I²C bus  (Inter-Integrated Circuit)

- Designed for low-cost, medium data rate applications.
  (Phillips Semiconductor, 1980s)
- **Tutorial:** [http://www.esacademy.com/faq/i2c/](http://www.esacademy.com/faq/i2c/)
- **Characteristics:**
  - serial, byte-oriented;
  - multiple-master;
  - fixed-priority arbitration;
  - moderate speeds:
    - standard mode: 100Kbits/s
    - fast mode: 400Kbits/s
    - high speed mode: 3.4 Mbits/s
- Many microcontrollers come with built-in I²C controllers.

Serial Buses Information Page: [http://www.epanorama.net/links/serialbus.html](http://www.epanorama.net/links/serialbus.html)
I²C data link layer

- Every device has an address
  - Set by device and/or system designer.
  - 7 bits in standard (10 bits in extension).
  - Bit 8 of address signals read (1) or write (0).
- General call address (0000000) for broadcast.
- Bus transaction = series of one-byte transmissions
  - Master sends slave address followed by data to or from slave.
  - Good for “data-push” programming.
I²C physical layer

- Uses only two wires (plus ground)
I²C electrical interface (standard & fast speeds)

- Open collector/drain drivers (default state high)
- No global master for clock

Source: I²C Specification
I²C signaling

- Bus = “wired-AND” configuration
  - Open collector/drain drivers on SDA & SCL
  - Resistor pulls bus up to logic 1.
  - Any sender can pull the bus down to 0, even if other senders are trying to drive the bus to 1.
  - Sender “releases” SDA by disabling its driver, allowing SDA to be pulled up to logic 1.
- Data on SDA must be stable while SCL high
  - Data on SDA is sampled while SCL is high
  - SDA may change only while SCL low

Exceptions:
- SDA 1→0 while SCL=1 signals START condition
- SDA 0→1 while SCL=1 signals STOP condition
I<sub>2</sub>C data format

Start:
SDA 1->0 while SCL=1

Stop:
SDA 0->1 while SCL=1

SDA stable while SCL=1

MSB
Clock synchronization

- Master generates its own clock on SCL during data xfer
- Clock synchronization uses wired-AND
  - Driving low pulls SCL low, resetting all clock counters
  - SCL remains low while any driver pulls it low
  - SCL low time = slowest clock (others in wait states)
- First device to finish high state pulls SCL low

Source: I2C Specification
Four I2C device operating modes

- Master-sender
  - Module issues START and ADDRESS, and then transmits data to the addressed slave device

- Master-receiver
  - Module issues START and ADDRESS, and receives data from the addressed slave device

- Slave-sender
  - Another master issues START and the ADDRESS of this module, which then sends data to the master

- Slave-receiver
  - Another master issues START and the ADDRESS of this module, which then receives data from the master.

Some devices only support slave modes – sensors, memories, etc.
I²C bus arbitration

- Master may start sending if bus free
  - 2 or more may generate START at same time
- Sender listens while sending.
  - Test SDA while SCL high
- Sender stops transmitting if arbitration lost
  - Transmit 1 and hear 0 on SDA.
- Arbitration continues through address & ack bits, and then data & ack bits if necessary
Arbitration example

Source: I2C Specification
Data transfer

- Send 8-bit byte (MSB first)
- Each byte followed by acknowledge bit
  - master releases SDA line (high) during ack clock
  - slave must pull SDA low for proper acknowledge
  - if SDA left high, master may stop or repeat start
  - if master is receiving from slave, slave releases SDA to allow master to pull SDA low for ack
- Slave can hold SCL low to force wait time between bytes
Basic data formats

Master transmitting data to slave

- Master transmitting data to slave
- '0' (write)
- Data transferred (n bytes + acknowledge)
- From master to slave
- From slave to master

MBC605

Master receiving data from slave

- Master receiving data from slave
- (read)
- Data transferred (n bytes + acknowledge)

MBC606
I²C transmissions (ACKs not shown)

**multi-byte write**

S adrs 0 data data P

**read from slave**

S adrs 1 data P

**write, then read**

S adrs 0 data S adrs 1 data P

Re-start without giving up the bus
STM32 I²C Module (3 in STM32F407)

- Standard I²C compliant bus interface.
  - All I²C bus-specific sequencing, protocol, arbitration, timing
  - 7-bit and 10-bit addressing
  - Standard (≤ 100KHz) or Fast (≤ 400KHz) speed modes
  - Multi-master capability – use as master or slave

- Also supports standards:
  - SMBus (System Management Bus)
  - PMBus (Power Management Bus)

- DMA support – between memory and data register
- 2 interrupt vectors – data transfer complete and errors
STM32 I²C Module

- Serial data
  - SDA
  - Data control
  - Data shift register
  - Comparator
  - PEC calculation
  - Own address register
  - Dual address register
  - PEC register

- Serial clock
  - SCL
  - Clock control
  - Clock control Register (CCR)
  - Control registers (CR1&CR2)
  - Status registers (SR1&SR2)
  - Control logic
  - Interrupts
  - DMA requests & ACK
STM32 I²C registers

I2C_DR – I²C data register
  byte to be transmitted (start on DR write)
  byte received (RxNE=1)

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DR[7:0]</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

I2C_OAR1 – I²C own address register 1
  ADDMODE  0 = 7-bit ADD[7:1]
          ;  1 = 10-bit ADD[9:0]

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ADD MODE</td>
<td>Reserved</td>
<td>ADD[9:8]</td>
<td>ADD[7:1]</td>
<td>ADD0</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

(A second “own address” is also supported)
### STM32 I²C – control register 1

**I2C_CR1**

<table>
<thead>
<tr>
<th></th>
<th>SWRST</th>
<th>Res.</th>
<th>ALERT</th>
<th>PEC</th>
<th>POS</th>
<th>ACK</th>
<th>STOP</th>
<th>START</th>
<th>NO STRETCH</th>
<th>ENGC</th>
<th>ENPEC</th>
<th>ENARP</th>
<th>SMB TYPE</th>
<th>Res.</th>
<th>SMBUS</th>
<th>PE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

#### PE
Peripheral function **Enable** (1 enables the I²C module)

#### STOP
Generate after current byte xfer or after start condition sent

#### START
Master: repeated start generation, Slave: release bus after byte xfer

#### ACK
ACK to be returned after byte received

#### POS
If ACK bit = 1: return ACK after current byte (0) or next byte (1)

#### SWRST
Software reset (or in reset state)

#### NOSTRETCH
Enable/disable clock stretch in slave mode when ADDR or BRG flag set, until reset by software

#### ENGC
Enable “general call” (ACK address 0x00)

#### SMBUS
0 for I²C mode; 1 for SMBus mode

(Other bits for packet error checking (PEC) or SMBus setup)
SRM32 I²C – control register 2

I2C_CR2

<table>
<thead>
<tr>
<th></th>
<th>LAST</th>
<th>DMA EN</th>
<th>ITBUF EN</th>
<th>ITEVT EN</th>
<th>ITERREN</th>
<th>Reserved</th>
<th>FREQ[5:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

FREQ[5:0] = peripheral clock frequency (in MHz)
allowed values [2MHz … 42MHz]

DMA Control:
LAST: 1 = next DMA EOT is the last transfer
DMAEN: 1 = DMA requests when TxE=1 or RxNE=1

Interrupt Control (interrupt generation events on next slide)
ITBUFEN: 1 = TxE/RxNE event generates Event interrupt
ITEVTEN: 1 = Event interrupt enabled
ITERREN: 1 = Error interrupt enabled
# STM32 I²C interrupts

<table>
<thead>
<tr>
<th>Interrupt event</th>
<th>Event flag</th>
<th>Enable control bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start bit sent (Master)</td>
<td>SB</td>
<td></td>
</tr>
<tr>
<td>Address sent (Master) or Address matched (Slave)</td>
<td>ADDR</td>
<td></td>
</tr>
<tr>
<td>10-bit header sent (Master)</td>
<td>ADD10</td>
<td></td>
</tr>
<tr>
<td>Stop received (Slave)</td>
<td>STOPF</td>
<td></td>
</tr>
<tr>
<td>Data byte transfer finished</td>
<td>BTF</td>
<td></td>
</tr>
<tr>
<td>Receive buffer not empty</td>
<td>RxNE</td>
<td>ITEVFEN and ITBUFEN</td>
</tr>
<tr>
<td>Transmit buffer empty</td>
<td>TxE</td>
<td></td>
</tr>
<tr>
<td>Bus error</td>
<td>BERR</td>
<td></td>
</tr>
<tr>
<td>Arbitration loss (Master)</td>
<td>ARLO</td>
<td></td>
</tr>
<tr>
<td>Acknowledge failure</td>
<td>AF</td>
<td></td>
</tr>
<tr>
<td>Overrun/Underrun</td>
<td>OVR</td>
<td></td>
</tr>
<tr>
<td>PEC error</td>
<td>PECERR</td>
<td></td>
</tr>
<tr>
<td>Timeout/Tlow error</td>
<td>TIMEOUT</td>
<td></td>
</tr>
<tr>
<td>SMBus Alert</td>
<td>SMBALERT</td>
<td></td>
</tr>
</tbody>
</table>
# STM32 I²C – status register 1 (of two)

**I²C_SR1**

<table>
<thead>
<tr>
<th>SMB ALERT</th>
<th>TIME OUT</th>
<th>Res.</th>
<th>PEC ERR</th>
<th>OVR</th>
<th>AF</th>
<th>ARLO</th>
<th>BERR</th>
<th>TxE</th>
<th>RxNE</th>
<th>Res.</th>
<th>STOPF</th>
<th>ADD10</th>
<th>BTF</th>
<th>ADDR</th>
<th>SB</th>
</tr>
</thead>
<tbody>
<tr>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
</tbody>
</table>

- **ADDR**: Master: 1= address sent  
  Slave: 1= received address matched OAR register or gen call
- **SB**: Master: 1= Start generated (clear by reading SR1 & DR)
- **TxE**: 1= transmitter buffer (DR) empty (can send a new byte)
- **RxNE**: 1= receiver buffer (DR) not empty (byte has been received)
- **BTF**: 1= data byte transfer finished successfully
  - RxNE=1 & DR not read yet; TxE=1 & DR not written yet
- **ARLO**: 1= arbitration lost detected (this device lost to another)
- **STOPF**: 1= slave detected stop condition after ACK
- **OVR**: 1= DR register overrun/underrun (data lost)
- **AF**: 1= ACK failure (no ACK returned)
- **BERR**: 1= bus error (misplaced Start/Stop condition)
- **ADD10**: 1= master sent 1\textsuperscript{st} byte of 10-bit address
STM32 I²C – status register 2

I2C_SR2

<table>
<thead>
<tr>
<th>BUSY:</th>
<th>1= communication ongoing on the bus (cleared by Stop)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSL:</td>
<td>0= slave mode (default)</td>
</tr>
<tr>
<td></td>
<td>1= master mode (START has been sent)</td>
</tr>
<tr>
<td>TRA:</td>
<td>From R/W address bit:</td>
</tr>
<tr>
<td></td>
<td>1= data bytes to be TRAnsmitted</td>
</tr>
<tr>
<td></td>
<td>0= data bytes to be received</td>
</tr>
<tr>
<td>DUALF:</td>
<td>Received address matches OAR1 (0) or OAR2 (1)</td>
</tr>
<tr>
<td>GENCALL:</td>
<td>General call address (0x00) received when ENARP=1</td>
</tr>
</tbody>
</table>

(Other bits for PEC or SMBus)
STM32 I2C bus “events” (from flags)

- Master modes:
  - EV5: Start bit sent
    - BUSY – MSL – SB
  - EV6: Slave acknowledged address
    - BUSY – MSL – ADDR
  - EV8: DR ready for new byte to transmit
    - BUSY – MSL – TXE (transmit buffer empty)
  - EV9: new byte received in the DR
    - BUSY – MSL – RXNE (receive buffer not empty)
STM32 I2C bus “events” (from flags)

- Slave modes:
  - EV1: Own address received, data to be received from master
    - BUSY – ADDR (MSL=0, TRA=0)
  - EV1: Own address received, data to be sent to master
    - BUSY – ADDR – TRA (MSL=0)
  - EV2: Slave byte received
    - BUSY – RNXE (receive buffer not empty)
  - EV3: Slave byte transmitted
    - BUSY – TRA - TXE (transmit buffer empty)
    - BUSY – TRA – TXE - BTF (transmit buffer empty and byte transfer finished)
I²C clock control register
I2C_CCR

F/S  0=standard mode (≤ 100KHz), 1=fast mode (≤ 400KHz)

Standard Mode:
  \[ T_{\text{high}} = T_{\text{low}} = CCR \times T_{\text{PCLK1}} \]

Fast Mode, DUTY = 0
  \[ T_{\text{high}} = T_{\text{low}} = CCR \times T_{\text{PCLK1}} \]

Fast Mode, DUTY = 1  (to reach 400KHz)
  \[ T_{\text{high}} = 9 \times CCR \times T_{\text{PCLK1}} \]
  \[ T_{\text{low}} = 16 \times CCR \times T_{\text{PCLK1}} \]

Ex: To generate 100KHz SCL in standard mode.
  If FREQR = 08, \( T_{\text{PCLK1}} = 125\text{ns} \)
  Set CCR = 40 (0x28)
  \[ T_{\text{high}} = T_{\text{low}} = 40 \times 125\text{ns} = 5000\text{ns} \]
Hierarchical/modular software design

Replace Application/Codec with other functions that use I2C driver
STM32 I2C peripheral driver functions

- **Configure control registers, etc.**
  - `I2C_Init()` – initialize control registers, clock, etc.
  - `I2C_Cmd()` – enable the I2C module
  - Other functions to set/clear individual control bits

- **Bus management functions**
  - `I2C_GenerateStart()` – signal START on the bus
  - `I2C_Send7bitAddress()` – send slave address
  - `I2C_GenerateStop()` – signal STOP on the bus

- **Data transfer functions**
  - `I2C_SendData()` – send one byte to DR
  - `I2C_ReceiveData()` – get one byte from DR

- **Bus monitoring functions**
  - `I2C_CheckEvent()` – test status flags for a bus “event”
  - `I2C_GetFlagStatus()` – test one flag in status register
Typical master-to-slave transfer

Codec_WriteRegister(RegAddr, RegValue)

- `I2C_GetFlagStatus()` – check flag BUSY=0
- `I2C_GenerateStart()` – signal START on the bus
- `I2C_CheckEvent()` – test EV5 flags (start correct)
- `I2C_Send7bitAddress()` – send slave address
- `I2C_CheckEvent()` – test EV6 flags (slave address ACK)
- `I2C_SendData()` – send first byte (register address) to DR
- `I2C_CheckEvent()` – test EV8 flags (data sending, DR ready for byte)
- `I2C_SendData()` – send second byte (register value) to DR
- `I2C_GetFlagStatus()` – check flag BTF=1 (byte transfer finished)
- `I2C_GenerateStop()` - signal STOP on the bus
Audio Code driver: key functions

- **Codec_Init()** – all related device/module initialization:
  - **Codec_GPIO_Init()**
    - Enable clocks in RCC and all GPIO pins for I2C, I2S, DAC
  - **Codec_Reset()** – reset the Codec (RESET pin)
  - **Codec_CtrlInterface_Init()**
    - Calls I2C_Init() with required parameters
  - Configure all Codec registers via I2C functions
  - **Codec_AudioInterfaceInit()**
    - Initialize DAC and I2S modules

- **Codec_WriteRegister()** – write value to a code register
- **Codec_ReadRegister()** – read value from a code register
STM32F4-Discovery Software

- STM32F407VG peripheral drivers added to project from “Pack” (I2C, SPI, DAC, etc.)
  - \textit{stm32f4xx_i2c.c} => all I2C control/access functions

- Discovery board chip drivers in
  - ..\stm32f4discovery_fw\Utilities\STM32F4-Discovery\stm32f4_discovery_audio_codec.c
    - Initialize and control audio codec chip
    - Calls functions from I2C, I2S, GPIO, RCC module drivers
Cirrus Logic CS43L22 Portable Audio DAC with Integrated Class D Speaker Driver
Discovery CS43L22 schematic