mikroBUS™

The mikroBUS is a standard connector with a predetermined pin assignment to allow the creation of portable add-on hardware that can be used on several different platforms requiring only adjustments on teh software side. It is currently implemented by the Hummingboard Gate and the Clearfog Pro.

PIN Header

Pin to GPIO mapping on Clearfog Pro:

mikroBUS	**Clearfog Pro**
AN	MCP3021 (via i2c)
RST	MPP29
CS	MPP43
SCK	MPP57
MISO	MPP58
MOSI	MPP56
PWM	MPP54
INT	MPP22
RX	MPP24
TX	MPP25
SCL	MPP26
SDA	MPP27

Enable pin function / clickboard in DeviceTree

To actually use the mikroBUS pins with a userspace application or from a kernel driver, their specific function has to be enabled in the kernel DeviceTree file. The DTS files to be modified are available in the kernel tree, under **arch/arm/boot/dts/** For the Clearfog Pro it is called **armada-388-clearfog.dts**. Any modifications belong inside the main {} block, the one that contains the *model* property, i.e. *model = “SolidRun Clearfog A1”;*

SPI

This is an example for a device attached to the SPI Bus:

```
TBD
```

I2C

i2c is already preconfigured to provide userspace access via **/dev/i2c-1** on the clearfog. The userspace i2c-tools can be used, e.g. *i2cdetect*

the i2c userspace interface, be it /dev/isc-* or i2c-tools, expect a **7-bit** device address! This is because the read/write bit is not considered part of the address by the driver.
Keep away from device address **0xA0** and **0xA1** (7-bit: **0x50**). these are reserved for the SFP module!

UART

TBD

GPIO

TBD

(Re-)Compile DeviceTree

Generically speaking, the raw .dts files can be compiled into .dtb using the device-tree-compiler *dtc*. The most straightforward way to invoke it is by configuring the linux kernel tree, and then running

```
make dtbs
```

The kernel-tree needs to be configured first. Please refer to A38X Kernel for generic instructions, or consult the distro documentation

DTB files can also be de- and recompiled. Consult the manpage of *dtc* for additional information.

Examples

EEPROM 3 click

This chip listens on multiple i2c addresses to facilitate access to all the eeprom memory addresses. The AT24CM02 datasheet explains this in detail!

This is example code for writing the value 0x2A to address 0x0000

```
/*
 * Copyright 2016 Josua Mayer <josua.mayer97@gmail.com>
 * Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
IN THE SOFTWARE.
 */
#include <stdio.h>
#include <stdint.h>
#include <linux/i2c-dev.h>
#include <errno.h>

/*
 * calculates 7-bit address (excluding r/w bit) (because i2c-dev expects it
that way)
 * hwaddr is either 0 or 1, depending on ADD jumper value
 * datawordaddr is full 18-bit memory address on the eeprom
 */
uint8_t calculate_device_address(uint8_t hwaddr, uint32_t datawordaddr) {
    // device identifier on bits 6-3
    uint8_t addr = 0b01010000;

    // hardware address on bit 2
    addr += hwaddr << 2;

    // 2 most significant bits of data word address on bits 1-0
    addr += (datawordaddr & 0b000000110000000000000000) >> 16;

    return addr;
}

int main(int argc, char *argv[]) {
    FILE *fp;
    uint8_t devaddr;
    uint32_t dataaddr;
    uint8_t data;
    int r;

    // open i2c device
    fp = fopen("/dev/i2c-1", "r+");
    if(fp == 0) {
        printf("Failed to open i2c device!\n");
        return -1;
    }

    // set slave device address (including data word address msb)
    dataaddr = 0x0000;
    devaddr = calculate_device_address(1, dataaddr);
    r = ioctl(fileno(fp), I2C_SLAVE, devaddr);
    if(r == -1) {
        printf("Failed to set i2c slave address:%i!\n", errno);
        return -1;
    }

    // construct write message (2 bytes addr + 1 byte data)
    uint8_t message[3] = { 0x00 };
    // first 2 bytes: address
    message[0] = ((uint8_t *)&dataaddr)[0];
    message[1] = ((uint8_t *)&dataaddr)[1];
    // last byte is data
    message[2] = 0x2A;
    printf("DEBUG: message=0x%02X%02X%02X\n", message[0], message[1],
message[2]);

    // write
    r = fwrite(&message, 3, 1, fp);
    if(r == -1 || r != 1) {
        printf("Failed to write data to address 0x%X:%i!\n", dataaddr,
errno);
        return -1;
    }

    // Done

    fclose(fp);
    return 0;
}
```

Because reading from a specific address is harder, there is no sample available yet.