MICROCONTROLLERS

STM32 Peripherials – I2C

Dr. Saeed Ebadollahi

References:

- ARM® Cortex® M4 Cookbook – Mark Fischer – Packt publishing – 2016
- The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors – Joseph Yiu – Newnes – 2014
- Discovering the STM32 Microcontroller - Geoffrey Brown - 2012

Introduction

- **I²C** (**Inter-Integrated Circuit**), pronounced *I-squared-C*, is a synchronous, multi-master, multi-slave, packet switched, single-ended, serial computer bus invented in 1982 by Philips Semiconductor (now NXP Semiconductors). It is widely used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication. Alternatively I²C is spelled **I2C** (pronounced I-two-C) or **IIC** (pronounced I-I-C).
- I2C is a two wire protocol used to connect one or more "masters" with one or more "slaves", although we only discuss the case of a single master (the STM32) communicating with slave devices.

SPI vs I2C

- As we shall see, the software required to interface with I2C devices is considerably more complicated than with SPI. For example, I2C has multiple error conditions that must be handled, SPI has no error conditions at the physical level. Similarly, I2C has multiple transaction types, while SPI has a single basic transaction type. Furthermore, SPI is generally a much faster bus (1-3Mbit/sec vs 100-400Kbit/sec). The greatest advantage of I2C over SPI is that the number of wires required by I2C is constant (2) regardless of the number of connected devices whereas SPI requires a separate select line for each device. In place of select lines, I2C devices have internal addresses and are selected by a master through the transmission of this address over the bus. This difference makes I2C a good choice where a large number of devices must be connected. Finally, I2C is a symmetric bus which can support multiple masters whereas SPI is completely asymmetric.

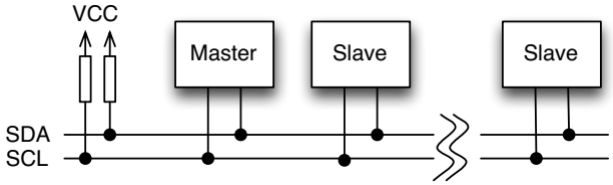
I2C vs SPI

I2C	SPI
Speed limit varies from 100kbps, 400kbps, 1mbps, 3.4mbps depending on i2c version.	More than 1mbps, 10mbps till 100mbps can be achieved.
Half duplex synchronous protocol	Full Duplex synchronous protocol
Support Multi master configuration	Multi master configuration is not possible
Acknowledgement at each transfer	No Acknowledgement
Require Two Pins only SDA, SCL	Require separate MISO, MOSI, CLK & CS signal for each slave.
Addition of new device on the bus is easy	Addition of new device on the bus is not much easy a I2C
More Overhead (due to acknowledgement, start, stop)	Less Overhead
Noise sensitivity is high	Less noise sensitivity

I2C	SPI
Operate in 7 bit / 10 bit mode	Mode 0,1,2,3 is used depending on that data will latched.
Maximum Device connection is limited by Bus Capacitance.	Limited with SS pins
Developed by Philips (currently known as NXP)	Developed by Motorola (Currently known as Freescale)
Clock Stretching mechanism available	No Clock Stretching
No Daisy chain configuration	Daisy chain configuration possible
Low Throughput	High Throughput
Complex Hardware	Simple Hardware
Hot Plugging is possible	Hot Plugging is not possible

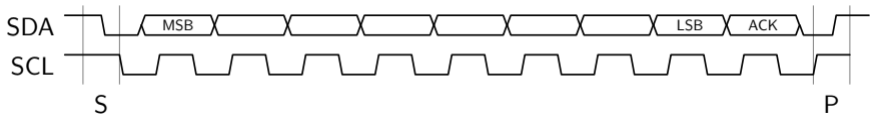
I2C

- Electrically, I2C is a "wired-or" bus – the value of the two signal wires is "high" unless one of the connected devices pulls the signal low. On the left side of Figure 9.1 are two resistors that force ("pull up") the default value of the two bus wires to VCC (typically 3.3V). Any device on the bus may safely force either wire low (to GND) at any time because the resistors limit the current draw; however, the communication protocol constrains when this should occur. The two wires are called SCL (serial clock line) and SDA (serial data/address). To communicate, a master drives a clock signal on SCL while driving, or allowing a slave to drive SDA. Thus, the bit-rate of a transfer is determined by the master.



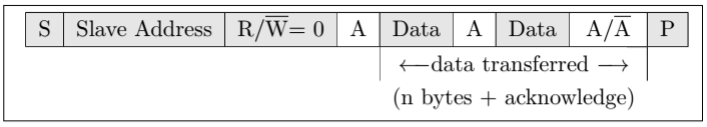
I2C Protocol

- Communication between a master and a slave consists of a sequence of transactions where the master utilizes the SCL as a clock for serial data driven by the master or a slave on SDA. Every transaction begins with a Start condition (S) and ends with Stop condition (P). A transaction consists of a sequence of bytes, delivered most significant bit (MSB) first, each of which is terminated by an Acknowledge (ACK), such as illustrated here, or Not Acknowledge (NACK). The data may be sent by either the slave or the master, as the protocol dictates, and the ACK or NACK is generated by the receiver of the data. Start (S) and Stop (P) conditions are always generated by the master. A high to low transition on SDA while SCL is high defines a Start. A low to high transition on SDA while SCL is high defines a Stop.

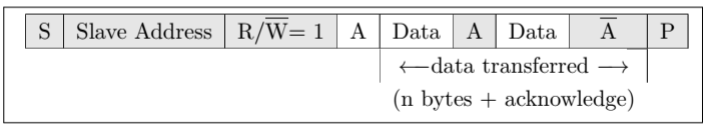


I2C Write and Read Transactions

- There are three types of transactions on the I2C bus, all of which are initiated by the master. These are write, read, and combined transactions, where a combined transaction concatenates a write and read transaction.



Write Transaction



Read Transaction

A = Acknowledge

$\overline{A}$ = Not Acknowledge

S = Start condition

P = Stop Condition

From master to slave

From slave to master

I2C Write and Read Transactions

- All 7-bit address transactions begin with a start event, the transmission of a slave address and a bit which indicates whether this is a write or read transaction by the master. The address phase it terminated by an ACK (0) provided by the slave or NACK (1). Notice that in the event there is no matching slave the electrical properties of the bus guarantee that a NACK is received by the master. In the event that address transmission is followed by a NACK, the master is obliged to generate a stop condition to terminate the transaction thus returning the bus to an idle state – i.e. both signals high.
- In a write transaction, the address phase if followed by a series of data bytes (MSB first) transmitted by the master each of which is followed by an ACK or NACK by the slave. The transaction is terminated with a Stop condition after the master has sent as much data as it wishes or the slave has responded with a NACK.
- A read transaction differs from a write transaction in that the data are provided by the slave and the ACK/NACK by the master. In a read transaction, the master responds to the last byte it wishes to receive with a NACK. The transaction is terminated with a Stop condition.