Contents

Actuators

Version	1.0.0
Files	actuators.c actuators.h
Needs	'timers' library
Description	This library is able to control 2 electric actuators.
Remarks	None
Instructions	1. Call actuatorsInit() in your main program initialization 2. Enable interrupts in your program (<avr/interrupt.h> is needed (compiler))

actuatorsInit

Description Initialize the actuator ports and calibrate the actuators hardware limits.

Parameters

uint8_t	*minActuator1*	0-255	Minimum position of actuator 1, which equals 'value' 0 at actuatorsPositionSet()
uint8_t	*maxActuator1*	0-255	Maximum position of actuator 1, which equals 'value' 100 at actuatorsPositionSet()
uint8_t	*minActuator2*	0-255	Minimum position of actuator 2, which equals 'value' 0 at actuatorsPositionSet()
uint8_t	*maxActuator2*	0-255	Maximum position of actuator 1, which equals 'value' 100 at actuatorsPositionSet()

Returns None

Example(s)

```
actuatorsInit(25,103,0,0); // Initialize the actuators
                           // actuator 1 moves between 25,103 and actuator 2 isn't used
```

actuatorsEnable

Description Enable or disable an actuator. Only leave them enabled when needed.

Parameters

uint8_t	*actuator*	1-2	Select the actuator
uint8_t	*state*	0-1	1 = on, 0 = off

Returns None

Example(s)

```
actuatorsEnable(1,1);      // Enables actuator 1
```

actuatorsPositionSet

Description	Set a new position for a actuator. You may do this before enabling the actuator.			
Parameters	uint8_t	*actuator*	1-2	Select the actuator
	uint8_t	*position*	0-100	The new position
Returns	None			
Example(s)	actuatorsPositionSet(1,50);		// Set the position of actuator 1 to 50 out of 100	

ADC

Version	1.0.0
Files	adc.c adc.h
Needs	None
Description	This library controls the analog inputs and supplies readout.
Remarks	None
Instructions	1. Call adcInit() in your main program initialization

adcInit

Description	Initialize the analog inputs.
Parameters	None
Returns	None
Example(s)	adcInit(); // Initialize the analog inputs

adcGet

Description	Get the current value of an analog input			
Parameters	uint8_t	*adc*	1-2	Select the analog input
	uint8_t	*averaging*	1-255	How many measurements must be taken before returning the averaged value (at least 16 is advised).
Returns	uint16_t	value	0 – 1023	The value of the channel
Example(s)	myVar = adcGet(1,16); // get an averaged value (16 measurements) for analog input 1			

Buzzer

Version	1.0.0
Files	buzzer.c buzzer.h
Needs	None
Description	This library controls the buzzer.
Remarks	None
Instructions	1. Call buzzerInit() in your main program initialization 2. Enable interrupts in your program (<avr/interrupt.h> is needed (compiler))

buzzerInit

Description	Initialize the buzzer output and timer
Parameters	None
Returns	None
Example(s)	buzzerInit(); // Initialize the buzzer

buzzerEnable

Description	Enable or disable the buzzer			
Parameters	uint8_t	*state*	0-1	1 = on, 0 = off
Returns	None			
Example(s)	buzzerEnable(1); // Enable the buzzer			

buzzerToneSet

Description	Set the tone of the buzzer. You may do this before enabling it.			
Parameters	uint16_t	*tone*	0-923	Set the tone between 61Hz and 20kHz. Note that it isn't linear. The higher a tone, the bigger the frequency steps.
Returns	None			
Example(s)	buzzerToneSet(448); // Set the tone of the buzzer around 1kHz			

Hitachi (display)

Version	1.0.0
Files	hitachi.c hitachi.h
Needs	None
Description	This library controls an optional display with a hitachi chip (most character displays). Only basic functions will be explained. Other possibilities to send/receive can be found in the AVR-libc Library Reference.
Remarks	None
Instructions	1. Call displayInit() in your main program initialization

displayInit

Description Initialize the display and backlight.

Parameters	uint8_t	*width*	1-40	The width of the display
	uint8_t	*lines*	1-4	The number of lines on the display

Returns None

Example(s) displayInit(20,2); // Initialize a display of 20 characters wide and 2 lines

displayGotoXY

Description Place the cursor at a certain position on the display

Parameters	uint8_t	*x*	0-39	X position (character number – 1)
	uint8_t	*y*	0-3	Y position (line number – 1)

Returns None

Example(s) displayGotoXY(0,1); // Goto begin of second line

displayClear

Description Clear the display content and reset the cursor location to the top left

Parameters None

Returns None

Example(s) displayClear(); // Clear the display content

displayCursor

Description	Show or hide the cursor			
Parameters	uint8_t	*onoff*	0-1	1 = show, 0 = hide
	uint8_t	*blink*	0-1	1 = blinking cursor (when showed), 0 = cont.
Returns	None			
Example(s)	displayCursor(1,1);	// Show a blinking cursor		

displayBacklight

Description	Enable or disable the backlight			
Parameters	uint8_t	*state*	0-1	1 = on, 0 = off
Returns	None			
Example(s)	displayBacklight(1);	// Enable the backlight		

fputc

Description	Put a character on the display. This is actually a GCC library (stdio.h) but you need to know how to use it for the display. For a complete reference of this function look into the GCC reference.			
Parameters	char	*data*	0-255	The character you want to put on the stream
	FILE	*stream*	FILE *	The stream you want to put a character to
Returns	int	*data*	0-255	The character you have put on the stream
Example(s)	fputc('H',display);	// Put the character 'H' on the display		

fputs

Description	Put a string with variables on the display. This is actually a GCC library (stdio.h) but you need to know how to use it for the display.			
Parameters	char*	*data*	char*	The string you want to put on the stream
	FILE	*stream*	FILE *	The stream you want to put a character to
Returns	int	*data*	0-255	The character you have put on the stream
Example(s)	fputs("Hoi",display);	// Put 'Hoi' on the display		

fprintf

Description	Put a string with variables on the display. This is actually a GCC library (stdio.h) but you need to know how to use it for the display.			
Parameters	FILE	*stream*	FILE *	The stream you want to put a string to
	For the rest of the parameters/returns look into the GCC reference for fprintf			
Example(s)	fprintf(display,"The variable is on %d",myVar);	// Print a string on the display		

I2C

Version	1.0.0
Files	i2c.c I2c.h
Needs	None
Description	This library reads the digital inputs.
Remarks	Input 3 and output 3 cannot be used when using i2c! All devices supported by YAAB will have their own library, just use i2cInit from this library.
Instructions	1. Call i2cInit() in your main program initialization

I2cInit

Description	Initialize the i2c ports
Parameters	None
Returns	None
Example(s)	i2cInit(); // Initialize the i2c ports

I2cStart

Description	Send a (repeated)start condition
Parameters	None
Returns	None
Example(s)	i2cStart(); // Reset the i2c bus and start new command

I2cStop

Description	Send a stop condition
Parameters	None
Returns	None
Example(s)	i2cStop(); // Send a stop command on the i2c bus

I2cWr

Description	Write a byte to the i2c bus			
Parameters	uint8_t	*data*	0-255	The data you want to write to the bus
Returns	uint8_t	*error*	0-1	1 = NACK, 0 = ACK

Example(s)

```
myVar = i2cWr(0x80);        // Write 0x80 to the bus
If (myVar == 1)  {
            // We received a NACK
}
```

I2cRd

Description	Read a byte from the i2c bus			
Parameters	uint8_t	*ack/nack*	0-1	Return a ack/nack 1 = ACK, 0 = NACK
Returns	uint8_t	*data*	0-255	The data read from the i2c bus

Example(s)

```
myVar = i2cRd(1);           // Get a byte from the i2c bus and return a ack
```

I2c IO

Version	1.0.0
Files	i2cIO.c I2cIO.h
Needs	i2c library
Description	This library controls the i2c I/O Expander(s) for YAAB
Remarks	You can control a maximum of eight I/O Expanders
Instructions	1. Call i2cIOConfigure() in your main program initialization for each I/O Expander

I2cIOConfigure

Description	Configure a I/O Expander			
Parameters	uint8_t	*address*	0-7	Which address you want to configure
	uint8_t	*outputs*	0-255	Which I/Os should be output (1 = output, 0 = input) (bit 0 = i/o 0, etc)
Returns	None			
Example(s)	i2cIOConfigure(0,0xF0);	// Configure expander 0, setting upper 4 i/o on output		

I2cIOSet

Description	Set a single i/o on a expander (which is configured as output)			
Parameters	uint8_t	*address*	0-7	Which address you want to set the i/o on
	uint8_t	*io*	1-8	Which i/o you want to set
	uint8_t	*state*	0-1	1 = on, 0 = off
Returns	None			
Example(s)	i2cIOSet(0,1,1);	// Set the first i/o to 'on' on expander 0		

I2cIOSetRaw

Description	Set all i/o on a expander (discarding the output/input settings)			
Parameters	uint8_t	*address*	0-7	Which address you want to set the i/o on
	uint8_t	*io*	0-255	Which I/Os should be on/off (1 = on, 0 = off) (bit 0 = i/o 0, etc)
Returns	None			
Example(s)	i2cIOSetRaw(0,0x0F);	// Set the first 4 i/o 'on' on expander 0		

I2cIOGet

Description	Get the status of a single i/o (which is configured as input)			
Parameters	uint8_t	*address*	0-7	Which address you want to get the i/o from
	uint8_t	*io*	1-8	Which i/o status you want to get
Returns	uint8_t	*state*	0-1	1 = on, 0 = off
Example(s)	myVar = i2cIOGet(0,4); // Get the fourth i/o status of expander 0			

I2cIOGetRaw

Description	Get the status of all i/o (discarding they are in- or output)			
Parameters	uint8_t	*address*	0-7	Which address you want to get the i/o's from
Returns	uint8_t	*io*	0-255	The i/o status of all i/o (1 = on, 0 = off) (bit 0 = i/o 0, etc)
Example(s)	i2cIOGetRaw(0); // Get all the i/o status of expander 0			

Inputs

Version	1.0.0
Files	inputs.c inputs.h
Needs	None
Description	This library reads the digital inputs.
Remarks	None
Instructions	1. Call inputsInit() in your main program initialization

inputsInit

Description	Initialize the digital inputs.			
Parameters	None			
Returns	None			
Example(s)	inputsInit(); // Initialize the digital inputs			

inputsGet

Description	Get the current state of an input.			
Parameters	uint8_t	*input*	1-5	Which input you want to read
Returns	uint8_t	*state*	0-1	1 = active, 0 = idle
Example(s)	myVar = inputsGet(2); // Get the state of digital input 2			

inputsGetPressedDebounced

Description	See if the input is pressed (with debounce check of 50ms)			
Parameters	uint8_t	*input*	1-5	Which input you want to read
Returns	uint8_t	*state*	0-1	1 = pressed, 0 = idle
Example(s)	myVar = inputsGetPressedDebounced(2); // See if input 2 is pressed (debounced)			

inputsGetReleasedDebounced

Description	See if the input is released (with debounce check of 50ms)			
Parameters	uint8_t	*input*	1-5	Which input you want to read
Returns	uint8_t	*state*	0-1	1 = released, 0 = active
Example(s)	myVar = inputsGetReleasedDebounced(3); // See if input 3 is released (debounced)			

Music

Version	1.0.0
Files	music.c music.h
Needs	buzzer library Timers library
Description	This library plays music generated with YAABStudio
Remarks	Create a song in YAABStudio and paste the generated table in your main.c
Instructions	1. Call musicInit() in your main program initialization 2. Call musicProcess() in your main loop 3. Make sure that <avr/pgmspace.h> is included

musicInit

Description	Initialize the music library
Parameters	None
Returns	None
Example(s)	musicInit(); // Initialize the music library

musicProcess

Description	Process the current song (if started)
Parameters	None
Returns	None
Example(s)	musicProcess(); // Play music

musicTrackStart

Description	Start playing a song. If already playing, you don't need to stop the other first		
Parameters	_ musicTrack **ptr*	trackName	Which track you want to play (table name)
Returns	None		
Example(s)	musicTrackStart(mario); // Start a track called 'mario'		

musicTrackStop

Description	Stop playing the current song.
Parameters	None
Returns	None
Example(s)	musicTrackStop(); // Stop playing the current song

NTC

Version	1.0.0
Files	ntc.c ntc.h
Needs	None
Description	This library converts a ADC value of a ntc to a known temperature unit
Remarks	None
Instructions	None

ntcGet

Description	Calculate a temperature			
Parameters	uint16_t	*adc*	0-65535	The ADC value of a ntc
	uint8_t	*unit*	NTC_UNIT_CELCIUS	Celcius
			NTC_UNIT_FAHRENHEIT	Fahrenheit
			NTC_UNIT_KELVIN	Kelvin
			NTC_UNIT_RANKINE	Rankine
			NTC_UNIT_REAUMUR	Reaumur
			NTC_UNIT_DELISLE	Delisle
			NTC_UNIT_ROMER	Romer
			NTC_UNIT_NEWTON	Newton
Returns	float	*temperature*	-x.x-x.x	The temperature of the given adc value
Example(s)	myValue = ntcGet(703,NTC_UNIT_CELCIUS); // Calc. the temperature in ‘C with 703 bit ADC			

Outputs

Version	1.1.0
Files	outputs.c outputs.h
Needs	None
Description	This library controls the digital outputs.
Remarks	None
Instructions	1. Call outputsInit() in your main program initialization

outputsInit

Description	Initialize the digital outputs.
Parameters	None
Returns	None
Example(s)	outputsInit(); // Initialize the digital outputs

outputsGet

Description	Get the current state of an output.			
Parameters	uint8_t	*output*	1-5	Which output you want to read
Returns	uint8_t	*state*	0-1	1 = on, 0 = off
Example(s)	myVar = outputsGet(1); // Get the state of digital output 1			

outputsSet

Description	Set the state of an output.			
Parameters	uint8_t	*output*	1-5	Which output you want to set
	uint8_t	*state*	0-2	2 = toggle, 1 = on, 0 = off
Returns	None			
Example(s)	outputsSet(2,1); // Enable digital output 2			

PWM

Version	1.0.0
Files	pwm.c pwm.h
Needs	None
Description	This library controls the pwm outputs.
Remarks	<u>*Frequency*</u> You can set only 1 frequency for both PWM outputs due they are on the same timer. Note that that each frequency is also linked to the resolution (8-10 bits).
Instructions	1. Call pwmInit() in your main program initialization

pwmInit

Description	Initialize the pwm outputs.
Parameters	None
Returns	None
Example(s)	pwmInit(); // Initialize the pwm outputs

pwmConfigure

Description Configure the pwm outputs (also enable/disable)

Parameters

uint8_t	*speed*	PWM_SPEED_*	Set the pwm frequency (values in pwm.h)
uint8_t	*statePwm1*	PWM_STATE_*	Set the state for pwm 1 (values in pwm.h)
uint8_t	*statePwm2*	PWM_STATE_*	Set the state for pwm 2 (values in pwm.h)

Returns None

Example(s)

```
pwmConfigure(PWM_SPEED_244,PWM_STATE_NORMAL,PWM_STATE_INVERTED);
// PWM timer on 244Hz, PWM 1 enabled normal, PWM 2 enabled with inverted signal
```

pwmValueSet

Description Set the value of a pwm

Parameters

uint8_t	*pwm*	1-2	Selected pwm output
Uint16_t	*value*	0-255/511/1023	Set the value. Depending on the SPEED you have taken, its maximum value divers (8/9/10 bit).

Returns None

Example(s)

```
(1,511); // Set pwm 1 on value of 128, halfway the 10bit pwm (+/- 2,5V)
```

Timers

Version	1.1.0
Files	timers.c timers.h
Needs	None
Description	This library allows you to have timeouts and repeating timers for custom use.
Remarks	*Using timer/waits* It is your own decision to use a blocking wait or a timeout. When you need a short wait you can easily use blocking waits because they cost less cpu time and you don't need to check it. For long timeouts or when you want to do something else while waiting (and not very critical), you can use the timer. This is a background task and you can check yourself if it's done yet. It is also possible to call a function when the timer is done (this is useful when critical timing is needed) You can also time things with a timer (see how much time was used for something).
Instructions	1. Call timersInit() in your main program initialization 2. If you want to use timers ('timersSet') make sure you enable interrupts ('sei();' and that <avr/interrupt.h> is included).

timersInit

Description	Initialize the timers.
Parameters	None
Returns	None
Example(s)	timersInit(); // Initialize the timers.

timersWaitUs

Description	Wait a certain amount of microseconds (us) before continuing (blocking, so program hangs)		
Parameters	uint16_t *value*	0-65535	The us you want to wait
Returns	None		
Example(s)	timersWaitUs(500); // Wait for 500 us here		

timersWaitMs

Description	Wait a certain amount of milliseconds (ms) before continuing (blocking, so program hangs)		
Parameters	uint16_t *value*	0-65535	The ms you want to wait
Returns	None		
Example(s)	timersWaitMs(500); // Wait for 500 ms here		

timersSet

Description	Set a timer. You need to keep track of your own timers which are in use or free. You can reset a timer at any given time. For example: every time you press a button, you set the timer for 1s. When somebody doesn't press the button within 1 second, you can do something. It is also possible to execute a (callback) function when the timer is triggered.			
Parameters	uint8_t	*timer*	1-5	The timer you want to use
	uint16_t	*timeoutMs*	1-65535	The timeout in Ms
	uint8_t	*repeat*	0-1	1 = keep repeating the timeout and set a flag when triggered, 0 = just once
	void	**callback(void)*	TIMERS_CALL_*x*	This is the callback function that will be executed when the timer is triggered. *x* = 1 – 5. Using a callback instead of calling timersTriggered(), you can be sure that your code is executed immediately. A callback pauses the normal program and forces the callback function to be executed. After this the program is resumed. Because everything stops during execution, you need to do as little as possible in this function.
Returns	None			

Example(s)

```
timersSet(1,1000,1,0); // Create a timer which triggers each second (1000ms) without
                          callback
timersSet(2,500,0,TIMERS_CALL_2); // Create a timeout of 500ms(0,5s) and execute
                           the 2nd callback function when timer is triggered
```

The name of the callback function is "timersCallback*x*" where *x* = 1 – 5.
Example for implementing the 2nd callback:

```
timersCallback2 {
        // Put your code here
}
```

timersGet

Description	Get the current value of a timer to see how much time Is left.			
Parameters	uint8_t	*timer*	1-5	The timer you want to read
Returns	uint16_t	*value*	0-65535	The value of the timer in timer ticks. To get ms: ms = value / TIMERS_1MS

Example(s)

```
myValue = timersGet(1);            // get the value of timer 1
myMs = myValue / TIMERS_1MS;   // Convert the value to milliseconds
```

timersTriggered

Description	See if the timer has triggered/ended.			
Parameters	uint8_t	*timer*	1-5	The timer you want to check
Returns	uint8_t	*status*	0-1	1 = the timer has triggered (repeat)/ended 0 = the timer is still counting

Example(s)

```
myValue = timersTriggered(1); // See if timer 1 has triggered
```

UART

Version	1.0.0
Files	uart.c uart.h
Needs	None
Description	This library allows you to communicate via the uart. Only basic functions will be explained. Other possibilities to send/receive can be found in the AVR-libc Library Reference.
Remarks	*Receive buffer* The receive buffer is 64 bytes, which means a pc (or other device) may send 64 characters at one time, and the YAAB can handle it in the background. Once 1 byte is read by the YAAB you can send a character again (FIFO). *Communication mode* The uart library is always communication with 8-bits, no parity and without any flowcontrol.
Instructions	1. Call uartInit() in your main program initialization 2. Enable interrupts in your program (<avr/interrupt.h> is needed (compiler))

uartInit

Description	Initialize the uart.			
Parameters	uint16_t	*baudrate*	UART_BAUD_*	The baudrate you want (found in uart.h)
	uint8_t	*stopbits*	1-2	The number or stopbits you want
Returns	None			
Example(s)	uartInit(UART_BAUD_57600,1); // Initialize the uart on 57K6 and with 1 stopbit.			

uartGetCount

Description	See how many bytes are received (and which you are allowed to read).			
Parameters	None			
Returns	uint8_t	*count*	0-64	The number of bytes in the receive buffer
Example(s)	myVar = uartGetCount(); // Get the number of bytes in the receive buffer			

uartBufferClear

Description	Clear the receive buffer.
Parameters	None
Returns	None
Example(s)	uartBufferClear(); // Throw all received bytes away

fputc

Description	Put a character on the uart. This is actually a GCC library (stdio.h) but you need to know how to use it for the uart. For a complete reference of this function look into the GCC reference.			
Parameters	char	*data*	0-255	The character you want to put on the stream
	FILE	*stream*	FILE *	The stream you want to put a character to
Returns	int	*data*	0-255	The character you have put on the stream
Example(s)	fputc('H',uart); // Put the character 'H' on the uart			

fgetc

Description	Get a character from the receive buffer. This is actually a GCC library (stdio.h) but you need to know how to use it for the uart. For a complete reference of this function look into the GCC reference.			
Parameters	FILE	*stream*	FILE *	The stream you want to get a character from
Returns	int	*data*	0-255	The character from the stream
Example(s)	myVar = fgetc(uart); // Get a character from the uart receive buffer			

fputs

Description	Put a string with variables on the uart. This is actually a GCC library (stdio.h) but you need to know how to use it for the uart.			
Parameters	char*	*data*	char*	The string you want to put on the stream
	FILE	*stream*	FILE *	The stream you want to put a character to
Returns	int	*data*	0-255	The character you have put on the stream
Example(s)	fputs("Hoi",uart); // Put 'Hoi' on the uart			

fprintf

Description	Put a string on the uart. This is actually a GCC library (stdio.h) but you need to know how to use it for the uart.			
Parameters	FILE	*stream*	FILE *	The stream you want to put a string to
	For the rest of the parameters/returns look into the GCC reference for fprintf			
Example(s)	fprintf(uart,"The variable is on %d",myVar); // Print a string on the uart			