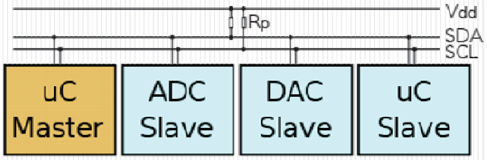
02-2 I2C

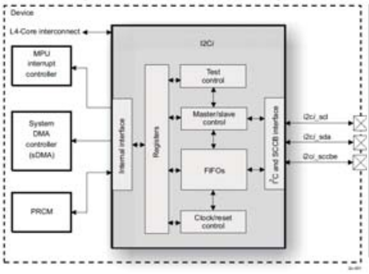
Interfacing with sensors over a serial bus

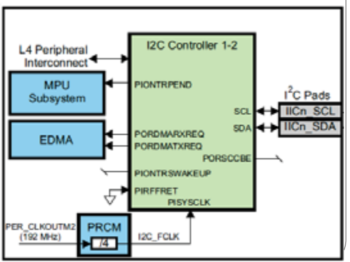


I^{2}C

- "two-wire interface" standard
- Used to attach low-speed peripherals to embedded systems
- The Bone has two I^{2}C controllers (Section 21 of TRM)

Figure 17-1. HS I^{2}C Controllers Overview Block Diagram





Hardware - Bone

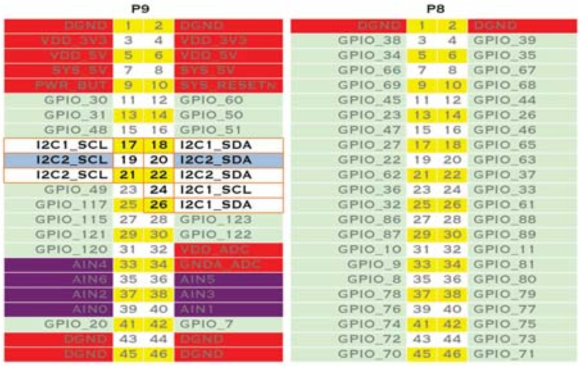
- You can see which ones are configured at boot time

```
beagle$ dmesg | grep i2c
[    0.156139] omap_i2c 44e0b000.i2c: bus 0 rev0.11 at 400 kHz
[    0.157673] input: tps65217_pwr_but as
/devices/ocp.2/44e0b000.i2c/i2c-0/0-0024/input/input0
[    0.169206] omap_i2c 44e0b000.i2c: unable to select pin group
[    0.170089] omap_i2c 4819c000.i2c: bus 1 rev0.11 at 100 kHz
[    0.172685] omap_i2c 4819c000.i2c: unable to select pin group
[    0.762708] i2c /dev entries driver
```

Two buses each running at different speeds

Time in seconds

i2c - bone



P9				P8			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	GPIO_38	3	4	GPIO_39
VDD_5V	5	6	VDD_5V	GPIO_34	5	6	GPIO_35
SYS_5V	7	8	SYS_5V	GPIO_66	7	8	GPIO_67
PWR_BUT	9	10	SYS_RESETN	GPIO_69	9	10	GPIO_68
GPIO_30	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
GPIO_31	13	14	GPIO_50	GPIO_23	13	14	GPIO_26
GPIO_48	15	16	GPIO_51	GPIO_47	15	16	GPIO_46
I2C1_SCL	17	18	I2C1_SDA	GPIO_27	17	18	GPIO_65
I2C2_SCL	19	20	I2C2_SDA	GPIO_22	19	20	GPIO_63
I2C2_SCL	21	22	I2C2_SDA	GPIO_62	21	22	GPIO_37
GPIO_49	23	24	I2C1_SCL	GPIO_36	23	24	GPIO_33
GPIO_117	25	26	I2C1_SDA	GPIO_32	25	26	GPIO_61
GPIO_115	27	28	GPIO_123	GPIO_86	27	28	GPIO_88
GPIO_121	29	30	GPIO_122	GPIO_87	29	30	GPIO_89
GPIO_120	31	32	VDD_ADC	GPIO_10	31	32	GPIO_11
AIN4	33	34	GNDA_ADC	GPIO_9	33	34	GPIO_81
AIN6	35	36	AIN5	GPIO_8	35	36	GPIO_80
AIN2	37	38	AIN3	GPIO_78	37	38	GPIO_79
AIN0	39	40	AIN1	GPIO_76	39	40	GPIO_77
GPIO_20	41	42	GPIO_7	GPIO_74	41	42	GPIO_75
DGND	43	44	DGND	GPIO_72	43	44	GPIO_73
DGND	45	46	DGND	GPIO_70	45	46	GPIO_71

The first I2C bus is utilized for reading EEPROMS on cape add-on boards and can't be used for other digital I/O operations without interfering with that function, but you can still use it to add other I2C devices at available addresses.

The second I2C bus is available for you to configure and use.

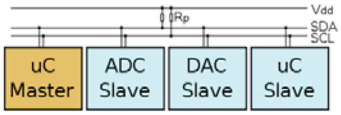
Hardware – TMP101

- Goal: Interface to a TMP101 temp sensor

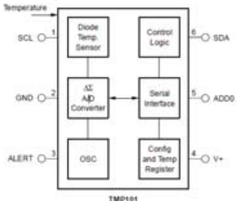
Parameter Name	Value
Typical Accuracy (°)	±2.0°C from −25°C to +85°C (max) ±3.0°C from −55°C to +125°C (max)
Supply Current (µA)	45µA, 0.1µA Standby
Resolution	9- to 12-bits,
Operating Voltage Range (V)	2.7V to 5.5V
Device Description	Serial Output Temp Sensor

http://www.ti.com/lit/gpn/tmp101

2-wire bus



- The two wires are
  - Serial Clock (SCL), is an input to the TMP101 and is used to clock data into and out of the TMP101.
  - Serial Data (SDA), is bidirectional and carries the data to and from the TMP101.
- The only other two pins on the TMP101 that you need to use are the Power Supply (Vdd) and Ground.



Software - bone

- See what's on a bus with **i2cdetect**

```
beagle$ i2cdetect -y -r 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- 48 49 -- -- -- -- -- --
50: -- -- -- -- UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- --
```

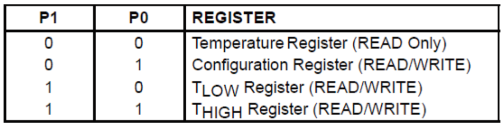
I have 2, TMP101's and an LED matrix.

- The TMP101's are at **1001 000** and **1001 001**
- Convert to hex **0x48** and **0x49**

Registers

- Each TMP101 has four registers

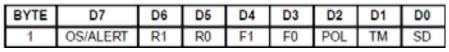
Table 2. Pointer Addresses of the TMP100 and TMP101 Registers



P1	P0	REGISTER
0	0	Temperature Register (READ Only)
0	1	Configuration Register (READ/WRITE)
1	0	T_{LOW} Register (READ/WRITE)
1	1	T_{HIGH} Register (READ/WRITE)

- Read with **$ i2cget -y 1 0x48 00**
- **0x18** which is 24C or 75.2F

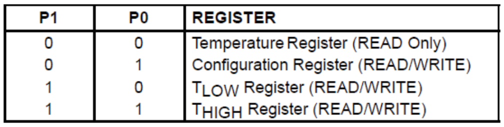
Table 6. Configuration Register Format



BYTE	D7	D6	D5	D4	D3	D2	D1	D0
1	OS/ALERT	R1	R0	F1	F0	POL	TM	SD

Registers

Table 2. Pointer Addresses of the TMP100 and TMP101 Registers



P1	P0	REGISTER
0	0	Temperature Register (READ Only)
0	1	Configuration Register (READ/WRITE)
1	0	T_{LOW} Register (READ/WRITE)
1	1	T_{HIGH} Register (READ/WRITE)

- Read with **$ i2cget -y 1 0x48 01**
- **0x80** which is **1000 0000**

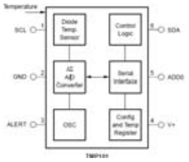
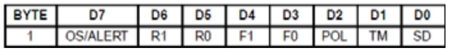


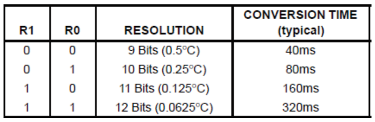
Table 6. Configuration Register Format



BYTE	D7	D6	D5	D4	D3	D2	D1	D0
1	OS/ALERT	R1	R0	F1	F0	POL	TM	SD

SD – Shutdown Mode
TM - Thermostat Mode
POL-Polarity
F1/F0 – Fault Queue
R1/R0 – Converter Resolution
OS – OS/Alert

Table 8. Resolution of the TMP100 and TMP101



R1	R0	RESOLUTION	CONVERSION TIME (typical)
0	0	9 Bits (0.5°C)	40ms
0	1	10 Bits (0.25°C)	80ms
1	0	11 Bits (0.125°C)	160ms
1	1	12 Bits (0.0625°C)	320ms

I^{2}C via C – myi2cget.c

```
int main(int argc, char *argv[]) {
        char *end;
        int res, i2cbus, address, size, file;
        int daddress;
        char filename[20];

        /* handle (optional) flags first */
        if(argc < 3) {
                fprintf(stderr,
                        "Usage:  %s <i2c-bus> <i2c-address> <register>\n",
                        argv[0]);
                exit(1);
        }
        i2cbus   = atoi(argv[1]);
        address  = atoi(argv[2]);
        daddress = atoi(argv[3]);
        size = I2C_SMBUS_BYTE;
```

I^{2}C via C

```
sprintf(filename, "/dev/i2c-%d", i2cbus);
file = open(filename, O_RDWR);
if (file < 0) {
    if (errno == ENOENT) {
        fprintf(stderr, "Error: Could not open file "
        "/dev/i2c-%d: %s\n", i2cbus, strerror(ENOENT));
    } else {
        fprintf(stderr, "Error: Could not open file "
            "`%s': %s\n", filename, strerror(errno));
        if (errno == EACCES)
            fprintf(stderr, "Run as root?\n");
    }
    exit(1);
}
```

I^{2}C via C

```
if (ioctl(file, I2C_SLAVE, address) < 0) {
    fprintf(stderr,
            "Error: Could not set address to 0x%02x: %s\n",
                address, strerror(errno));
    return -errno;
}

res = i2c_smbus_write_byte(file, daddress);
if (res < 0) {
    fprintf(stderr, "Warning - write failed, filename=%s,
            daddress=%d\n", filename, daddress);
}
res = i2c_smbus_read_byte_data(file, daddress);
close(file);
```

myi2ctest

- See **`exercises/i2c/matrixLEDi2c.c`** for an example that controls an LED grid
- See **`exercises/realtime/boneServer.js`** for an example that uses **`i2cdump`** and **`i2cset`** to control an LED grid
- See **`exercises/i2c/i2c-tools-3.1.0`** for source code for ic2 tools