Lab 1 Report

Noah Terry and Lucy Utke

Critical Information

This lab implements the functionality for hardware interrupts and the use of timer 0. Timer 0 is capable of overflow and compare A/B interrupts. Timer0 requires a broader code structure to be implemented such that it can be used for timing information of hardware and software events. To fully test this structure Lab 1's serial communication is to be added to, as specified by Figure 1. These two additional signals will return the current time and the time it takes to complete an iteration in the main loop either once when a lowercase t is sent or continuously when an uppercase T is sent. It will also be necessary to add a timeout structure within Lab 1's serial communication so that if the USB input buffer is left incomplete for over 100ms the buffer is flushed to prevent stalling.

Host -> Device Communication Messing Definition

	CMD char	CMD Hex	# Bytes	Format	Resulting Action
Lab 1	~	0x7e	1	c	Reset the program (return 0)
	*	0x2a	9	cff	Multiply the two floats and return the product.
	/	0x2f	9	cff	Divide the first float by the second and return the result.
	+	0x2b	9	cff	Sum the floats and return the result.
	-	0x2d	9	cff	Subtract the second from the first and return the result.
	UNDEF				If the command char is unrecognized, flush the input buffer, and respond with a '?' (0x3f) character
Lab 2	t	0x54	2	cc	Return the time it requested followed by the time to complete the action specified by the second imput char. 0x00 -> Time Now 0x01 -> Time to complete a full loop iteration Others as you define
	T	0x74	6	ccf	Return the time it requested followed by the time to complete the action specified by the second imput char and returns the time every X milliseconds. If the time is zero or negative it canceles the request.

Device -> Host Communication Messaging Definition
[MSG Length] [Format C-Str][Host Initiating CMD Char][DATA]
Time It Example (time it 0x01): [10] [ccf0x00] [T][0x01] [1.3] Multiply Example: [8] [cf0x00] [*] [-.239] Undefined Command Example (>) : [5] [cc0x00] [>][?] Acceleration Example: [18] [cfff0x00] [A][0.001 -0.09 9.89]

Figure 1: Calculator Specifications

Registers and Pin Usage

Pins within this lab were used for reading and writing to Timer0. Timer0 itself uses the register TCNT0, which tracks the timer counter value. In order to allow comparison interrupts the bits COM0A0 and COM0A1 need to be set to 0 in the TCCR0A register. The bits CS01 and CS00 need to be set to 1 in the TCCR0B register in order to set up timer 0 with a prescaler of

64 which will provide the desired precision. The counter value from TCNT0 is continuously compared with the register OCR0A which is set to 249.

The PC7 pin is the pin used for timer 0 and is shown in the last image below. The reason timer 0 is used is because the other timers are connected to pins with useful functionality such as the motor PWM for timer 1, the buzzer PWM for timer 4, and the proximity LED PWM for timer 3. The registers to use were found in the ATmega16U4-32U4 datasheet. The following images show the details for each of the necessary registers to control timer 0.

13.8.1 Timer/Counter Control Register A – TCCR0A

Bit	7	6	5	4	3	2	1	0	
	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:6 – COM01A:0: Compare Match Output A Mode**

These bits control the Output Compare pin (OC0A) behavior. If one or both of the COM0A1:0 bits are set, the OC0A output overrides the normal port functionality of the I/O pin it is connected to. However, note that the Data Direction Register (DDR) bit corresponding to the OC0A pin must be set in order to enable the output driver.

When OC0A is connected to the pin, the function of the COM0A1:0 bits depends on the WGM02:0 bit setting. The table shows the COM0A1:0 bit functionality when the WGM02:0 bits are set to a normal or CTC mode (non-PWM).

13.8.2 Timer/Counter Control Register B – TCCR0B

Bit	7	6	5	4	3	2	1	0	
	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – FOC0A: Force Output Compare A**

The FOC0A bit is only active when the WGM bits specify a non-PWM mode.

However, for ensuring compatibility with future devices, this bit must be set to zero when TCCR0B is written when operating in PWM mode. When writing a logical one to the FOC0A bit, an immediate Compare Match is forced on the Waveform Generation unit. The OC0A output is changed according to its COM0A1:0 bits setting. Note that the FOC0A bit is implemented as a strobe. Therefore it is the value present in the COM0A1:0 bits that determines the effect of the forced compare.

A FOC0A strobe will not generate any interrupt, nor will it clear the timer in CTC mode using OCR0A as TOP.

The FOC0A bit is always read as zero.

13.8.4 Output Compare Register A – OCR0A

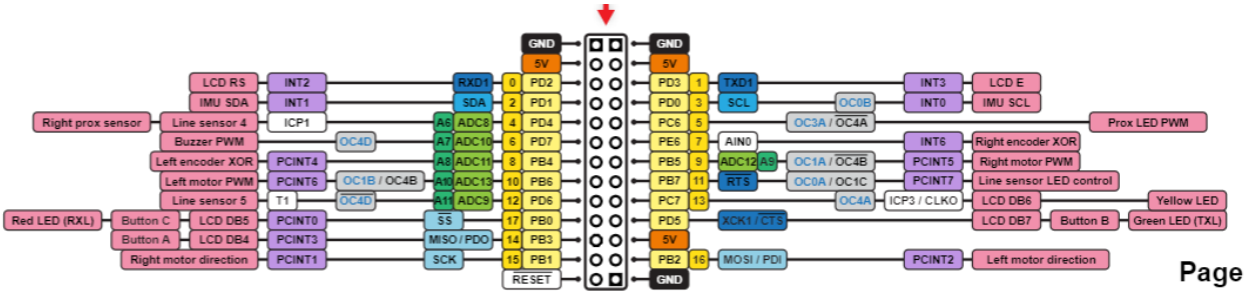
Bit	7	6	5	4	3	2	1	0	
	OCR0A[7:0]								OCR0A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Output Compare Register A contains an 8-bit value that is continuously compared with the counter value (TCNT0). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC0A pin.

13.8.3 Timer/Counter Register – TCNT0

Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
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	

The Timer/Counter Register gives direct access, both for read and write operations, to the Timer/Counter unit 8 -bit counter. Writing to the TCNT0 Register blocks (removes) the Compare Match on the following timer clock. Modifying the counter (TCNT0) while the counter is running, introduces a risk of missing a Compare Match between TCNT0 and the OCR0x Registers.



Future Modifications

With the current functionality of Lab2 it is possible to time events sent to the robot as well as flush the input buffer if it goes stale. These require very little modification to be used and will be excellent for both debugging and preventing the serial from getting stuck. However, this is not the only way that timers can be used. When encoders are implemented they will require a timer to interpret the pulse frequency. This frequency will allow for understanding of the velocity of the robot, and will require reconfigurations of Lab 2's code. Within Lab 3, the functions Counts_Left() and Count_Right() return the number of counts from their respective encoders. These will act as a hardware interrupt, and use the same core logic as the compare interrupts created within Lab 2. Furthermore, these returned values will need to be compared to the time over which they were collected in order to understand the encoder velocity. This can use Lab 2's GetTime() function without modification.