|  |  |
| --- | --- |
| **Documentation Title** | Roots |
| **Version** | 1.0 |
| **Authors** | Kaviarashe and Kaviya |

**Functionalities of A1 with A3:**

* A2 or A3 informs A1 that run-timer should be ON/OFF
* A1 receives the driver motor settings from A3
* A3 sends the Pot calibration information to A1
* A1 gets the Pot value for A3
* A1 gets the Pedal level from A3
* A1 receives acknowledgement from A2 and A3 when board starts up to make sure communication is working
* A1 requests driver motor information from A3
* A1 sends updated driver motor settings to A3
* A1 sends an alarm to inform A3 CAN communication error to A2 to disable all the functions

**1.** **A2 or A3 informs A1 that run-timer should be ON/OFF**

**File name:** intercommunication.c (A2\_RB\_800\_M)

void send\_runtime\_timer (*uint8\_t* status)

{

runtimer\_on = status;

*uint8\_t* message [1];

message [0] = status;

transmit\_CAN (RUNTIMER\_CAN, 0, 1, message);

}

**2. A1 receives the driver motor settings from A3**

**File name:** intercommunication.c (A3\_RB\_800\_DM)

void send\_motor\_settings(void)

{

//Send all motor settings

*uint8\_t* message [8];

*uint8\_t* i = 0;

message[i++] = DM\_SETTINGS\_1;

message[i++] = acceleration;

message[i++] = deceleration;

message[i++] = brake\_delay;

message[i++] = reverse\_speed;

transmit\_CAN (DM\_SETTINGS\_CAN, 0, 5, message);

//\_delay\_ms (10); //IVA002: Removed unnecessary delay

i = 0;

message[i++] = DM\_SETTINGS\_2;

message[i++] = slow\_speed;

message[i++] = fast\_speed;

message[i++] = rated\_current;

message[i++] = max\_current;

message[i++] = full\_overload\_time;

transmit\_CAN (DM\_SETTINGS\_CAN, 0, 6, message);

//\_delay\_ms (10); //IVA002: Removed unecessary delay

}

**3. A3 sends the Pot calibration information to A1**

**File name:** intercommunication.c (A1\_RB\_800)

void send\_pot\_calibrating(void)

{

*uint8\_t* message[1];

message[0] = pot\_calibrating;

transmit\_CAN (POT\_CALIBRATION\_CAN, 0, 1, message);

*\_delay\_ms* (10);

}

**4. A1 gets Pot value from A3**

**File name:** intercommunication.c (A1\_RB\_800)

void Request\_pot\_value(void)

{

*uint8\_t* message [1];

message [0] = 0; //IVA001: Send a valid value

//message [0] = pot\_calibrating;

transmit\_CAN (REQUEST\_POT\_VALUE\_CAN, 0, 1, message);

*\_delay\_ms* (10);

}

**5. A3 sends the Pedal level to A1**

**File name:** intercommunication.c (A3\_RB\_800\_DM)

void filter\_pedal\_state(bool chkAck, *uint8\_t* st)

{

if (chkAck && pedal\_ack\_pending)

{

pedal\_state\_q = st;

pedal\_upd\_pending = true;

return;

}

*uint8\_t* message[2];

message[0] = PEDAL\_STATE;

message[1] = st;

pedal\_ack\_pending = true;

transmit\_CAN(ACCEL\_PDL\_CAN, 0, 2, message);

}

//Get pedal level from the A3

case ACCEL\_PDL\_CAN:

if (data[0] == PEDAL\_LEVEL)

{

pedal\_level = data[1] << 8;

pedal\_level += data[2];

A3\_can\_functioning\_timer = 0;

}

break;

**6. A1 receives acknowledgement from A2 and A3 when board starts up to make sure communication is working**

**File name:** intercommunication.c (A3\_RB\_800\_DM)

void send\_CAN\_ACK (void)

{

*uint8\_t* message [1];

message[0] = A3\_CAN;

transmit\_CAN (ACK\_CAN, 0, 1, message);

*\_delay\_ms* (10);

}

**7. A1 requests the driver motor information from A3**

**File name:** intercommunication.c (A1\_RB\_800)

void request\_statistics(void)

{

*uint8\_t* message[1];

message[0] = REQUEST\_STATISTICS;

transmit\_CAN (REQUEST\_STATISTICS, 0, 1, message);

*\_delay\_ms* (10);

}

**8. A1 sends updated driver motor settings to A3**

**File name:** intercommunication.c (A1\_RB\_800)

void update\_DM\_settings(void)

{

//Send all motor settings

*uint8\_t* message[8];

message[0] = UPDATE\_DM\_SETTINGS;

*uint8\_t* i = 1;

message[i++] = DM\_SETTINGS\_1;

message[i++] = acceleration;

message[i++] = deceleration;

message[i++] = brake\_delay;

message[i++] = reverse\_speed;

transmit\_CAN (DM\_SETTINGS\_CAN, 0, 6, message);

*\_delay\_ms* (10);

i = 1;

message[i++] = DM\_SETTINGS\_2;

message[i++] = slow\_speed;

message[i++] = fast\_speed;

message[i++] = current\_rating;

message[i++] = max\_current;

message[i++] = overload\_time;

transmit\_CAN (DM\_SETTINGS\_CAN, 0, 7, message);

*\_delay\_ms* (10);

}

**9. A1 sends an alarm to inform A3 Can communication error to A2 to disable all the functions**

**File name:** intercommunication.c (A1\_RB\_800)

//Send an alarm to inform other boards

void send\_alarm (*uint8\_t* error)

{

*uint8\_t* message[1];

message[0] = error;

transmit\_CAN (ALARM\_CAN, 0, 1, message);

*\_delay\_ms* (10);

}

//Send an alarm to inform A3 CAN communication error to A2 board to disable all functions

void send\_emergency\_off(void)

{

*uint8\_t* message[1];

message[0] = 0; //A1\_CAN;//IVA001: Send a valid value

transmit\_CAN (EMERGENCY\_OFF\_CAN, 0, 1, message);

*\_delay\_ms* (10);

}

* **Transmit CAN function (CAN.c in all A1, A2, and A3)**

void transmit\_CAN\_(*uint8\_t* dataID, *uint8\_t* remote\_transmission\_request,

*uint8\_t* data\_length, *uint8\_t* data[])

{

*uint8\_t* retry = 1;

jump\_retry:

Check\_flag = 1;

CAN\_CHIP\_SELECT\_LOW;

write\_byte\_SPI(LOAD\_TX\_BUFFER\_CAN | LOAD\_BUFFER\_0);

/\*Write CAN bus communication to SPI\*/

//Arrange explicit parameters to the register map of the MCP25625

//IVA002: J1939 Implementation

// 0x10FFxxA1

// |<--- std --------------------->|<-------------- extended -------------------------->|

// 10 ..................03.......00 .... .......................

// 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

// 1 0 0 0 0 1 1 1 1 1 1 1 1 x x x x x x x x 1 0 1 0 0 0 0 1

//Bits 10-3 of standard identifier

write\_byte\_SPI(0b10000111);

//Bits 2-0 of standard identifier

//& extended identifier enable bit & bits 17-16 of extended identifier

//Extended identifier is used

write\_byte\_SPI(0b11101011);

//Bits 15-8 of extended identifier

write\_byte\_SPI(dataID);

//Bits 7-0 of extended identifier

write\_byte\_SPI(0xA1);

//If remote transmission request is required

//set RTR bit as 1, and set data length to 0

if (remote\_transmission\_request)

{

*uint8\_t* rtr\_dlc = 0x40;

write\_byte\_SPI(rtr\_dlc);

}

//Else set data length and write the data bytes

else

{

*uint8\_t* rtr\_dlc = data\_length;

write\_byte\_SPI(rtr\_dlc);

for (*uint8\_t* i = 0; i < data\_length; i++)

{

write\_byte\_SPI(data[i]);

}

}

CAN\_CHIP\_SELECT\_HIGH;

//Send out data through CAN

CAN\_CHIP\_SELECT\_LOW;

write\_byte\_SPI(WRITE\_CAN);

write\_byte\_SPI(TXB0CTRL);

if (Check\_flag)

{

write\_byte\_SPI(0x08);

}

else

{

if (retry == 1)

{

retry--;

goto jump\_retry;

}

}

CAN\_CHIP\_SELECT\_HIGH;

Check\_flag = 0;

}

**Functionalities of A2 with A3.:**

* Resets A3 if instructed
* Turn off all the motors if Top event occurs

**1. Resets A3 if instructed**

**File name:** intercommunication.c (A2\_RB\_800\_M)

case RESET\_DEVICE\_CAN\_A3:

if (reset\_timer >= 10)

{

wdt\_enable (WDTO\_15MS);

REBOOT ();

}

break;

**2. Turn off all the motors if Top event occurs**

**File name:** intercommunication.c (A2\_RB\_800\_M)

case TOP\_CAN:

if (data[0] == A3\_CAN)

{

message[0] = A3\_TOP\_EVENT;

message[1] = 0;

log\_event(message);

}

//Disable all 9201 devices

PORTF\_OUTSET = (1 << DISABLE\_9201);

//Disable all 8701 devices

PORTE\_OUTCLR = (1 << ENABLE\_8701);

water\_pump\_set (OFF, OFF);

Solenoid\_set (OFF);

detergent\_pump\_set (OFF, OFF);

send\_top\_flag = FALSE;

emergency\_off = TRUE;

break;

**Driver Motor:**

|  |  |  |
| --- | --- | --- |
|  | **Index** | **Sub Index** |
| Velocity Acceleration | 0x6048 | 0x01 (DS), 0x02 (DT) |
| Velocity Deceleration | 0x6049 | 0x01 (DS), 0x02 (DT) |
| Velocity Quick stop (Brake delay) | 0x604A | 0x01 (DS), 0x02 (DT) |
| Rated Current | 0x6075 | 0x00 |
| Overload percentage – Axis 0 | 0x2311 | 0x01 |
| Rest time – Axis 0 | 0x2311 | 0x02 |
| Remaining time – Axis 0 | 0x2311 | 0x03 |

**Steering Controller:**

|  |  |  |
| --- | --- | --- |
|  | **Index** | **Sub Index** |
| Profile Acceleration | 0x6083 | 0x00 |
| Profile Deceleration | 0x6084 | 0x00 |
| Rated Current | 0x6075 | 0x00 |
| Overload percentage – Axis 0 | 0x2311 | 0x01 |
| Rest time – Axis 0 | 0x2311 | 0x02 |
| Remaining time – Axis 0 | 0x2311 | 0x03 |