Comminication Protocols of the Walker Controller — Core — Segments

Mathias Kussinger

March 1, 2015

1 Overview

The Walker consists of multiple components that have their own processor and run their own code. These components have to communicate with each other.
The components are

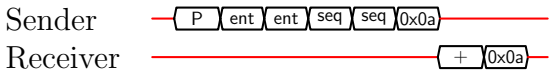
- The Core module. It coordinates the movement and is the master of the communication with the Segment modules on the Walker.
- The Segment modules. They control the movement of two legs. They talk to the core and the Dynamixel servos.
- The Control software on the PC. It is used to control the Walker, store fault messages and visualize log data.

2 Controller - Core

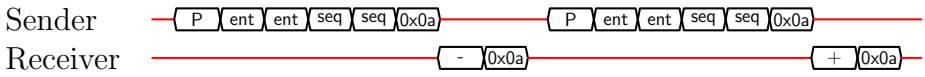
– missing yet –

2.1 Sending and Acknowledge

There is no master for the communication between the controller and the core module. The communication is full duplex. Each message has to be acknowledged. A message of type + is a positive acknowledge, a message of type − is a negative acknowledge and signals that the sender has to send the message again.
Positive acknowledge:



Negative acknowledge and re-sending:



3 Core - Segments

RxD	1-4	PJ0/RXD3 [15]	in	Serial data reception line. RXSeln selects the active sender. Pull up!
TxD	1-4	PJ1/TXD3 [14]	out	Serial data transmission line.
(RXAct)	1-4	PE4/INT4 [2]	in	Not used. Receiver expect data. Multiplexed OC.
Sync	1-4	PA3 [25]	out	Segment synchronization.
TXSel	1	PC0 [37]	out	Transmit select for segment 1.
	2	PC1 [36]	out	Transmit select for segment 2.
	3	PC2 [35]	out	Transmit select for segment 3.
	4	PC3 [34]	out	Transmit select for segment 4.
TXAck	1-4	PE5/INT5 [3]	in	Transmission acknowledge. Wired and with OC.
RXSel	1	PC4 [33]	out	Receive select for segment 1.
	2	PC5 [32]	out	Receive select for segment 2.
	3	PC6 [31]	out	Receive select for segment 3.
	4	PC7 [30]	out	Receive select for segment 4.
TXRqH	1	PK0/PCINT16 [A8]	in	High priority request segment 1.
	2	PK1/PCINT17 [A9]	in	High priority request segment 2.
	3	PK2/PCINT18 [A10]	in	High priority request segment 3.
	4	PK3/PCINT19 [A11]	in	High priority request segment 4.
TXRqL	1	PK4/PCINT20 [A12]	in	Low priority request segment 1.
	2	PK5/PCINT21 [A13]	in	Low priority request segment 2.
	3	PK6/PCINT22 [A14]	in	Low priority request segment 3.
	4	PK7/PCINT23 [A15]	in	Low priority request segment 4.
reset?			out	Controll the reset lines of the segments.

Table 1: Communication signals of the Core module. The port names and pin numbers are for an Arduino Mega 2560.

The communication between the Core and the Segments uses the standard serial interfaces supported by multiple ports for signaling and handshake.

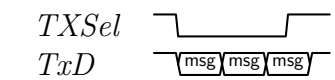
TxD	PJ0/RXD3 [15]	in	Serial data transmission line of the Core.
RxD	PJ1/TXD3 [14]	out	Serial data reception line of the Core. The RXSel input enables the line.
RXAct	PA0 [22]	out	Transmission hand-shake. Multiplexed OC.
TXAck	PA3 [25]	out	Reception acknowlege. Wired and with OC.
Sync	PD0/INT0 [21]	in	Segment synchronization.
TXSel	PE5/INT5 [3]	in	Transmit select for the segment.
RXSel	PE4/INT4 [2]	in	Receive select for the segment.
TXRqH	PA1 [23]	out	High priority request from the segment.
TXRqL	PA2 [24]	out	Low priority request from the segment.

Table 2: Communication signals of a Segment module. The port names and pin numbers are for an Arduino Mega 2560.

It is optimized to reduce the communication overhead. The Core is the communication master.
All segments communicate with the core through a pair of serial lines. The communication is fully bidirectional.
The core uses the *TXSel* lines to select the receiver of the transmitted data. It may select multiple segments at a time. There is no reception handshake. To receive, the core selects one segment using the *RXSel* lines. It may only select one sender at a time. The selection connects the sender's serial send port to *RxD* and connects the transmission signaling line to the *RXAct* port. The sender uses *RXAct* to signal a transmission. There is no transmission handshake. The sender may only transmit one message at a time.
The segments use the *TXRqH* and *TXRqL* lines to signal the core that they have high or low priority messages ready to transmit.
The table 1 shows the communication signals and their pin names of the core module. The numbers in brackets are the port numbers of an Arduino Mega 2560. The table 2 shows the signals and the pin names of a segment module.

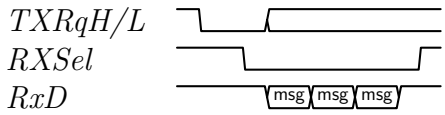
3.1 Core Transmitting to Segment

The core sets one or multiple of the *TXSel* lines to 0 and starts sending on *TxD*.



3.2 Core Receiving from Segment

The segment signals the request to send by pulling *TXRqH* or *TXRqL* to 0. When the core is ready to receive data it pulls the *RXSel* of the segment 0. This also connects the *RXAct* and *RxD* signalsof the selected segment to the core. When the segment starts the transission it pulls *RXAct* to 0 and sends the message on *RxD*. After the transmission is completed, *RXAct* goes to 1. Thecore de-selects the segment by switching *RXSel* to 1.



Signal	Core	Segment
TXRqH4:1		Signals the core that the segment has a high priority message ready to send. The Core has to initiate the transmission. The value may change after RXSel has become low.
TXRqL4:1		Same as TXRqH but for low priority messages.
RXSel4:1	The Core selects one Segment for transmission.	On a **falling edge** the Segment prepares the message for transmission. It starts the transmission. On a **rising edge** the Segment aborts any active transmission.
RXAct		not used
RxD	The Core receives the serial data stream on this input. The format is 9N1, bit8 set signals the message start.	The Segments sends the serial data stream through this output.

4 Messages

4.1 Format

4.2 Entities

4.3 Message Types

4.3.1 C Command Message

4.3.2 E Entity Request Message

A request sent by a Segment controller to the Core to get it's entity id set.

4.3.3 e Entity Set Message Message

A message sent by the Core to a segment to set it's entity id.

4.3.4 f Fault Notification

4.3.5 L Rog Data Request

4.3.6 l Log Data

4.3.7 P Ping Request

4.3.8 p Ping Response

4.3.9 s State Change Request

Request a state change.

message type	char	1	
source entity id	uint16	2	
state level	uint8	1	

4.3.10 s State

The state of an entity and optional status flags.

message type	char	1	
source entity id	uint16	2	
state level	uint8	1	
status flags (optional)	uint8	0-n	

4.3.11 t Link Test

4.3.12 V Version Request

4.3.13 v Version response