Byggern

Release latest

Simen Eine, Jørgen Dyrskog, Line Høybakk

Jan 01, 1980

NODES

Welcome to the documentation of TTK4155 Ping Pong source code. This is the API documentation of the Ping Pong game developed in the course TTK4155 (also known as "Byggern").

The Ping Pong game is built on two modules with each own source code that independently controls part of the system and communicates over CAN.

Node 1 receives control data and sends these to Node 2. Node 2 controls the Ping-Pong board. The Ping-Pong board has a servo, a solenoid and a motor that you can play ping-pong with.

The source code can be read at GitHub

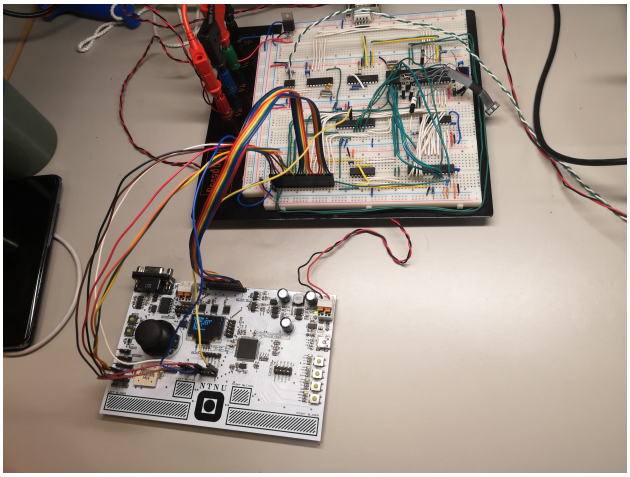

Fig. 1: Node 1

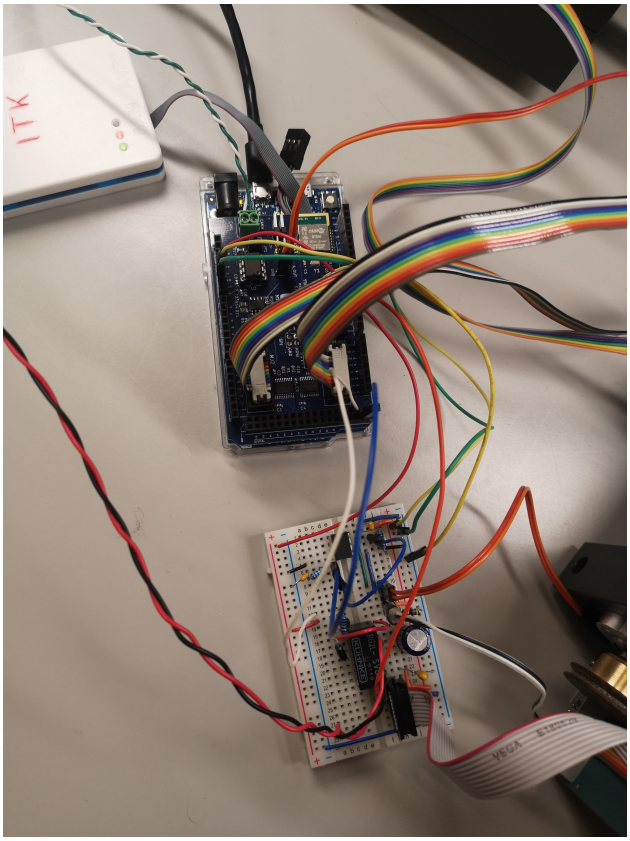

Fig. 2: Node 2

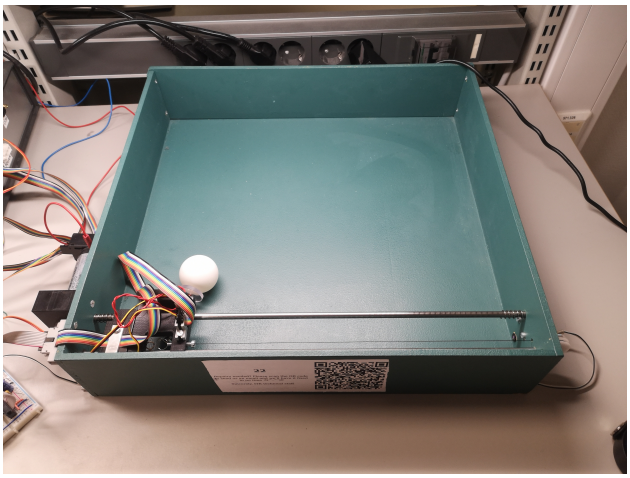

Fig. 3: Ping Pong board

CHAPTER

ONE

NODES

1.1 Documentation for Node 1

Node 1 contains an AVR162 microcontroller that gathers sensor data for the game and sends these to the game controller (Node 2).

Node 1 also contains a menu displayed on an oled screen. This menu is used to start the game or do hardware tests.

1.1.1 Modules

UART Node 1

The UART in Node 1 can be connected via a RS232 converter over serial to the desktop computers at the lab.

The UART is configured with 9600 baud. The UART can be used via the *printf* function. This example shows how to use *printf* with the UART API:

```
uart_setup();
fdevopen(send_character, receive_character);
printf("Hello world");
```

API Reference

group **uart**
UART library API for node 1.

Defines

BAUD
UART baud rate

F_CPU
External oscillator frequency

Functions

int **send_character**(char data, FILE *file)
Send character over UART.

Parameters

- **data** – char to send
- **file** – C stream

Returns 0 when done transmitting

uint8_t **uart_getchar**()
Receive character over UART.

Returns uint8_t Received byte

int **receive_character**(FILE *file)
Setup stream for printf.

Parameters

- **file** – C stream

Returns int Received byte

void **uart_setup**(void)
Setup UART communication on node1.

CAN Node 1

Node 1 CAN library communicates with a MCP2515 can transceiver. The API to communicate over CAN includes functions for using this transceiver, SPI and basic CAN functions.

The Highes level usage for this library is via the *can_frame* struct, this can be sent via the can API.

API Reference

group **can**
CAN library API for node 1.

Defines

F_CPU
External oscillator frequency

BRP
Baud rate prescaler. BRP = 3 yields a Time Quanta (TQ) = 500ns

PROPSEG
Propagation segment length = 2

PS1
Phase segment 1 length = 7

PS2
Phase segment 2 length = 6

Functions

void **can_setup**()
Sets up CAN communication on the MCP2515.

void **can_write**(struct *can_frame* *can_msg)
Transmits a message over CAN.

Parameters

- **can_msg** – Pointer to the message frame to be transmitted

uint8_t **get_n_new_messages**(uint8_t status_byte)
Get the number of new received messages.

Parameters

- **status_byte** – Byte representing rx status from can driver.

Returns uint8_t Number of new messages ready to be received.

uint8_t **can_receive**(struct *can_frame* *can_msg)
Receive can message.

Parameters

- **can_msg** – Pointer to *can_frame* struct where the new message will be stored.

Returns uint8_t Number of messages received

void **can_test**()
Test CAN communication sending messages with increasing IDs.

struct **can_frame**
#include <*can_lib.h*> CAN message struct.

group **mcp2515**
MCP2515 API.

Enums

enum **rx_read_mode**
Enum used to choose receive buffer in the MCP2515

Values:

enumerator **RXBUF0_START_ID**

enumerator **RXBUF0_START_DATA**

enumerator **RXBUF1_START_ID**

enumerator **RXBUF1_START_DATA**

enum **tx_write_mode**
Enum used to choose transmit buffer in the MCP2515

Values:

enumerator **TXBUF0_START_ID**

enumerator **TXBUF0_START_DATA**

enumerator **TXBUF1_START_ID**

enumerator **TXBUF1_START_DATA**

enumerator **TXBUF2_START_ID**

enumerator **TXBUF2_START_DATA**

Functions

uint8_t **mcp_read_byte**(uint8_t address)
Read register of MCP2515 with passed address.

Parameters

- **address** – 8-bit memory address

Returns Data byte

void **mcp_read_rxbuffer**(uint8_t read_mode, uint8_t *buffer, uint8_t len)
Read the RX buffer of MCP2515.

Parameters

- **read_mode** – Type rx_read_mode. Defines what RX buffer to read from
- **buffer** – Pointer to receive buffer
- **len** – Length of data to receive

void **mcp_write_byte**(uint8_t address, uint8_t data)
Write one byte of data to the MCP2515.

Parameters

- **address** – Memory address to write to
- **data** – Byte to be written

void **mcp_load_txbuffer**(uint8_t write_mode, uint8_t *data, uint8_t len)
Load transmit buffer of MCP2515.

Parameters

- **write_mode** – Type tx_write_mode. Defines what TX buffer to use
- **data** – Pointer to local data buffer
- **len** – Number of bytes to be written

void **mcp_request_to_send**()
Request transmission of TX buffer 0 on the MCP2515.

void **mcp_bit_modify**(uint8_t address, uint8_t mask, uint8_t data)
Set or clear individual bits in the MCP2515 using a mask.

Parameters

- **address** – Address of the register to modify
- **mask** – Determines what bits are allowed to change
- **data** – Determines the value of the modified bits

void **mcp_reset**()
Resets the MCP2515. Same as hardware reset.

uint8_t **mcp_read_status**()
Read status register of MCP2515.

Returns Status byte

uint8_t **mcp_read_rx_status**()
Read status of the reiceve buffers in the MCP2515.

Returns RX buffer status byte

void **mcp_setup_loopback**()
Setup the MCP2515 in loopback mode.

group **spi**
SPI library API for node 1.

Defines

SPI_SS
Slave select pin

SPI_MOSI
Master out, slave in pin

SPI_MISO
Master in, slave out pin

SPI_SCK
SPI clock pin

Functions

void **spi_setup**()
Initialize SPI on the ATMega162.

uint8_t **spi_transceiveByte**(uint8_t data)
Transmit and receive one byte over SPI.

Parameters

- **data** – Byte to transmit

Returns Received byte

int8_t **spi_transceive**(uint8_t *tx, uint8_t *rx, uint8_t txLen, uint8_t rxLen)
Transmit and receive bytes over SPI.

Parameters

- **tx** – Pointer to transmit buffer
- **rx** – Pointer to receive buffer
- **txLen** – Number of bytes to transmit
- **rxLen** – Number of bytes to receive

Returns 0 when transceive operation is done

OLED

The OLED in Node 1 is used for menu and game feedback. The library for OLED includes functions for controlling the OLED, fonts, images and a menu library.

API Reference

group **oled_menu**
OLED menu API for node 1.

Defines

MENU_LENGTH

SCREEN_WIDTH

Functions

void **print_menu**()
Print the menu saved to current_menu to the terminal.

void **change_menu**(void *next_page)
Save a new menu to current_menu and print it.

Parameters

- **next_page** – pointer to next page

void **run_option**()
Run the callback function in selected option.

void **select_up**()
Move the option selection one option up.

void **select_down**()
Move the option selection one option down.

void **game_menu**(uint8_t score)
Show game stats during game.

Parameters

- **score** – Current game score

void **game_over**(uint8_t score)
Show result of game after it is over.

Parameters

- **score** – Final game score

struct **menu_option**
#include <oled_menu.h> Struct representing option for menu.

- name: Array of characters representing option text.

- callback: Pointer to callback function that will be called when selected. This function takes a void pointer that can be used to pass data.
- callback_parameter: void pointer that will be passed to callback.

struct **menu_page**
#include <oled_menu.h> Struct representing a page in the menu.

- title: Array of characters representing title of page.
- options: Array of menu options that page will contain.

group **oled**
OLED API for Node 1.

Functions

void **oled_setup**(void)
Set up oled.

void **oled_clear**()
Clear oled screen.

void **oled_set_column**(uint8_t column)
Set selected column.

Parameters

- **column** – Column between 0 and 7

void **oled_set_line**(uint8_t line)
Set selected line.

Parameters

- **line** – Line between 0 and 127

void **oled_set_page**(uint8_t page)
Set selected page.

Parameters

- **page** – Page between 0 and 7

void **oled_set_pos**(uint8_t page, uint8_t column)
Set selected position.

Parameters

- **page** – Page between 0 and 7
- **column** – Column between 0 and 127

void **oled_command_write**(uint8_t command)
Write command to oled.

Parameters

- **command** – Oled command

void **oled_data_write**(uint8_t data)
Write data to oled.

Parameters

- **data** – Oled data

void **oled_putchar**(uint8_t c)
Put character to oled screen.

Parameters

- **c** – Char character to write

int **oled_send_character**(char data, FILE *file)
Send character (wrapper for putchar)

Parameters

- **data** – Character to write
- **file** – C stream

Returns int Always returns 0

void **oled_print**(char *text)
Print string to oled.

Parameters

- **text** – Array of characters

void **oled_print_number**(uint8_t number)
Print longer number to oled.

Parameters

- **number** – Number between 0 and 255

void **show_img_troll**()
Show image of a troll on OLED.

void **show_img_bird**()
Show image of bird on OLED.

void **show_img_skyline**()
Show image of a skyline on OLED.

Joystick and slider

The joystick and slider in Node 1 controls the ping-pong board. The values for joystick and slider are read through the ADC and through interrupts.

The library for joystick and slider can be used for updating and reading values from joystick and slider.

API Reference

group **js_slider**
Joystick and slider API for node 1.

Defines

F_CPU
External oscillator frequency

ADC_START_ADR
Start address for ADC read.

Enums

enum **JS_position**
Joystick direction enum

Values:

enumerator **DEFAULT**

enumerator **RIGHT**

enumerator **LEFT**

enumerator **UP**

enumerator **DOWN**

Functions

void **js_slider_update**(void)
Update joystick and slider values from ADC.

void **calibrate_joystick**(void)
Averages X amount of adc-measurements to set the default adc-values for the joystick in default position.

bool **new_slider_left**()
Return true if there is a new value at left slider.

Returns True if new value available

bool **new_slider_right**()
Return true if there is a new value at right slider.

Returns True if new value avaible

bool **new_joystick_direction**(void)
Return true if there is a new direction at joystick.

Returns True if new value available

uint8_t **left_slider_pos**()
Return left slider position.

Returns uint8_t left slider position

uint8_t **right_slider_pos**()
Return right slider position.

Returns uint8_t right slider position

uint8_t **joystick_direction**()
Return joystick direction.

Returns uint8_t joystick direction

bool **new_event**(bool left_btn_pressed, bool right_button_pressed)
Return true if something has changed from input.

Parameters

- **left_btn_pressed** – Bool representing left touch button
- **right_button_pressed** – Bool representing right touch button

Returns True if there is a change in the data from the multifunction USB board

void **write_event_data**(uint8_t *buffer, bool left_btn_pressed, bool right_btn_pressed)
Write event data to tx buffer, used for sending over CAN.

Parameters

- **buffer** – Data to transmit
- **left_btn_pressed** – Bool representing left touch button
- **right_btn_pressed** – Bool representing right touch button

struct **Joystick_t**
#include <js_slider.h> Struct holding current joystick values

struct **Slider_t**
#include <js_slider.h> Struct holding current slider values

SRAM

The SRAM library includes functions for interacting with the SRAM in Node 1. To ensure that this SRAM works as expected, the *SRAM_test* function can be run.

API Reference

group **sram**
SRAM API for node 1.

Defines

EXRAM_START
Start address for SRAM in ATMega162 external memory

Functions

void **sram_setup**()
Setup of external SRAM for ATMega162.

void **sram_write**(uint16_t address, uint8_t data)
Write byte to SRAM address.

Parameters

- **address** – SRAM address
- **data** – Byte to write

uint8_t **sram_read**(uint16_t address)
Read byte from SRAM address.

Parameters

- **address** – SRAM address

Returns uint8_t received byte

void **sram_test**()
Stress test of the SRAM.

1.2 Documentation for Node 2

Node 2 contains an ATSAM3X8E microcontroller on an Arduino Due board. This Node controls the motor, servo and solenoid in the ping-pong game.

This Node receives commands from Node 1 over can, and reacts on these commands.

1.2.1 Modules

Motor

This is the Motor control API for Node 2

The motor at the ping pong board is controlled via an analog signal, controlled via the DAC at Node 2.

API Reference

group **motor**
Motor control API for node 2.

Defines

MJ1_NOT_OE
!OE pin. Encoder output enable

MJ1_NOT_RST
!RST pin. Encoder reset

MJ1_SEL
SEL pin. Encoder select low/high byte

MJ1_EN
EN pin. Motor enable

MJ1_DIR
DIR pin. Motor direction

MJ2_PINS
Mask used to select MJ2 pins on the motorbox. PC1-PC8 on the ATSAM3x8E.

Enums

enum **motor_direction**
Used to set motor direction.

Values:

enumerator **MOTOR_RIGHT**

enumerator **MOTOR_LEFT**

Functions

void **motor_setup**()
Setup of shield interface with external motorbox, i.e. DAC, MJ1 and MJ2 pins.

void **set_motor_direction**(uint8_t direction)
Set the motor direction.

Parameters

- **direction** – Number of type motor_direction

void **set_motor_speed**(uint8_t speed)
Set the motor speed.

Parameters

- **speed** – Speed in range 0-100

int16_t **encoder_read**()
Read the motorbox encoder.

Returns Raw encoder value

uint8_t **encoder_get_position**()
Get the motor position. 0 is right wall, 100 is left wall.

Returns Position in range 0-100

void **encoder_reset**()
Reset motorbox encoder.

void **encoder_calibrate**()
Calibrate motorbox encoder.

Variables

uint16_t **encoder_max**
Max encoder value. Assigned in calibration function.

PID

The PID library contains an implementation of a digital PID controller. This digital PID controller is implemented with the motor control API (link).

API Reference

group **pid**
PID controller API for node 2.

Defines

GAIN_SCALING
PID parameter scaling factor. Used to get higher resolution parameters

MAX_U
PID output upper bound. Equal to 100*GAIN_SCALING

MAX_UI
Integral term output upper bound. Serves as anti-windup.

ERROR_TOLERANCE
Error tolerance. Errors within this parameter are mapped to 0 to avoid oscillations caused by the motor sticking

PID_SAMPLING_INTERVAL_MS
Sampling interval of the controller

Typedefs

typedef struct *pid_t* **PID_DATA**
Struct containing PID parameters.

Functions

void **pid_tune**(struct *pid_t* *pid, float K, uint8_t Ti, uint8_t Td, uint8_t N)
Calculates and changes the controller parameters in the passed struct.

Parameters

- **pid** – Pointer to struct of type *pid_t* that contains the pid parameters
- **K** – Controller gain
- **Ti** – Integral time constant
- **Td** – Derivative time constant
- **N** – Derivative filter constant

int16_t **pid_controller**(struct *pid_t* *pid, uint8_t r, uint8_t y)
Calculates PID controller output.

Parameters

- **pid** – Pointer to struct of type *pid_t* that contains the pid parameters
- **r** – Setpoint
- **y** – Process value

Returns Controller output

struct **pid_t**
#include <pid.h> Struct containing PID parameters.

Console Node 2

The console library contains drivers for different peripherals in Node 2. The functions in this library can be used for setting up and interacting with the ping-pong board.

API Reference

group **console**
Drivers for handling IR sensor, solenoid actuation and incoming multifunction USB board data.

Defines

IR_PIN
Infrared sensor pin

SOLO_PIN
Solenoid relay pin

SOLO_DELAY_MS
Solenoid actuation delay

Typedefs

typedef struct *console_data_t* **CONSOLE_DATA**
Struct to keep current readings of joystick and buttons from node 1

typedef struct *game_data_t* **GAME_DATA**
Struct to keep current game status

Enums

enum **JS_direction**
Joystick direction enum

Values:

enumerator **DEFAULT**

enumerator **RIGHT**

enumerator **LEFT**

enumerator **UP**

enumerator **DOWN**

Functions

void **JS_Handler**(uint8_t direction)
Actuates servo and solenoid based on Joystick readings from node 1.

Parameters

- **direction** – Joystick direction from node 1, type JS_direction.

void **ir_setup**()
Sets up the infrared sensor pin and interrupts.

void **solonoid_setup**()
Sets up the solenoid relay pins.

Variables

uint8_t **currentServoPos**
Current servo position in the range 0-180

struct **console_data_t**
#include <*console_lib.h*> Struct to keep current readings of joystick and buttons from node 1

struct **game_data_t**
#include <*console_lib.h*> Struct to keep current game status

PWM

The PWM library contains PWM control functions for interacting with the servo at the ping-pong board.

API Reference

group **pwm**
PWM library API for node 2.

Functions

int **pwm_setup**()
Set up PWM on pin PC18, PWM channel 6H.

Returns int 0 if succesfull, negative value on error

void **pwm_set_duty_cycle**(uint8_t duty_cycle, uint8_t channel)
Set duty cycle for pwm.

Parameters

- **duty_cycle** – Unsigned number between 0 and 100 representing percentage of duty-cycle to set
- **channel** – PWM channel

void **servo_set_pos**(uint8_t pos, uint8_t channel)
Set servo position.

Parameters

- **pos** – Unsigned number between 0 and 180 representing servo position
- **channel** – PWM channel

INDEX

N

O

P

R

S

T

U

W