Creating, debugging and operating a custom I2C peripheral.
Disclaimer on Third Parties

All product and company names are trademarks™ or registered® trademarks of their respective holders. Use of them does not imply any affiliation with or endorsement by them.
Overview

• Typical applications
• Introduction to the I2C bus
• Custom slaves - why and how
• Master
• Debugging methodology and tools
• Example: steering a 4WD drone.
• Ideas for advanced bus configurations
• Recap
• Q/A
Typical Applications

- Interfacing with relatively slow peripherals. Ex: sensors, mechanical actuators.
- Controlling “fast” peripherals, that use other channels for exchanging data. Ex: codecs.
- In a PC, Linux usually interacts over I2C with:
  - temperature and battery voltage meters;
  - fan speed controllers;
  - audio codecs.
- Multiple bus controllers, each at different speeds.
Overview

• Typical applications
• Introduction to the I2C bus
• Custom slaves - why and how
• Master
• Debugging methodology and tools
• Example: steering a 4WD drone.
• Ideas for advanced bus configurations
• Recap
• Q/A
Introduction to the I2C Bus - Part 1

- Only 2 lines: Serial CLOCK and Serial DATA (plus ground).
- 4 speeds: 100kHz, 400kHz, 1MHz, 3.2MHz.
- Typically, 1 master device and 1 or more slaves.
- Communications are always initiated by a master device.
- Multiple masters can co-exist on the same bus (multi-master).
- Open-Drain: both SDA and SCL need pull-up resistors.
“Clock Stretching”
- The master controls SCL, but a slave can hold it down (because open drain), if it needs to adjust the speed.
- The master must check for this scenario.
- A slave can get stuck and jam the bus: need for reset lines from the master to the slave.

- Typically 7-bit addressing, but also 10 bit is supported.
- Logical protocol: actual voltage levels are not specified and depend on individual implementations.
  Ex: 1.8V / 3.3V / 5.0V
Example of bus configuration
Protocol (simplified)

- 2 messages: read / write
- Start / Stop bit - represented as “[“ and “]”
- Address: 7 or 10 bits
- R/W bit: R = 1 / W = 0
- Byte on the bus: (Address << 1 | R/W)
- Registers

Ex:
Write - [ address/write_bit register value(s) ]
Read - [ address/write_bit register [ address/read_bit read(s) ]
Example of bus write cycle.
Example of bus read cycle - Part 1
Example of bus read cycle - Part 2
Overview

• Typical applications
• Introduction to the I2C bus
• Custom slaves - why and how
• Master
• Debugging methodology and tools
• Example: steering a 4WD drone.
• Ideas for advanced bus configurations
• Recap
• Q/A
Custom Slaves

Why creating a custom I2C slave?

- Desired sensor/actuator unavailable with I2C interface.
- Less unique addresses available than slaves needed.
- Desired custom functionality on the slave:
  - Semi-autonomous reactions to stimuli.
  - Filtering/preprocessing input data.
  - Power optimization: custom “sensor hub” does the housekeeping while the main processor is idle.
  - Realtime response to inputs.
  - [your imagination here]
How to design a custom I2C slave?

- Define requirements (see previous slide).
- Choose microcontroller or microprocessor.
- Choose Scheduler or Operating System (if any).
- Define communication sub-protocol:
  - Define parameters and commands to be exchanged.
  - Organize them into “registers” and choose a free address.
Overview

• Typical applications
• Introduction to the I2C bus
• Custom slaves - why and how
• Master
• Debugging methodology and tools
• Example: steering a 4WD drone.
• Ideas for advanced bus configurations
• Recap
• Q/A
Key design criteria:

- Weight/Dimensions.
- Required computational power and average latency.
  - PC-like device
  - Embedded device, typically headless.
- Preferred programming language: interpreted vs compiled.
- Availability of busses/gpios for driving the slave(s):
  - GPIOs only: bitbang the protocol
  - I2C: user-space application vs kernel driver.
  - No GPIOs/I2C interfaces available: USB to I2C adapter.

Design of the I2C Master
Overview

• Typical applications
• Introduction to the I2C bus
• Custom slaves - why and how
• Master
• Debugging methodology and tools
• Example: steering a 4WD drone.
• Ideas for advanced bus configurations
• Recap
• Q/A
Debugging: Divide and Conquer.

- Take direct control of the bus with an ad-hoc device.
  Examples:
  - Bus Pirate (useful also for other busses)
  - USB to I2C Master adapter, also based on the FTDI FT232R chip.
  - Custom device (could be a separate project).

- Snoop the bus with a logic analyzer or a scope/advanced meter.
  Examples:
  - sigrok/pulseview with compatible logic analyzer
  - 2-channels standalone scope/meter

- Use slave-specific In Circuit Debugger/In Circuit Emulator:
  Example:
  - AVR Dragon for AVR chips (Arduino UNO, Nano, Mini, MiniPro)
Bus Pirate

- Primarily for development purposes.
- Can both sniff the bus and drive it.
- Console interface over serial (ttyACM) port, including macros, or programmatic access for several programming languages.
- Built-in pullup resistors and voltage sources (5V / 3.3V)
- Supports many other protocols.

http://dangerousprototypes.com/docs/Bus_Pirate
USB to I2C adapter

- Small footprint.
- Suitable for permanent installations.
- No need for special connections on the host: it can be used to interface with a typical PC.
- Variant available that is also SPI-capable.
- No console interface, only serial binary protocol.
- Requires protocol wrapper.

http://www.robot-electronics.co.uk/htm/usb_i2c_tech.htm
sigrok/pulseview

- De-facto standard for PC-driven measurements on linux (but available on other OSes too).
- Support for vast range of logic analyzers, scopes and meters.
- Various protocol decoders, including I2C.
- Useful for visualizing the logical signals and debugging protocol errors.
- Even very low end, inexpensive HW can provide a whole new dimension to debugging.

https://sigrok.org
https://sigrok.org/wiki/PulseView
https://sigrok.org/wiki/Supported_hardware
Overview

• Typical applications
• Introduction to the I2C bus
• Custom slaves - why and how
• Master
• Debugging methodology and tools
• Example: steering a 4WD drone.
• Improvement Ideas
• Recap
• Q/A
Bus configuration

Master @5V

Slave 1 @5V

Slave 2 @5V

Configuration with Alternative Master @3.3V

Bi-Directional Level Shifter

V1

V2

SDA1

SDA2

SCL1

SCL2

Pull Up

Pull Up

Pull Up

Pull Up

3.3V

5V
Custom Slaves

How to design a custom I2C slave?

- **Define requirements.**
- Choose microcontroller or microprocessor.
- Choose Scheduler or Operating System (if any).
- Define communication sub-protocol:
  - Define parameters and commands to be exchanged.
  - Organize them into “registers” and choose a free address.
Example: Steering a 4WD Drone

The I2C slave:

- Controls the amount of **torque** applied to each wheel.
- Controls the **direction** each wheel spins.
- Measures the **rotation speed** of each wheel through an optical encoder (Odometer).
- Exposes the parameters above to the I2C Master.
Custom Slaves

How to design a custom I2C slave?

- Define requirements (see previous slide).
- Choose microcontroller or microprocessor.
- Choose Scheduler or Operating System (if any).
- Define communication sub-protocol:
  - Define parameters and commands to be exchanged.
  - Organize them into “registers” and choose a free address.
Selecting the Slave: Arduino Mini Pro (AVR328P)

- Enough pins/functions to provide for each wheel:
  - 1 PWM output with independent configuration of the duty-cycle.
  - 2 GPIOs for selecting: Forward, Reverse, Idle, Lock
  - 1 GPIO for registering odometer input as IRQ.
- I2C HW block for interrupt-driven i2c exchanges.
- Dedicated pins for SPI-based programming.
- Small footprint.
- Low Cost.
- The clone represented in the picture has layout optimized for DIL socket mounting.

https://www.arduino.cc/en/Main/ArduinoBoardProMini
Slave-specific ICD: AVR Dragon

- Supports various programming modes, included SPI programming, through AVRDude.
- Doesn’t interfere with normal AVR operations, so it can be left plugged into the system.
- After enabling debugWire interface, it allows configuring HW/SW breakpoints, by a dedicated backend for gdb/ddd.

http://www.atmel.com/webdoc/avrdragon/
http://www.nongnu.org/avrdude/
http://www.larsen-b.com/Article/315.html
How to design a custom I2C slave?

- Define requirements (see previous slide).
- Choose microcontroller or microprocessor.
- Choose Scheduler or Operating System (if any).
- Define communication sub-protocol:
  - Define parameters and commands to be exchanged.
  - Organize them into “registers” and choose a free address.
Selecting the OS: ChibiOS

- RTOS: preemption, tasks, semaphores, dynamic system tic, etc.
- Small footprint: link only used code/data.
- Distinction between RTOS and BSP through HAL.
- GPLv3 for non-commercial use.
- Actively developed, but already mature.

However it had limited BSP support for AVR, lack of:

- interrupts driver for AVR GPIOs (added).
- I2C support for AVR slave mode (custom).

http://www.chibios.org/dokuwiki/doku.php
https://github.com/igor-stoppa/ChibiOS/tree/car/
Custom Slaves

How to design a custom I2C slave?
● Define requirements (see previous slide).
● Choose microcontroller or microprocessor.
● Choose Scheduler or Operating System (if any).
● Define communication sub-protocol:
   ○ Define parameters and commands to be exchanged.
   ○ Organize them into “registers” and choose a free address.
Communication Parameters - 1

For each wheel:

- **Duty Cycle** of the PWM signal used to drive it - 1 byte.
  
  $0xFF = \text{max torque} / 0x00 = \text{no torque}$.

- **Direction** of rotation - 1 byte.
  
  $0x00 = \text{idle} / 0x01 = \text{reverse} / 0x02 = \text{forward} / 0x03 = \text{locked}$

- **Average period** in between slots of the optical encoder - 2 bytes.
  Writing anything resets the measurement.
Communication Parameters - 2

- **Parameter Index - 1 nibble:**
  - 0 = Duty Cycle
  - 1 = Direction
  - 2 = Average Period

- **Wheel indexes - 1 nibble:**
  - 0 = Left Rear
  - 1 = Right Rear
  - 2 = Right Front
  - 3 = Left Front
  - 4 = All
Custom Slaves

How to design a custom I2C slave?

- Define requirements (see previous slide).
- Choose microcontroller or microprocessor.
- Choose Scheduler or Operating System (if any).
- Define communication sub-protocol:
  - Define parameters and commands to be exchanged.
  - Organize them into “registers” and choose a free address.
Sub-Protocol: registers

Register format: 0xαβ
- α = Parameter Index
- β = Wheel Index

Address: 0x10

Bus Pirate format:
[ = start bit
] = end bit
r = read byte
address times 2, for R/W bit

Example - in Bus Pirate Format:

[ i2c_addr reg_addr=(parm,wheel) reg_value]

[0x20 0x20 0x02] Left Rear Forward
[0x20 0x21 0x01] Right Rear Backward
[0x20 0x22 0x01] Right Front Backward
[0x20 0x23 0x02] Left Front Forward
[0x20 0x14 0xFF] Wheels set to max torque

The car spins clockwise.
Design of the I2C Master

Key design criteria:
- Weight/Dimensions: must fit on the drone.
- Required computational power and average latency
  - Embedded device, it can provide enough computational power.
- Availability of busses/gpios for driving the slave(s):
  - Native I2C available on most candidates
  - user-space application is sufficient: no requirement for extremely low latency, might change later on
- Preferred programming language: interpreted vs compiled.
Master:
Intel Edison

- x86-64
- Built-in connectivity:
  - Wifi
  - Bluetooth
  - OTG - Ethernet over USB
  - Serial Console
- I2C available through add-on breakout board, normally @3. 3V, here hacked @5V

https://www.sparkfun.com/products/13034
**5V Mod for Sparkfun I2C Breakout**

Disconnected internal voltage regulators

Purpose:
make the I2C breakout open drain, 5V compatible

Removed Pull-UP resistors

Can accept 5V
OS: Linux Flavors

**Official Edison distro (based on Poky/OE)**

**Ubilinux (Debian port)**
http://www.emutexlabs.com/ubilinux

**Ostro Project using libmraa**
https://download.ostroproject.org/builds/ostro-os/latest/images/edison/
http://iotdk.intel.com/docs/master/mraa/
From Bus Pirate format to Python

Example - in Bus Pirate Format:

```
[ i2c_addr  reg_addr=(parm,wheel)  reg_value]
[0x20  0x20  0x02]  Left Rear Forward
[0x20  0x21  0x01]  Right Rear Backward
[0x20  0x22  0x01]  Right Front Backward
[0x20  0x23  0x02]  Left Front Forward
[0x20  0x14  0xFF]  Wheels: max torque
```

The car spins clockwise.

Note:

Bus Pirate simply dumps data on the bus, so the address 0x10 must be shifted left because of the R/W bit.

Example - Python with libmraa:

```
#!/usr/bin/python

import mraa

x = mraa.I2c(1) # Select the correct I2C bus
x.address(0x10) # The library does the shift
x.writeReg(0x20, 0x02) # Left Rear Forward
x.writeReg(0x21, 0x01) # Right Rear Backward
x.writeReg(0x22, 0x01) # Right Front Backward
x.writeReg(0x23, 0x02) # Left Front Forward
x.writeReg(0x14, 0xFF) # Wheels: max torque
```

The car spins clockwise.
Alternative Master: BeagleBone Black

- Cortex A8
- Built-in connectivity:
  - Ethernet
  - Ethernet-over-USB
  - Serial Console
- I2C available through standard connector, open drain, compatible with @3.3V
- C userspace program using libi2c.

https://beagleboard.org/black
Overview

• Typical applications
• Introduction to the I2C bus
• Custom slaves - why and how
• Master
• Debugging methodology and tools
• Example: steering a 4WD drone.
• Ideas for advanced bus configurations
• Recap
• Q/A
Ideas for improvement

- Add multi-master support
  - The current implementation is efficient wrt Slave time because it is event-driven and there is action happens only as result of an IRQ firing (no polling).
  - The Master, however, is polling the slave and polling is never a particularly good idea:
    - poll too often and it will overload the system
    - poll too seldom and important events might escape the window-of-opportunity

- Add arbitrary capability to R/W memory areas over I2C
  - live debugging of the I2C Slave.
  - Useful for memory mapped peripherals.
  - Could be used in conjunction with the memory map & linker scripting.
Overview

• Typical applications
• Introduction to the I2C bus
• Custom slaves - why and how
• Master
• Debugging methodology and tools
• Example: steering a 4WD drone.
• Ideas for advanced bus configurations
• Recap
• Q/A
Questions?
Thank you!