.

A bit in the Flag register (SPCIFR) will be set when a pin change notification is received if the corresponding bit in the Mask register (SPCIMSK) is set.

A bit in the Flag register is cleared via software by writing a one to the bit.

A bit set in the Flag register will cause an IRQ to be generated if the corresponding bit in the Control register (SPCICR) is set.

Neither the Mask register nor the Control register support bit operations, so a read-modify-write operation should be used to change individual bits.

7.1.11 XLR8ADCR – Sno Edge ADC Control Register

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
(0x7D)	XLR8ADCR	AD12EN	–	–	–	–	–	–	–	
Read/Write		RW	R	R	R	R	R	R	R	
Initial Value		0	0	0	0	0	0	0	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 result is truncated (not rounded) from the 12 bit result.

7.1.12 FCFGCID – Chip ID Register

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
(0xCF)	FCFGCID	Chip ID register								
Read/Write		R	R	R	R	R	R	R	R	write-reset
Initial Value		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

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).

7.1.13 FCFGDAT, FCFGSTS, FCFGCTL – FPGA Reconfiguration Registers

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
(0xD2)	FCFGDAT	FPGA Reconfiguration Data Register								
Read/Write		W	W	W	W	W	W	W	W	
Initial Value		0	0	0	0	0	0	0	0	
(0xD1)	FCFGSTS	FCFGDN	FCFGOK	FCFGFAIL	FCFGRDY	–	–	–	–	
Read/Write		RW1C	RW1C	RW1C	R	R	R	R	R	
Initial Value		0	0	0	0	0	0	0	0	
(0xD0)	FCFGCTL	–	FCFGSEC			–	FCFGCMD		FCFGEN	
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	

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.

7.1.14 XLR8VERL, XLR8VERH, XLR8VERT – Version Number Registers

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
(0xD6)	XLR8VERT	XLR8 Version Number Flags								
(0xD5)	XLR8VERH	XLR8 Version Number Register High Byte								
(0xD4)	XLR8VERL	XLR8 Version Number Register Low Byte								
Read/Write		R	R	R	R	R	R	R	R	
Initial Value		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

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).

7.1.15 XLR8Quad – XLR8 Quadrature

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
(0xE9)	QERAT3	Upper 8 bits of quadrature rate data								
Read/Write		R	R	R	R	R	R	R	R	
Initial Value		0	0	0	0	0	0	0	0	
(0xE8)	Q ERAT2	Upper-middle 8 bits of quadrature rate data								
Read/Write		R	R	R	R	R	R	R	R	
Initial Value		0	0	0	0	0	0	0	0	
(0xE7)	QERAT1	Lower-middle 8 bits of quadrature rate data								
Read/Write		R	R	R	R	R	R	R	R	
Initial Value		0	0	0	0	0	0	0	0	
(0xE6)	QERAT0	Lower 8 bits of quadrature rate data								
Read/Write		R	R	R	R	R	R	R	R	
Initial Value		0	0	0	0	0	0	0	0	
(0xE5)	QECNT3	Upper 8 bits of quadrature count data								
Read/Write		R	R	R	R	R	R	R	R	
Initial Value		0	0	0	0	0	0	0	0	
(0xE4)	QECNT2	Upper-middle 8 bits of quadrature count data								
Read/Write		R	R	R	R	R	R	R	R	
Initial Value		0	0	0	0	0	0	0	0	
(0xE3)	QECNT1	Lower-middle 8 bits of quadrature count data								
Read/Write		R	R	R	R	R	R	R	R	
Initial Value		0	0	0	0	0	0	0	0	
(0xE2)	QECNT0	Lower 8 bits of quadrature count data								
Read/Write		R	R	R	R	R	R	R	R	
Initial Value		0	0	0	0	0	0	0	0	
(0xE0)	QECR	QEEN	QEDIS	QECLR	QERATE	QECHAN				
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	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.

7.1.16 XLR8PID – XLR8 PID

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
(0xF6)	PID_OP_L	Low Byte output								
Read/Write		R	R	R	R	R	R	R	R	
Initial Value		0	0	0	0	0	0	0	0	
(0xF5)	PID_OP_H	High Byte output								
Read/Write		R	R	R	R	R	R	R	R	
Initial Value		0	0	0	0	0	0	0	0	
(0xF4)	PID_PV_L	Process variable low byte								
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
(0xF3)	PID_PV_H	Process variable high byte								
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
(0xF2)	PID_SP_L	Set point low byte								

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
(0xF1)	PID_SP_H	Set point high byte								
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
(0xF0)	PID_KP_L	KP coefficient low byte								
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
(0xEF)	PID_KP_H	KP coefficient high byte								
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
(0xEE)	PID_KI_L	KI coefficient low byte								
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
(0xED)	PID_KI_H	KI coefficient high byte								
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
(0xEC)	PID_KD_L	KD coefficient low byte								
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
(0xEB)	PID_KD_H	KD coefficient high byte								
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
(0xEA)	PIDCR	PEDEN	PIDDIS	PIDUPD	PIDCHAN					
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	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.

7.1.17 SVPWH, SVPWL, SVCR – Servo XB Registers

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Notes
(0xFD)	SVPWH	–	–	–	–	Servo Pulse Width High Register				
Read/Write		R	R	R	R	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
(0xFC)	SVPWL	Servo Pulse Width Low Register								
Read/Write		RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	
(0xFB)	SVCR	SVEN	SVDIS	SVUP	SVCHAN					
Read/Write		RW	W	W	RW	RW	RW	RW	RW	
Initial Value		0	0	0	0	0	0	0	0	

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).

7.2 Using the Sno Edge Registers in Software

The Sno Edge registers can be accessed from Arduino sketches in much the same way as standard Arduino registers. The Sno Edge has been defined as a "variant" in the Arduino IDE so its registers are available to the Arduino compiler. Simply use the register names and register field names as defined in Figure 18. No #include statements required to pull in the register definitions. Just select the Sno Edge board in the Arduino IDE under Tools->Board.

The register names are defined using the _SFR_MEM8(), such that using the name causes the register to be read or written, depending on the context.

The field names are defined as a number between 0 and 7, to indicate which bit in the register the field starts at. For multiple bit fields this will indicate the low order bit of the field. This makes it simple to use left shift operators to specify bit positions.

The values of registers can be read by specifying their names on the right side of an equal sign.

```
// Read value of the XICR register into the "var1" variable
var1 = XICR;
```

To write registers use the name of the register on the left side of the equal sign.

```
// Write the value of the "var2" variable to the XICR register
XICR = var2;
```

To use a field name to specify a bit location, use the left shift operator.

```
// Shift a 1 to the bit position of the XIOX8 field in the XICR register
// and write it to the XICR register. All other bits in the register will
// be set to 0.
XICR = (1 << XIOX8);
```

To preserve the other bits in the register from changes while setting/clearing a specific bit, use the compound assignment operators |= or &=.

```
// Set the bit at the XIOX8 location in the XICR register while preserving
// the state of all other bits.
XICR |= (1 << XIOX8);
```

```
// Clear the bit at the XIOX8 location in the XICR register while preserving
// the state of all other bits.
XICR &= ~(1 << XIOX8);
```

To set multiple bits in a register, multiple left shift operations can be bit-wise OR'd together.

```
// Set both the XIOX8 and the XIGPIO bits in the XICR register.
XICR = (1 << XIOX8) | (1 << XIGPIO);
```

Multibit fields can be left shifted the same as single bit fields since the field name is set to the lowest order bit in the field. Care must be taken that field values are within the max value of the field or run the risk of fields overlapping during the shift operations.

```
// Shift the values for the fields of the FCFGCTL register to the correct
// offsets in the register and write to the FCFGCTL register. Though these
// are multi-bit fields, the field definitions are set to shift the values
// to the correct location.
FCFGCTL = (var3 << FCFGSEC) | (var4 << FCFGCMD) | (var5 << FCFGGEN);
```

Another way to make accessing the fields of the registers is to create struct types to define them and then access the subfields. Using the FCFGCTL register as an example:

```
// Define a struct type for the FCFGCTL register
typedef struct {
  unsigned int fcfgen  : 1; // [  0] - Enable
  unsigned int fcfgcmd : 2; // [2:1] - Command
  unsigned int rsrv3   : 1; // [  3] - unused
  unsigned int fcfgsec : 3; // [6:4] = Sec
  unsigned int rsrv7   : 1; // [  7] - unused
} fcfgctl_t;

fcfgctl_t fcfgctl;      // Create fcfgctl as a struct of type fcfgctl_t

// Read register fields
fcfgctl = FCFGCTL;      // Read the FCFGCTL reg into the struct
i = fcfgctl.fcfgsec;    // set i to the value of the fcfgsec field
```

```
// Write register fields
fcfgctl.fcfgcmd = 0x2;  // Set the value of a field
FCFGCTL = fcfgctl;      // Write the struct to the register
```

8 Schematics and Other Resources

Schematics, Pin Map, and Product Brief Schematics, product brief, and a standalone pin map document are available on the resources page for Sno Edge here:

- Product Brief
- Pin Map
- Schematics

9 Credits

Some code is used and modified from the AVR core written by Ruslan Lepetenok (lepetenokr@yahoo.com) that is available at http://opencores.org/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:

10 Appendix A – Arduino IDE Installation and Running Test Program

10.1 Installing Arduino IDE

The first step in setting up your computer to connect to and program the Sno Edge is to install the standard Arduino IDE software. Follow the instructions below to install the Arduino IDE on your computer.

10.1.1 Microsoft Windows

1. Click here for the official Arduino IDE installation guide for Microsoft Windows.
2. Follow the instructions for installing the IDE.
3. Once the IDE is installed, return here to finish installation of the Alorium Technology board specific packages and libraries.

10.1.2 Mac OS X

1. Click here for the official Arduino IDE installation guide for Mac OS X.
2. Follow the instructions for installing the IDE.
3. Once the IDE is installed, return here to finish installation of the Alorium Technology board specific packages and libraries.

10.1.3 Linux

If you are running Linux, the setup steps are a bit different. Therefore, we have created one tutorial that incorporates all of the steps Linux requires to setup Arduino IDE.

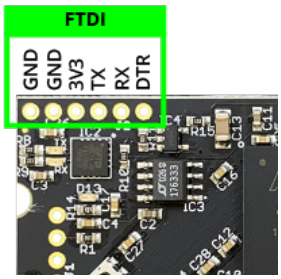
This document was originally created when we released our XLR8 board, and it still carries the XLR8 name in the title. However, the steps remain the same for using Arduino with Sno Edge, as well.

Click the link below to see our Linux Setup Tutorial:

- Linux Setup Tutorial

10.2 FTDI Driver Installation

Sno Edge can be programmed with the Arduino IDE across an FTDI interface located at the top edge of the board.



A USB-to-FTDI adapter of some kind will be required to connect your computer to Snō for programming with Arduino. There are a variety of cables and solutions available on the market. One of our favorites is the SparkFun Beefy 3 Basic FTDI Breakout.

In order to communicate with the FTDI breakout board, drivers for the FTDI chip may need to be installed. A great set of instructions for installing the driver can be found here:

- SparkFun FTDI Installation Guide

The SparkFun guide will tell you if you need to install the driver. You may need to reboot your computer after installation.

A note about FTDI drivers and Mac OS:

If you are running macOS, you may run into issues with the usb serial port disappearing and not reconnecting. There are known issues between the factory installed macOS FTDI drivers and drivers available for installation from FTDI directly. And, unfortunately, the jury still appears to be out on which version of macOS will work consistently without ever seeing the lost serial port problem.

The following video on our YouTube channel provides the steps for a potential fix to this Mac related issue that has worked for several of us at Alorium Technology since the summer of 2017. It's no iron-clad guarantee, but it seems to have solved the problem so far.

How to Fix FTDI Driver Issue on Mac and macOS

10.3 Installing Sno Edge Board Package and Libraries

To take advantage of the XBs that come with Sno Edge, you'll need to take the following additional steps.

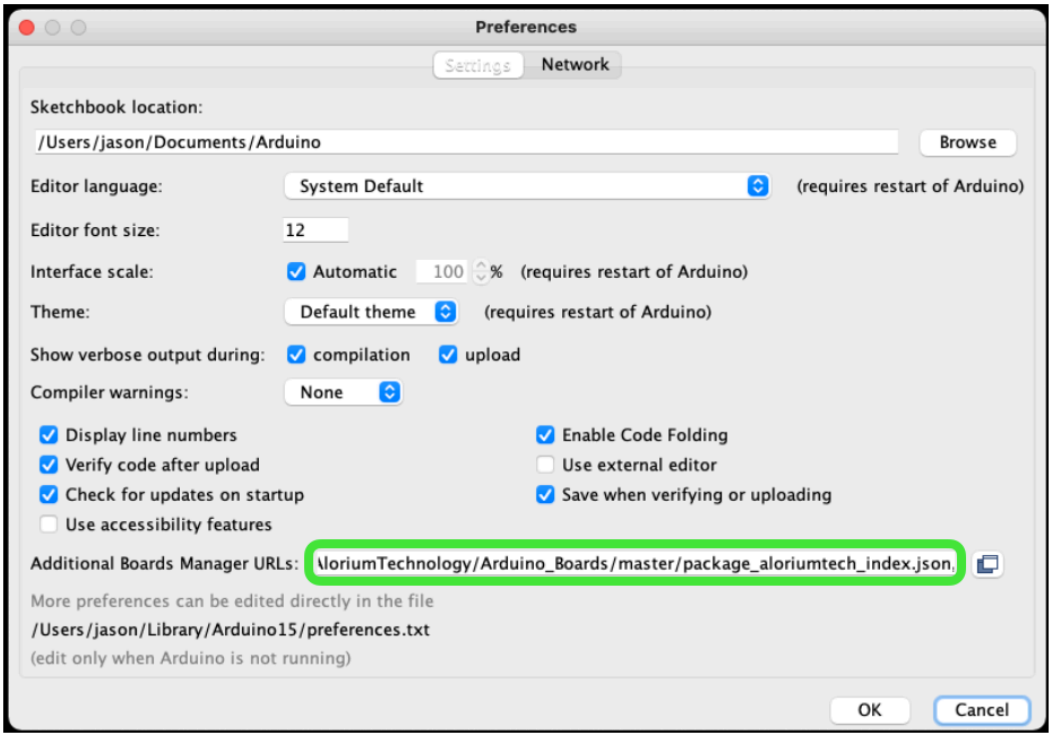
Note: Sno Edge is part of our XLR8 family of boards, and they are all supported with the top-level XLR8 boards package and XLR8 Arduino libraries. So, you will be downloading and installing files that have the XLR8 name.

10.3.1 Add Sno Edge Board Support

Open the Arduino IDE and follow these steps to add board support in the Arduino IDE.

1. For Windows and Linux: Go to **File > Preferences**, in your Arduino IDE menu bar.
2. For Mac: Go to **Arduino** > **Preferences**, in your Arduino IDE menu bar.
3. Locate the 'Additional Boards Manager URLs' input field.
4. Copy and paste this URL into the "Additional Boards Manager URLs" input field

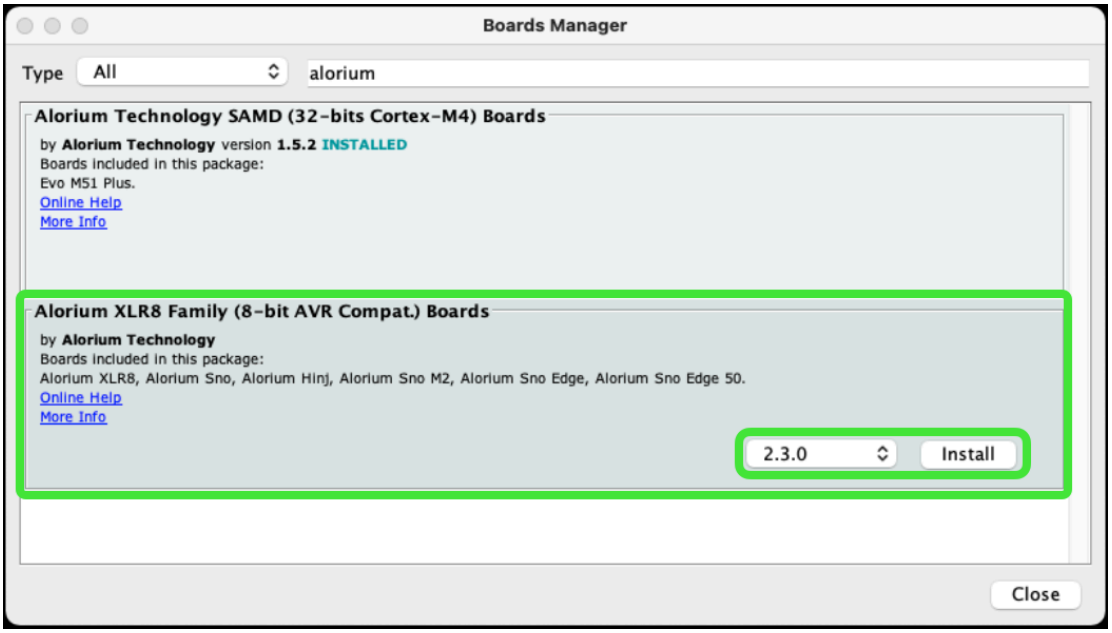
https://raw.githubusercontent.com/AloriumTechnology/Arduino_Boards/master/package_aloriumtech_index.json



Note: Multiple URLs can be added to this field by separating each URL with a comma.

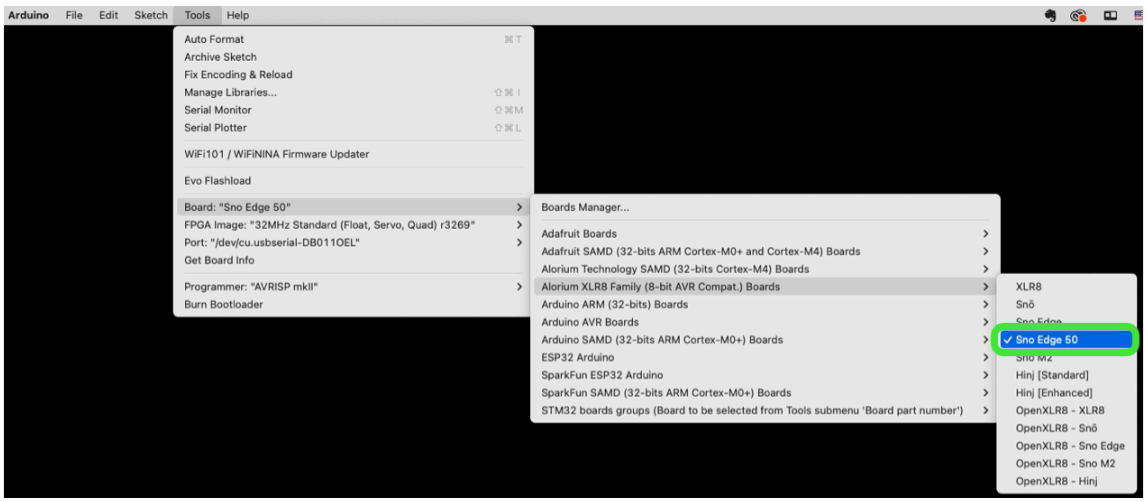
Install Alorium's XLR8 board package

1. Go to **Tools > Board > Boards Manager**.
2. Type "alorium," in the search field and you will see an option to install board files for Alorium XLR8 AVR compatible boards.
3. Select the "**Alorium XLR8 Family (8-bit AVR Compat.) Boards**" package and then click "Install."



Select the Sno Edge Board

1. Go to **Tools > Board**. You should see a new section titled "Alorium XLR8 Family (8-bit AVR Compat.) Boards" now exists.
2. Select **"Sno Edge 50"** board.



After selecting Sno Edge 50, you will find a new menu item at Tools > FPGA Image, where you will see the list of released Sno Edge images that are packaged with the Arduino IDE.

10.3.2 *Sno Edge Libraries*

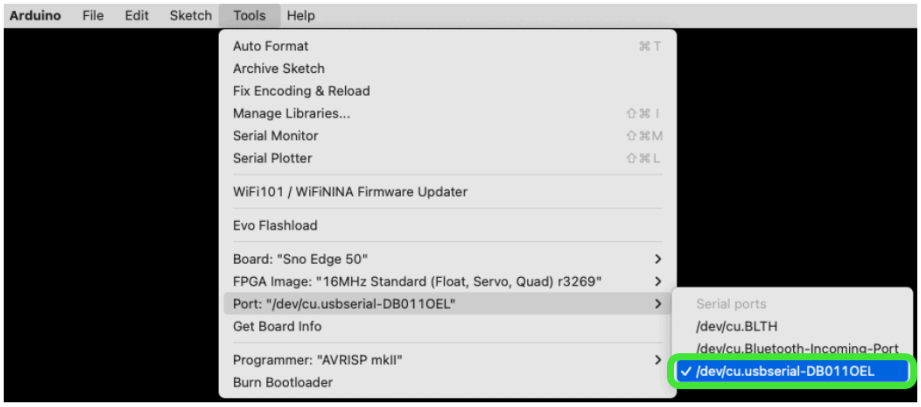
All libraries required to use Sno Edge are packaged with the Alorium Technology XLR8 Arduino board package.

As new functionality or Xcelerator Blocks (XBs) are added for the FPGA, new libraries may be released. Detailed instructions for installing required libraries will be added at that time.

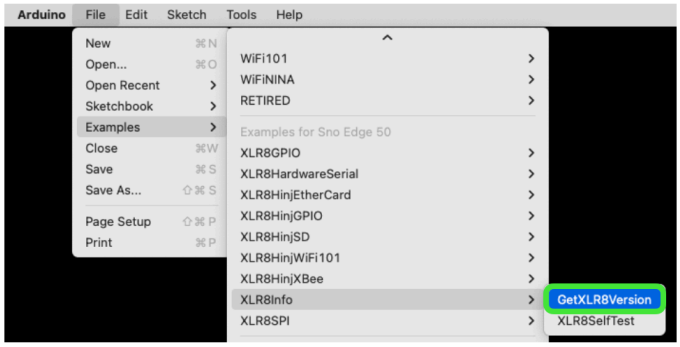
10.4 *Running an Example Sketch/Program*

To be sure that everything is installed and working correctly, we have provided an example Arduino sketch called "GetXLR8Version" that you can load from the Arduino IDE Examples menu.

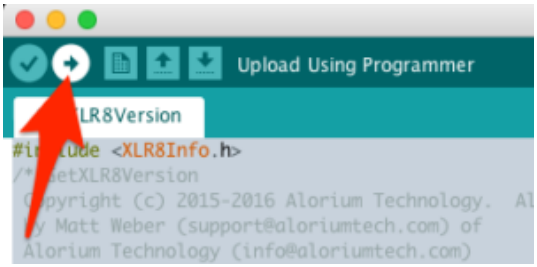
1. Be sure that your Sno Edge board is connected to your computer either with the FTDI interface or to a USB cable on your Sno Edge carrier board.
2. Go to **Tools** > **Port** and verify that Arduino IDE is connected to the Sno Edge serial port. Note that your will likely have a different identifier than what's shown below.



3. Go to **File** > **Examples** > **XLR8Info** and select **GetXLR8Version**



4. In the GetXLR8Version sketch window, click on the Upload button



5. Check the Serial Monitor window for the output, which should look like the output below. **Note that you will need to set the baud rate for the Serial Monitor to 115200 for this sketch to display output correctly.**

```
============================================================
Board Type: Sno Edge 50
FPGA Image: 16 MHz r3253
============================================================
XLR8 Hardware Version = 3253
  Modified working copy
XLR8 CID              = 0xC020960C
------------------------------------------------------------
Design Configuration  = 0x8000C8A
  Image       = 1
  Clock       = 32MHz
  PLL Speed   = 16MHz
  FPGA Size   = M50
------------------------------------------------------------
No Builtin XB Enabled
------------------------------------------------------------
OpenXLR8 Info Regs    = 3
  Info Reg 1 = 0x11
  Info Reg 2 = 0x12
  Info Reg 3 = 0x13
------------------------------------------------------------
------------------------------------------------------------
Int Osc = 55.08 MHz
------------------------------------------------------------
```

If you get this output from GetXLR8Version, that means everything is installed correctly. Congratulations!