Lab 3: Sensor Interfacing
Alex Gall, Noah Terry

Relevant Information

In this lab we implemented functionality to read encoder values and the battery voltage. This can be useful for future projects where reading encoder values is required since it is a common application in robotics. Additionally, it is useful to be able to read the battery voltage in order to prevent damaging the batteries. Learning how to read the battery voltage was also useful since we were required to use PWM and analog to digital conversions. These applications will be useful for projects requiring PWM or reading data from analog sensors. The most impactful learning outcome from this lab was to be careful when setting bits in registers and ensure that the bits are being set as intended. We encountered an issue where the right encoder values were off by a factor of two due to a pin in a register not being set correctly.

Encoder Registers and Pins

For the encoders, four external pins were used, pin B4, pin E2, pin E6, and pin F0. All these pins needed to be set as digital input pins by setting their respective data direction register bits to zero and setting their data/port register bit to 1, see the following registers below.

10.4.3 Port B Data Direction Register – DDRB

Bit	7	6	5	4	3	2	1	0	
	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

10.4.12 Port E Data Direction Register – DDRE

Bit	7	6	5	4	3	2	1	0	
	-	DDE6	-	-	-	DDE2	-	-	DDRE
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

10.4.15 Port F Data Direction Register – DDRF

Bit	7	6	5	4	3	2	1	0	
	DDF7	DDF6	DDF5	DDF4	-	-	DDF1	DDF0	DDRF
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

10.4.2 Port B Data Register – PORTB

Bit	7	6	5	4	3	2	1	0	
	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

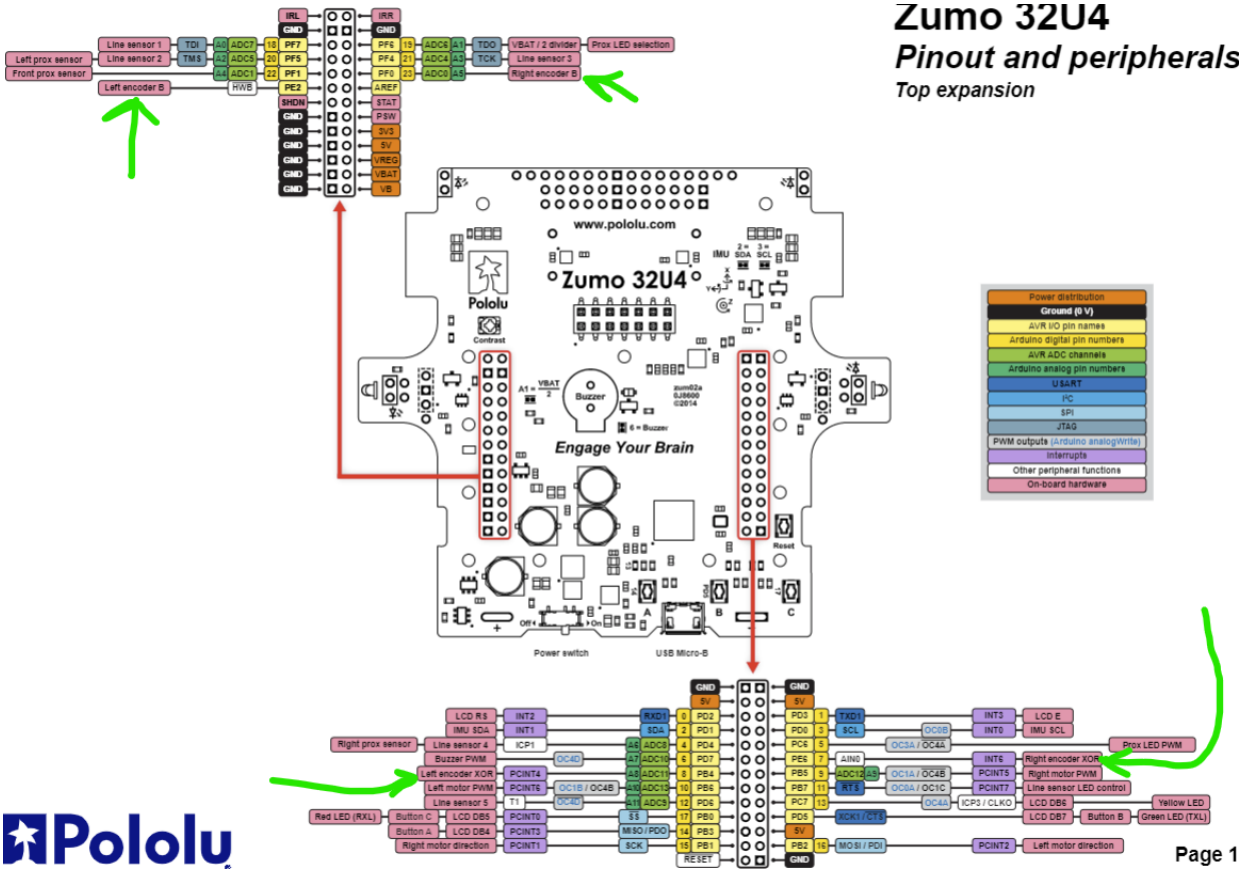
10.4.11 Port E Data Register – PORTE

Bit	7	6	5	4	3	2	1	0	
	-	PORTE6	-	-	-	PORTE2	-	-	PORTE
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

10.4.14 Port F Data Register – PORTF

Bit	7	6	5	4	3	2	1	0	
	PORTF7	PORTF6	PORTF5	PORTF4	-	-	PORTF1	PORTF0	PORTF
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

We knew that these bits needed to be set to their values based on Table 10-1 in the manual for the AtmegaU4. We knew that these were the correct pins to use based on the electrical schematic for the Zumo car, see the figure below.



As seen above, the XOR value for the left encoder is connected to the PB4 pin, the B value for the left encoder is connected to the PE2 pin, the XOR value for the right encoder is

connected to the PE6 pin, and the B value for the right encoder is connected to the PF0 pin. That means to read the A values for either the left or right encoder, the respective XOR and B pins need to be XOR'ed.

Other Important Encoder Registers

11.1.3 External Interrupt Mask Register – EIMSK

Bit	7	6	5	4	3	2	1	0	
	-	INT6	-	-	INT3	INT2	INT1	IINT0	EIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

11.1.2 External Interrupt Control Register B – EICRB

Bit	7	6	5	4	3	2	1	0	
	-	-	ISC61	ISC60	-	-	-	-	EICRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

11.1.4 External Interrupt Flag Register – EIFR

Bit	7	6	5	4	3	2	1	0	
	-	INTF6	-	-	INTF3	INTF2	INTF1	IINTF0	EIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

11.1.5 Pin Change Interrupt Control Register - PCICR

Bit	7	6	5	4	3	2	1	0	
			–	–	–	–	–	PCIE0	PCICR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

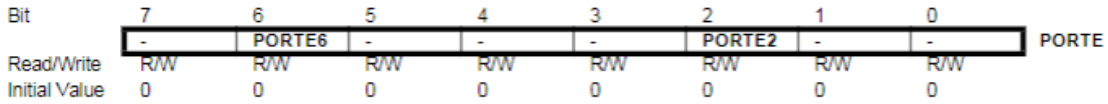
11.1.7 Pin Change Mask Register 0 – PCMSK0

Bit	7	6	5	4	3	2	1	0	
	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

11.1.6 Pin Change Interrupt Flag Register – PCIFR

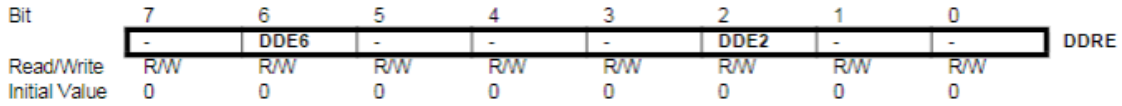
Bit	7	6	5	4	3	2	1	0	
			–	–	–	–	–	PCIF0	PCIFR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

10.4.11 Port E Data Register – PORTE



Bit	7	6	5	4	3	2	1	0	
	-	PORTE6	-	-	-	PORTE2	-	-	PORTE
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

10.4.12 Port E Data Direction Register – DDRE



Bit	7	6	5	4	3	2	1	0	
	-	DDE6	-	-	-	DDE2	-	-	DDRE
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Battery Registers and Pins

Several registers are required to read the battery voltage using an analog to digital conversion. Register ADCSRA is used to set the conversion frequency to 125 kHz by setting bits ADPS0, ADPS1, and ADPS2 to 1, which is a prescaler value of 128. The prescale value of 128 was the only valid option to bring the conversion frequency between 50 kHz and 200 kHz. The register ADCSRA is also used to enable analog to digital conversion by setting the bit ADEN to 1. In order to set the gain to zero, the bits MUX0, MUX3, and MUX4 need to be set to 0 and the bits MUX1 and MUX2 need to be set to 1 in the ADMUX register. Setting bit REFS0 to 1 in register ADMUX will select the voltage reference in the analog to digital conversion. Since the battery uses the analog to digital converter on channel 6 (ADC6), the digital inputs for ADC6 need to be disabled by setting bit ADC6D in register DIDR0 to 1. Lastly, in order to initiate the analog to digital conversion, bit ADSC in register ADCSRA must be set. Once the conversion has completed, the resulting digital value will be stored in registers ADCL and ADCH. All of the registers used for the battery can be found in the ATmega16U4-32U4 Datasheet and are shown below.

Battery Registers

24.9.2 ADC Control and Status Register A – ADCSRA

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ADC Control and Status Register B – ADCSRB

Bit	7	6	5	4	3	2	1	0	
	ADHSM	ACME	MUX5	–	ADTS3	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R/W	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

24.9.1 ADC Multiplexer Selection Register – ADMUX

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

24.9.5 Digital Input Disable Register 0 – DIDR0

Bit	7	6	5	4	3	2	1	0	
	ADC7D	ADC6D	ADC5D	ADC4D	-	-	ADC1D	ADC0D	DIDR0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

24.9.3 The ADC Data Register – ADCL and ADCH

24.9.3.1 ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
Bit	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

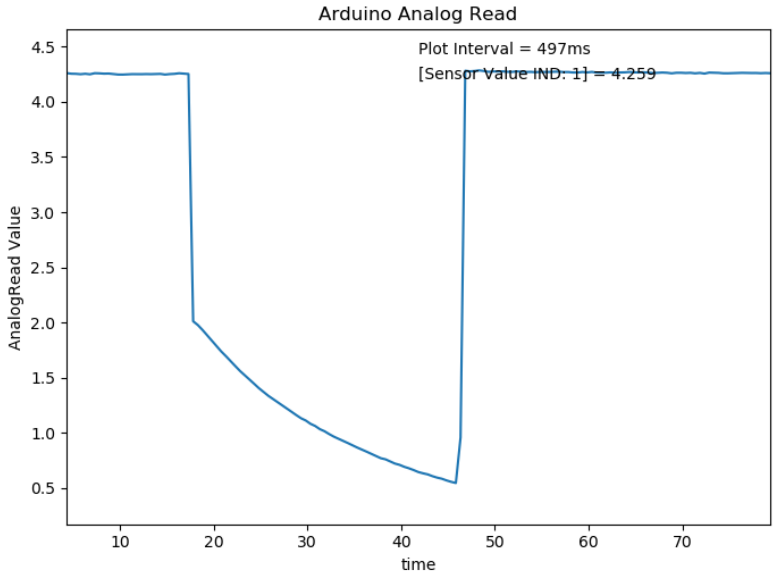
24.9.3.2 ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
Bit	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Filter Design

For our filter for the voltage signal from the battery we made a discrete filter using a Butterworth design. We chose a Butterworth shaped filter due to its flat response throughout the pass region. For the order, we chose a fourth order filter. We used a fourth order filter since we made a same order filter for a prior homework, and its performance looked good. We calculated our coefficients by normalizing our cutoff frequency and using the Matlab 'butter' function to find the coefficients. Our normalized cutoff frequency was 0.04, or a cutoff frequency of 20 Hz with a sampling frequency of 500 Hz (or 2 ms). We chose this cutoff frequency since we found too low of a cutoff frequency resulted in the filtered values being much higher than our expected battery voltage.

We found that our filter does a decent job removing noise in the signal to about two or three decimal places, good enough for our purposes. The graph below shows the performance of the filter when the car is switched from ON to OFF to back ON.



There was no need for a filter on the encoder since we want the raw squarewave data to tell us when a count happened. If we filter that data, it might smooth over the signal, not giving us accurate counts. Furthermore, the noise in the signal is likely not enough to cause a miscount in the first place.

Updated Pseudocode

Functions in Encoder.h/c

FUNCTION Encoders_Init()
DESCRIPTION Initializes the encoders
INPUTS NONE
RETURNS NONE

//Need to enable global interrupts…?
//sei() also does this? Do we need to call sei twice?
SET the 'SREG' register bit 'I' TO 1 //Bit 7

// Enable interrupts on the INT6 pin
SET the 'EIMSK' register bit 'INT6' TO 0 // (needs to be cleared before setting EICRB)
SET the 'EICRB' register bit 'ISC60' TO 1 // Bit 4
SET the 'EICRB' register bit 'ISC61' TO 0 // Bit 5
SET the 'EIFR' register bit 'INTF6' TO 1 // makes sure no interrupts are initially there
SET the 'EIMSK' register bit 'INT6' TO 1 // Bit 6

// Enable interrupts on the PCINT4 pin
SET the 'PCICR' register bit 'PCIE0' TO 1 // Bit 0
SET the 'PCMSK0' register bit 'PCINT4' TO 1 // Bit 4
SET the 'PCIFR' register bit 'PCIE0' TO 1 // makes sure no interrupts are initially there

// Initialize all the necessary pins as inputs
SET the 'DDRB' register bit 'DDB4' TO 0
SET the 'DDRE' register bit 'DDE2' TO 0
SET the 'DDRE' register bit 'DDE6' TO 0
SET the 'DDRF' register bit 'DDF0' TO 0

SET the 'PORTB' register bit 'PORTB4' TO 0
SET the 'PORTE' register bit 'PORTE2' TO 0
SET the 'PORTE' register bit 'PORTE6' TO 0
SET the 'PORTF' register bit 'PORTF0' TO 0

SET _left_counts TO 0
SET _right_counts TO 0

END FUNCTION

FUNCTION Counts_Left()

DESCRIPTION Returns the number of counts from the left encoder
INPUTS NONE
RETURNS Number of counts from the left encoder

RETURN _left_counts

END FUNCTION

FUNCTION Counts_Right()
DESCRIPTION Returns the number of counts from the right encoder
INPUTS NONE
RETURNS Number of counts from the right encoder

RETURN _right_counts

END FUNCTION

FUNCTION Rad_Left()
DESCRIPTION Returns the number of radians from the left encoder
INPUTS NONE
RETURNS Number of radians for the left encoder

CALL Counts_Left()
CONVERT Counts_Left() to radians
RETURN radian conversion

END FUNCTION

FUNCTION Rad_Right()
DESCRIPTION Returns the number of radians from the right encoder
INPUTS NONE
RETURNS Number of radians for the right encoder

CALL Counts_Right()
CONVERT Counts_Right() to radians
RETURN radian conversion

END FUNCTION

Functions in Battery_Monitor.h/c

FUNCTION Battery_Monitor_Init()

DESCRIPTION Initializes functionality for battery monitoring
INPUTS NONE
RETURNS NONE

// Need to set the conversion frequency to be between 50kHz and 200kHz
// Our processor is at 16MHz -> dividing by 128 gives us 125kHz
SET the register 'ADCSRA' bit 'ADPS0' TO 1 // Bit 0
SET the register 'ADCSRA' bit 'ADPS1' TO 1 // Bit 1
SET the register 'ADCSRA' bit 'ADPS2' TO 1 // Bit 2

// Need to set the correct channel to read from and the gain value
// Channel: ADC6. Gain: 1x
SET the register 'ADMUX' bit 'MUX0' TO 0
SET the register 'ADMUX' bit 'MUX1' TO 1
SET the register 'ADMUX' bit 'MUX2' TO 1
SET the register 'ADMUX' bit 'MUX3' TO 0
SET the register 'ADMUX' bit 'MUX4' TO 0

// Enable the car's analog reference & decoupling capacitor
// I.e. enable external reference and capacitor
SET the register 'ADMUX' bit 'REFS0' TO 1 // Bit 6

//Enable the ADC
SET the register 'ADCSRA' bit 'ADEN' TO 1 // Bit 7
SET the register 'ADCSRB' bit 'ADHSM' TO 1 (high speed) (might not need)
SET the register 'ADCSRB' bit 'ADTS2' TO 0 (might not need)
SET the register 'ADCSRB' bit 'ADTS1' TO 0
SET the register 'ADCSRB' bit 'ADTS0' TO 1

//Enable ADC conversion complete interrupt
SET the register 'ADCSRA' bit 'ADIE' TO 1 // Bit 3 (might not be needed)

//Disable digital inputs to enable the ADC on the ADC6
SET the register 'DIDR0' bit 'ADC6D' TO 1 // Bit 6 // set all bits in reg
SET the register 'DIDR2' bit 'ADC6D' TO 1 // set all bits in reg

END FUNCTION

FUNCTION Battery_Voltage()

DESCRIPTION Measures the voltage of the battery using the A/D measurement
INPUTS NONE
RETURNS Battery voltage

START analog to digital conversion // ADC SRA register bit ADSC

READ data in the A/D low register // register ADCL bit ADC6
READ data in the A/D high register // register ADCH bit ADC6
MULTIPLY data in ADCH by 2^8

Value = ADCL + ADCH
Set const to 2
Return value*2

COMBINE data from the high and low register and store in variable v_bat
RETURN v_bat

END FUNCTION