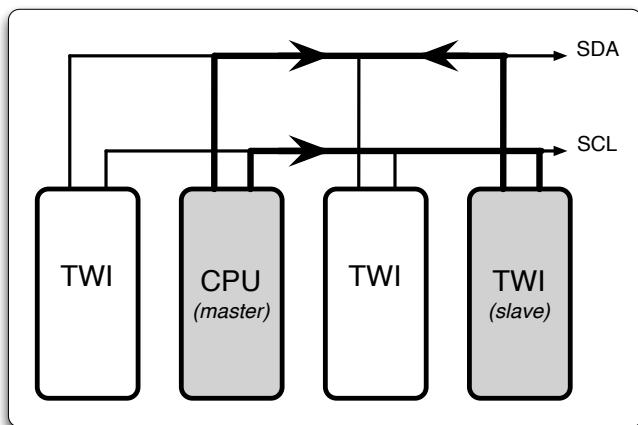


# Mikrodatorprojekt TSIU51

## TWI — version 26 januari 2022

### Inledning

Mellan processor och kringenheter i inbyggda system används inte sällan seriell kommunikation. Det finns flera, till exempel USART och SPI.



Ett annat vanligt protokoll för sådan dataöverföring är TWI, *Two-Wire Interface*. Det är det vi skall använda i denna lab.

I detta labhäftet beskrivs TWI endast översiktligt. En noggrannare beskrivning av protokollet finns på respektive komponents datablad. Använd det för laborationsuppgifterna.

Din uppgift är att skriva ett program som genomför en korrekt TWI -transaktion till kretsen PCF8574. Kretsen är en 8-bits seriell till parallell-omvandlare.

Programmet skall visa r16:s låga nibble i hexadecimal form på en sjusegments display.

Till din ledning skall du själv först skriva hela programmets datastruktur i JSP och därefter skriva själva assemblerkoden.

Laborationen är till stor del att läsa datablad, genomföra en strukturerad lösningsgång och lösa uppgiften med JSP.

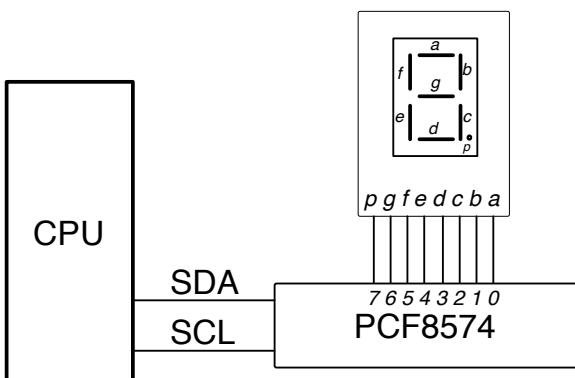
Laborationen förutsätter vana i AVR-assembler.

### Hårdvara

Innan laborationen måste du läsa på om:

- TWI -protokollet<sup>1</sup>
- I/O-kretsen PCF8574
- Sjusegmentsmodulen

Sjusegmentsmodulen är ansluten via en PCF8574 på kortet.



PCF8574 tar emot en byte seriellt enligt TWI -protokollet och visar den, efter att hela bytes överförts, parallellt på sina utpinnar där en sjusegmentsmodul är ansluten.

Signaleringen sker via SDA och SCL enligt komponentens datablad.

**Elektrisk signalering** På DAvid finns inbyggda pull-up-motstånd som håller signalaerna SDA och SCL höga i vila.

Bussignalerna för **aldrig** drivs höga. De får enbart drivas låga (sänkas).

I praktiken betyder det att signalnivåerna styrs av varje portpinnes utgångsbuffert

- Frånkopplad buffert tillåter pull-up-motståndet att dra bussignalen hög,
- inkopplad buffert med portbit satt till läg, tillåter den att driva bussignalen läg.

**Hög signal fås alltså genom att sätta portbiten till ingång, låg signal genom att sätta den till utgång.**

Processorns datablad lovar att samtliga I/O-register är nollställda vid RESET, dvs PORT<sub>x</sub> och DDR<sub>x</sub> är båda 0.

<sup>1</sup>En översiktig beskrivning finns sist i denna labinstruktion.

# Dataöverföring TWI

Notera i beskrivningen nedan speciellt begreppen

- *Master*
- *Slav*
- *Transaktion*
- *Busstillstånd (START/STOP)*
- *Slavadress (SLA+R/SLA+W)*
- *Kvittens, acknowledge, (ACK)*

**Översiktligt** genomgår en dataöverföring följande förflopp:

1. Kommunikation sker mellan två enheter: *Master* och *Slav*. Mellan dessa enheter kan läsning eller skrivning ske.
  2. En *transaktion* är en komplett överföring från busstillståndet **START** till **STOP**.
  3. Master initierar en transaktion genom att begära, och få, bussen. Detta sätter bussen i **START-tillstånd**.
  4. Master adresserar en enhet på bussen genom att skicka dess adress, *slave address, SLA*, (7 bitar) tillsammans med en bits läs- eller skriv-begäran, *read/write, R/W*. Denna byte kallas **SLA+R** eller **SLA+W**.
- Master genererar klockan oavsett läsning eller skrivning.
5. Slav följer med i busskommunikationen men kan inte initiera en transaktion.
  6. Slav kan känna igen sin egen adress och skickar kvittens när den fångat upp denna med en *acknowledge, ACK*.
  7. Information överförs byte för byte. Den enhet som är *mottagare* av data ansvarar för att skicka **ACK** till *sändaren*. Alltså oberoende om den är master eller slave.
  8. Master avslutar transaktionen genom att sätta bussen i **STOP-tillstånd** och släpper därmed bussen.<sup>2</sup>

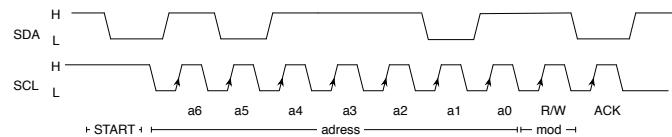
För enbart skrivning till kretsen PCF8574 behövs således fyra steg:

1. Begär och få bussen (**START**), detta gör Arduino till Master.
2. Addressera Slav och begär skrivning, **SLA+W**
3. Överför data. En byte per transaktion. Mottagaren kvitterar varje byte med **ACK**.
4. Släpp bussen (**STOP**).

<sup>2</sup>I ett generellt fall kan nu en annan enhet skicka **START** och bli master. På DAvid kan dock enbart Arduino göra detta.

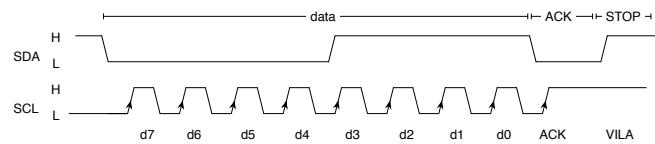
<sup>3</sup>För skrivning skall **R/W** vara låg.

**Digital signalering** En **TWI-Slav** addreseras för läsning med en 7-bitars slavadress och hög<sup>3</sup> R/W-bit genom sekvensen **START** och **SLA+R**



Här har slaven kvitterat genom att hålla SDA låg under klockpulsen för **ACK**.

Nu vet master att adresseringen lyckats och fortsätter generera klockpulser. Vid läsning lägger slav ut data i takt med den givna klockan



och master kvitterar med SDA låg under klockpulsen **ACK**. I figuren ovan har master sedan släppt bussen.

**Notera** En byte kräver åtta klockflanker och mottagaren kvitterar med en nionde bit, **ACK, acknowledge**. Genom att läsa denna kan sändaren avgöra om den får fortsätta sända eller inte. Master måste alltid ge en klockpuls för att mottagaren skall kunna signalera **ACK**.

Sammanlagt överförs alltid nio bitar, oavsett adressering **SLA+RW** eller data.

I vila och både **SDA** och **SCL** är höga.

Enligt **TWI** får **SDA** endast växla nivå då **SCL** är låg, dvs **SDA** är oförändrad så länge **SCL** är hög.

Databiten kan läsas av vid **SCL**:s stigande flank.

Mottagaren kvitterar genom den nionde biten, **ACK**.

# Lågnivårutiner

Till din hjälp är den längsta nivån av rutiner redan kodade. Använd dessa som längsta ”löv” i ditt JSP-diagram.

Rutinerna är konstruerade under antagandet att **SCL** är hög vid vila och ledig buss, dvs innan **START** respektive efter **STOP**. **SCL** är låg mellan giltiga data vid tagen buss.

DAvid-kortet har inbyggda *pull-ups* och hela **PORTC** är automatiskt ingång vid start av processorn dvs ingen portinitiering behöver göras.

Använd inte rutinerna nedan innan du övertygat dig om hur de fungerar.

Då  $f_{CPU} = 16 \text{ MHz}$  behövs en **WAIT**:<sup>4</sup>. Du får själv skriva den rutinen, mer än  $10 \mu\text{s}$  fungerar bra.

```
.equ ADDR_RIGHT8 = $25
.equ SLA_W      = (ADDR_RIGHT8 << 1) | 0
.equ SLA_R      = (ADDR_RIGHT8 << 1) | 1

.equ SCL = PC5
.equ SDA = PC4

START:
    sbi    DDRC, SDA
    call   WAIT
    sbi    DDRC, SCL
    call   WAIT
    ret

STOP:
    sbi    DDRC, SDA
    call   WAIT
    cbi    DDRC, SCL
    call   WAIT
    cbi    DDRC, SDA
    call   WAIT
    ret

SDL:
    sbi    DDRC, SDA
    call   WAIT
    cbi    DDRC, SCL
    call   WAIT
    sbi    DDRC, SCL
    call   WAIT
    ret

SDH:
    cbi    DDRC, SDA
    call   WAIT
    cbi    DDRC, SCL
    call   WAIT
    sbi    DDRC, SCL
    call   WAIT
    ret
```

---

<sup>4</sup>Experiment visar att den inte behövs på alla ställen i rutinerna men för att vara konsekvent är allt nedslöat med **WAIT**:

# Laborationsuppgifter

Med ny hårdvara och begränsade möjligheter att felsöka är det väsentligt att vara helt säker på så mycket som möjligt. **Begränsa möjliga felkällor så mycket du bara kan!** En del i detta är att läsa på och designa kod innan den överförs till assembler.

I detta läge är det oerhört nyttigt att kunna se signaleringen på TWI -bussen. Att kunna se antalet klockpulser, datavärdena och om den adresserade enheten svarar med ACK underlättar mycket. I nuläget har vi dock inte den lyxen.

I brist på oscilloskop eller logikanalysator får vi istället programmera försiktigt. Det är således helt i sin ordning att starta med ett mycket enkelt program å la:

```
TWI_SEND:  
    call    START  
    call    SDH      ; adress  
    call    SDL  
    :  
    :          ; data  
    :  
    call    STOP  
    ret
```

för att få sjusegmentsdisplayen att svara med rätt reaktion.

Om displayen påverkas vet man att antalet klockpulser stämmer, att enheten addresseras, att man hanterar ACK osv.

Prova med några olika data och konstatera att displayen svarar på rätt sätt.

När rutinen fungerar måste den snyggas till, vilket i det här fallet handlar om att faktorisera, loopa det uppenbara och förse den med *enbart* det argument den behöver för att göra sitt jobb (Se uppgift 1e)

## 1. Skriv till TWI-bussen

Första delen av laborationen syftar till att kunna skriva till en TWI -enhet.

**Uppgift 1a** Skriv ett program som skickar ut r16:s innehåll till en av sjusegmentsdisplayerna. Själva TWI -transaktionen skall heta **TWI\_SEND:** och behöver adress och data.

Rita först ett strukturdiagram enligt JSP som löser uppgiften. Skriv sedan programmet i enlighet med ditt strukturdiagram.

Använd JSP för att identifiera rimliga subrutinkandidater och skriv dessa som subrutiner. Subrutiner skall utföras så generella som möjligt.

Identifiera speciellt vilka delar som är sekvens, iteration respektive selektion. Koden skall, åtminstone efter uppsnyggning, vara strukturerad.

Programmet måste vara förberett, simulerat och ”färdigt” innan laborationen.

**Uppgift 1b** Skriv och använd en födröjningsrutin **DELAY** för att få displayen att visa och räkna 0–F i i lämplig takt.

**Uppgift 1c** (Extra). Få istället displayen att tända en slingrande ”mask” i åtta med någon lagom takt.

**Uppgift 1d** Använd samma rutin för att tända och släcka de två lysdioderna i rotationsencodern.

**Uppgift 1e** Kodstruktur. Nu vet du att du kan påverka sjusegmentsdisplayerna och lysdioderna i rotationsencodern om du förser dina rutiner med korrekta argument. Du vill ha enkla *handtag* till respektive funktion utan att behöva komma ihåg alla argument.

Skriv rutiner som stelopererar argumenten till några olika funktioner, till exempel:

Rutin	Funktion
ROTLED_GREEN:	Tänd grön lysdiod
ROTLED_RED:	Tänd röd lysdiod
ROTLED_BOTH:	Tänd båda lysdioderna
ROTLED_OFF:	Släck båda lysdioderna

Med rutinerna konstruerade och namngivna som ovan behöver de endast anropas med namn och deras interna konstruktion, adresser och så vidare är osynligt.

För skrivning till höger och vänster sjusegmentsdisplay behövs ett argument i r16 med data, resten kodas hårt i rutinerna:

Rutin	Funktion, argument i r16
RIGHT8_WRITE:	Skriv till den högra sjusegmentdisplayen
LEFT8_WRITE:	Skriv till den vänstra sjusegmentdisplayen

Alternativt kan även ett argument motsvarande höger/vänster användas. Rutinerna ovan kan fortfarande användas.

## 2. Läs från TWI-bussen

I den andra delen av laborationen ska du läsa från knapparna L1/L2/R1/R2/JOY\_L/JOY\_R.

Enligt den tidigare beskrivningen är det fortfarande **MASTER** som genererar klockflankerna vid läsning och **SLAV** får bara ändra data när SCL är låg.

En konsekvens av detta blir att data är giltigt (redan) då SCL går hög. Kod för avläsning av en bit kan då utföras som en sekvens av avläsning av datavärde och därpå kommande komplett klockcykel på SCL. Denna klockcykels fallande flank ger då klartecken till **SLAV** att lägga ut nästa bit och så vidare.

**Uppgift 2a** Se i schemat hur knapparna L1/L2/R1/R2/JOY\_L/JOY\_R nås, dvs vilken I/O-adress och pinne som de hör till. Skriv kod för att läsa in dessa knappar och skicka ut detta direkt till en sjusegmentsdisplay.

Du kommer behöva skriva rutinen SCP: som ger en hel klockcykel på SCLoch **SWITCHES**: som returnerar bitarna L1/L2/R1/R2/JOY\_L/JOY\_R i r16.

**Avkodning och kodstruktur.** Det kan vara stölkigt att ha en enda rutin som känner av alla knappar i r16 som man sedan studerar när man kan ha rutiner som gör detta jobb åt oss.

Här föreslås två metoder där vardera har för- och nackdelar beroende på sammanhanget.<sup>5</sup>

**Metod 1.** Använd **SWITCHES**: för rutinerna nedan (där Q står för *Question*) som returnerar status för enskilda knappar

Rutin	Funktion
L1Q	Z=1 om L1 nedtryckt
L2Q	Z=1 om L2 nedtryckt
R1Q	Z=1 om R1 nedtryckt
R2Q	Z=1 om R2 nedtryckt
JOY_LQ	Z=1 om JOY_L nedtryckt
JOY_RQ	Z=1 om JOY_R nedtryckt

**Metod 2.** Ett annan metod är att låta **SWITCHES**: returnera en avkodad siffra i r16 motsvarande en knapp. De två översta bitarna är inte anslutna.

Avläst	Avkodat	Betydelse
--111111	0	Ingen nedtryckt
--111110	1	R1 nedtryckt
--111101	2	R2 nedtryckt
--111011	3	L1 nedtryckt
--110111	4	L2 nedtryckt
--101111	5	JOY_R nedtryckt
--011111	6	JOY_L nedtryckt

**Uppgift 2b** Välj en av metoderna ovan och styr de två lysdioderna i rotationsencodern så att ett tryck på L1 tändar röd diod och L2 tändar grön diod.

<sup>5</sup>Fundera på: Vad ska returneras om flera knappar är nedtryckta? När är den ena eller andra att föredra?

# Hårdvarustöd

Processorn ATmega328p har inbyggt hårdvarustöd för TWI. Med den kännedom om TWI vi fått ur laborationerna hittills är vi redo att använda den komplicerade hårdvaran istället för de hittills använda hemmasnickrade rutinerna.

Läs kapitlet om *Two Wire Serial Interface* i processorns datablad. Delar om *Multi-master* och arbitrering kan lämnas därhän. På DAvid-kortet är processorn alltid **MASTER** dvs enbart **MASTER-transmitter** och **MASTER-receiver** är aktuellt.

---

**Grovt sett** används den inbyggda TWI-enheten så att programmet begär en funktion, exempelvis ”lägg beslag på bussen” dvs skicka **START**, genom att sätta en bit i ett kontrollregister. Hårdvaran tar då itu med sin uppgift och meddelar när den är klar genom att sätta en bit, **TWINT**, i samma kontrollregister. Det är nu upp till programmet att beordra en fortsättning, exempelvis, ”skicka adress för skrivning”, **SLA+W**, och invänta nästa gång **TWINT**-biten meddelar att uppdraget är utfört.

På detta sätt styr programmet hela hårdvarutransaktionen men behöver inte bry sig om de finaste detaljerna.

Notera att man *nollställer* **TWINT**-biten genom att skriva en *etta* till den.

*Avancerad användning.* Till varje **TWINT**-bit bifogas en statuskod som meddelar framgång eller inte. **TWINT**-biten kan orsaka ett avbrott och statuskoden kan då användas för berätta för avbrottsrutinen varför avbrottet skedde. Avbrottsrutinen vet på så sätt var i transaktionen hårdvarustödet befinner sig och kan fortsätta (eller avsluta) transaktionen.

Planen är att ha ett fåtal funktioner/drivrutiner för att kommunicera med hårdvara

Funktion	Syfte	Inargument	Returvärde
<b>TWI_OPEN:</b>	Initiera TWI	Datatakt	—
<b>TWI_READ:</b>	Läs en byte från TWI	Adress	Data
<b>TWI_WRITE:</b>	Skriv en byte till TWI	Adress	Data

(Beroende på om till exempel **REPEATED START** behövs i projektet kan beskrivningen senare utökas. För närvarande är avsikten att få samma funktionalitet med inbyggd hårdvara som tidigare gjorts med enbart programvara.)

De hårdvaruregister<sup>6</sup> som är av intresse vid genomläsningen är

Namn	Betydelse	Argument/ Innehåll
<b>TWBR</b>	<i>Bitrate Register</i>	Bithastighet
<b>TWCR</b>	<i>Control Register</i>	Handskakning
<b>TWDR</b>	<i>Data Register</i>	Sänd/mottagen byte
<b>TWSR</b>	Status Register	Hur gick det?

**Uppgift 3a** Den tredje och avslutande delen av laborationen blir att skriva om dina TWI -rutiner så de använder den inbyggda hårdvarustödet.

---

<sup>6</sup>Observera att dessa register ligger utanför den låga I/O-arean och de nås genom **sts** och **lds**.

## A. Översiktlig beskrivning TWI

TWI är kompatibel med den seriella bussen I2C. De fungerar på samma sätt bara namnet skiljer. Här följer en översiktlig beskrivning av I2C för att ge en bakgrund och historia. Det är ett särtryck ur *Electronics World + Wireless World*, January 1994, pp 24-28.



**The I<sup>2</sup>C approach to distributed processing allows the designer to include every kind of processing and signal conditioning function on a simple two-wire bus. This proprietary Philips concept is so flexible and accommodating that other semiconductor companies have adopted it and added to the function range.**

**Design consultant Mike Button\*** reveals the secrets of its success.

## BUSMAN'S GUIDE TO I<sup>2</sup>C

**T**he low cost of microprocessor devices makes it common sense to provide future compatibility in all but the most trivial of designs. Consequently the majority of electrical and electronic circuits employ the ubiquitous microprocessor to implement logical functions. With the advent of the microcontroller with on-chip prom or eprom in the 70's, single chip solutions are now a norm.

Control functions, often involving human reaction times, are the norm in the majority of systems. The high speed data transfer rate of an 8-bit parallel data bus is likely to be unnecessary and expensive. A simple, two-wire serial interface often provides enough performance for a surprising range of applications.

The Inter-Integrated Circuit Bus, written I<sup>2</sup>C

for short and pronounced "I squared C", was invented and patented by Signetics and Philips and has become a *de facto* standard in chip to chip and board to board communication. Due to Philips' involvement in audio, television and telecoms, a legion of I<sup>2</sup>C bus devices is now available. Other semiconductor manufacturers are also making devices for the bus.

The range includes 8-bit data converters, adc/dac, audio frequency generators, clock timers, ram, eeprom, led/lcd display drivers and a range of audio, radio and television control circuits. Several microcontrollers have on-chip hardware to ease programming and relieve the processor of software overheads. The PCB8XC552 and PCB8XC652 are of particular note.

The ability to add more master devices at

any time puts great power at the finger tips of a system designer. When the microcontroller software becomes overloaded additional microcontrollers can be added. Alternatively external test equipment can contain a master and slaves to exercise, test and report on system functionality. In control functions where response time in the order of 1ms is acceptable, the I<sup>2</sup>C bus provides a convenient adaptable and low cost solution.

### Definitions

The I<sup>2</sup>C bus is a bi-directional two wire serial bus having a defined protocol which allows data transfer between compatible integrated circuits. The number of devices that can be

\*TDR Ltd.

attached to the bus is limited only by the bus capacitance. The bus is so designed that the addition or removal of a device will not affect the working of any devices still on the bus. Philips defines the bus as multi-master, multi-slave working.

The standard-mode of operation can handle data and clock signals at baud rates up to 100kHz. Fast-mode devices are now being made available that will work at 400kHz. The low speed mode is used when microprocessors need to poll the bus in software. A 10-bit address mode was recently introduced to provide more independent slave addresses. All modes of operation conform to the same protocol and provide enhancements for use in special cases.

The I<sup>2</sup>C bus uses two leads plus a common (earth) return. The SCL lead carries the clock pulses, the SDA lead carries the data information. Commencement of a data transfer is indicated by a START condition [S]. The end of transfer is indicated by a STOP condition [P].

Data is transferred in a 9-bit word, comprising of eight data bits plus an acknowledge bit. The acknowledge signal, ACK, is sent after every data byte to indicate that data transfer may continue. The NACK (not ACK) signal indicates that no further data transfer is possible and a STOP or repeated START condition should be sent. The device that generates the START and STOP conditions and provides clock pulses is called a MASTER. Devices that respond to a MASTER are called SLAVES.

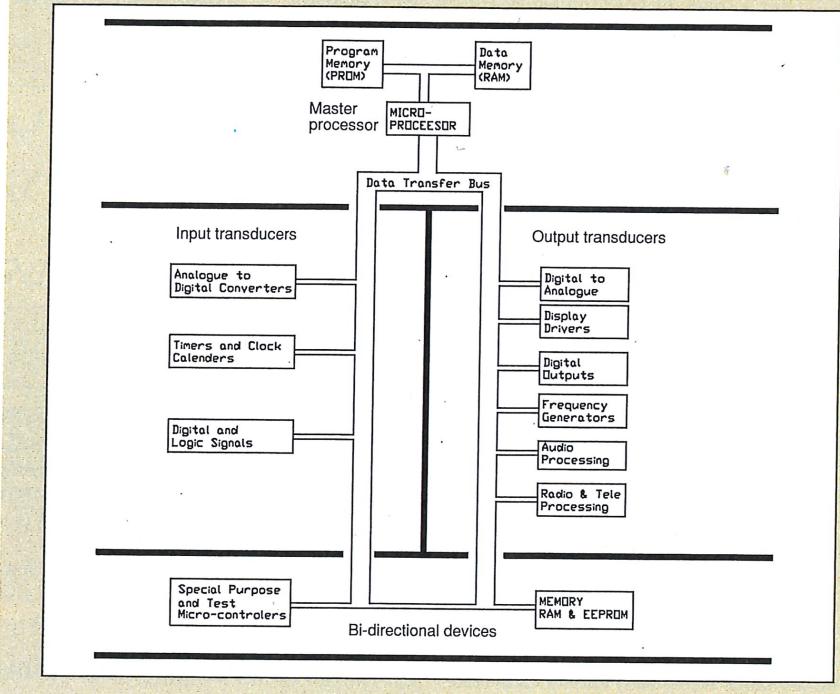
All SLAVES devices are provided with a unique SLAVE ADDRESS. A MASTER, wishing to transfer data to or from a SLAVE must, prior to data transfer, generate the address appropriate to the required SLAVE. A SLAVE on recognising its own address will generate an ACK signal.

The device that sends data is called a TRANSMITTER. Conversely, the device that accepts data is called a RECEIVER. Except for the condition when a SLAVE acknowledges its own address, it is the RECEIVER that generates the ACK signal.

Thus an I<sup>2</sup>C bus device can be any one of four types dependent on its function during data transfer. The majority of devices produced for the bus are slave devices and can be either transmitters or receivers dependent on function or mode of operation. The master

## Hypothetical complex microcontroller system

The I<sup>2</sup>C bus provides a two wire communications channel for both commands and data for all elements of the system. It replaces a multiple line address/data bus with consequent savings in PCB complexity and area. It was designed as a simple communications channel between individual ICs but is increasingly used as a local network between systems, providing they are not speed sensitive. System inputs may include: converted analogue to digital signals from transducers such as temperature sensors, analogue joysticks, etc. and logical signals from level switches, key contacts, etc. Transducer outputs may include: converted digital to analogue signals to drive motors, current loops, etc., digital signals to switch relays or lamps and drivers for led or lcd displays. Special functions such as television receiver channel selection or teletext reception and display may also be included.



function is normally provided from a microprocessor with an I<sup>2</sup>C bus controller chip (*PCD8584*) or a microcontroller with on-chip I<sup>2</sup>C bus hardware.

Master → direction of data → slave transmitter  
Master → direction of data → slave receiver → transmitter  
Subject to bus capacitance limitations, there

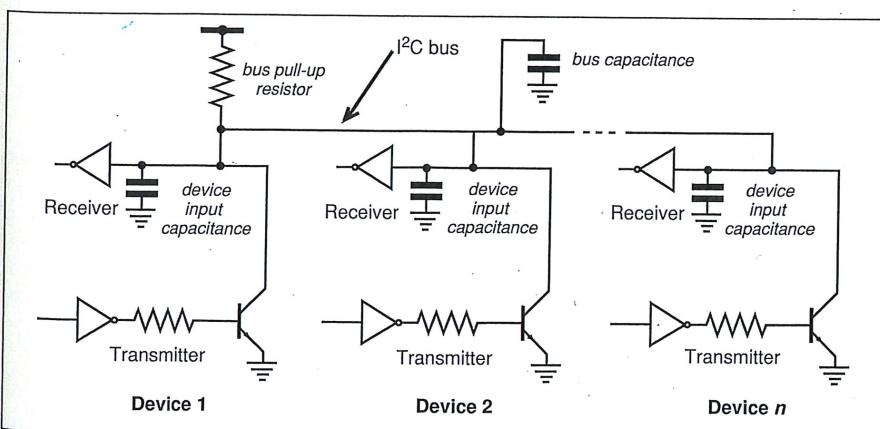
can be any number of masters or slaves but only one transmitter-receiver pair are allowed to use the bus at any one time.

### Electrical properties

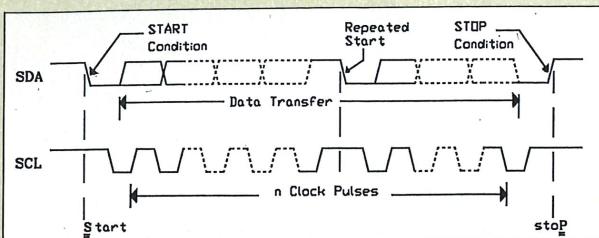
The electrical connections to the I<sup>2</sup>C bus rely on open collector wired-and logic gating. Both the SDA and SCL leads have the same electrical configuration.

**Figure 1** shows a typical bus connection for one of the wires (SCL or SDA). If all the device transmitters (devices 1, 2, ..., n) are at logic-high the bus wire will be pulled high to V<sub>CC</sub> (normally but not necessarily +5V) via the bus pull-up resistor. All of the device receivers will see this high state on the bus as a logic-high signal. If any of the device transmitters go to logic-low the bus wire will be pulled to ground potential and all of the device receivers, including the receiver of the device

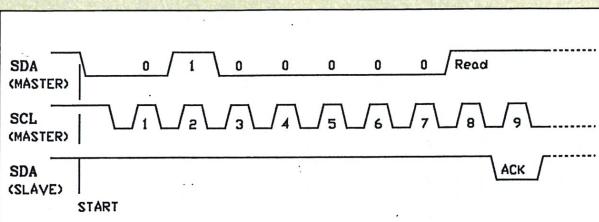
**Electrical and logic circuit of the SDA & SCL leads.** The wired-and connection allows each device to simultaneously monitor the bus while transmitting data. When a device transmits a logic-high it expects to see a logic-high on its input, if a logic low is received then another device is using the bus.



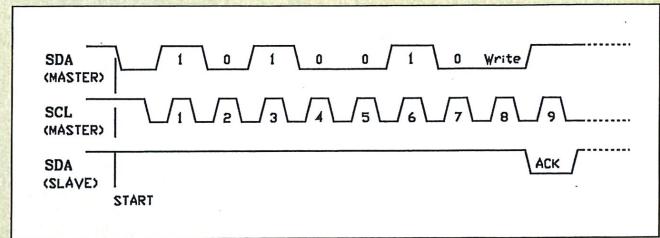
### Data transfer under I<sup>2</sup>C



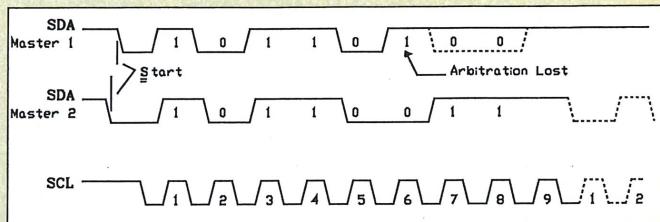
Start and stop conditions shows the relationship of the start, repeated start and stop conditions on the SDA lead with reference to the SCL Lead. The repeated start condition is used when a master needs to retain control of the bus during a combined write/read transfer, for example, when accessing a memory device.



Addressing a slave transmitter. Shows waveform to read from slave address 80. (8-bit I/O PCF8574). Note that the SDA lead is high (read) during the 8th SCL clock pulse.



Addressing a slave receiver. Shows waveform to write to slave address A0. (Eeprom PCF8582). Note that the SDA lead is low (write) during the 8th SCL clock pulse. The ACK signal, during the 9th SCL clock pulse is generated by the slave.



Arbitration. When a master sends a start condition it must check the bus for arbitration. The waveform shows two masters starting at the same time. The first master to send a logic-low on the SDA lead when the other master sends a logic-high wins the arbitration. In the waveform above master 1 is attempting to address slave 1011 010 and master 2 addresses slave 1011 001. Master 1 loses arbitration on clock pulse 6 and releases the bus. (Leaves the SDA lead high).

transmitting the signal, will receive a logic-low signal.

Both the clock lead (SCL) and the data lead (SDA) use this wired-and function to perform checks on data transfer. If the MASTER monitors its own transmitted signals it will expect to see the bus responding to these signals. The presence of another device on the bus can be detected if a logic-low is received when transmitting a logic-high. This feature is used to control the clock rate on the SCL lead and to obtain data arbitration on the SDA lead.

A slave can optionally control the clock pulses received from the master by holding the SCL lead at logic-low. Thus data speed and synchronisation of data exchange may be controlled by the slave device.

The transmitting device can check for the presence of other transmitters on the bus by monitoring the state of the SDA lead. If a logic-low is received when transmitting a logic-high then another device is also transmitting on the bus. This condition is known as lost arbitration. The bus specification requires that any master transmitter shall check the bus for arbitration and, if the presence of another transmitter is detected, the master shall relinquish any control of the bus.

#### Data transfer

When the I<sup>2</sup>C bus is idle both the SDA and SCL leads are high. A start condition is

#### SDA & SCL lead DC requirements

Parameter	Symbol	Standard Mode		Fast Mode		Unit
		Min	Max	Min	Max	
Low level input voltage	V <sub>IL</sub>	-	0.3V <sub>DD</sub>	-	0.3V <sub>DD</sub>	V
High level input voltage	V <sub>IH</sub>	0.7V <sub>DD</sub>	-	0.7V <sub>DD</sub>	-	V
Low level output voltage at 3mA sink current	V <sub>OL</sub>	0	0.4	0	0.4	V
at 6mA sink current		-	-	0	0.6	V
Input capacitance, each lead	C <sub>i</sub>			10	10	pF

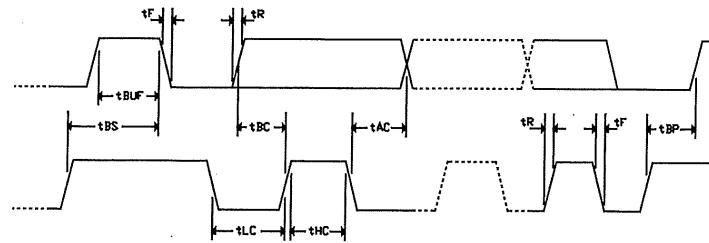
#### I<sup>2</sup>C bus line length limitations

Because of the non active pull-up feature of the wired-and bus, the capacitance on each of the bus wires restrict both the number of devices connected and the working distance. This capacitance comprises of the total input capacitance of the connected devices and the bus wire leakage capacitance. The minimum value of the pull-up resistor is defined by the maximum low level sink current of the devices. It may also be necessary to provide a resistor in series with each device to provide input protection against voltage spikes on the bus.

Data sheets for all of the Philips devices give information on how to calculate the pull-up and series resistor values for a given bus. To obtain maximum distance the pull-up resistor should be a minimum value, without series resistor. With a 5V system and 3mA maximum sink current the minimum value of the pull-up is 1.7kΩ (5.1V/3mA).

## SDA and SCL lead timing requirements.

Parameter	Symbol	Standard Mode		Fast Mode		Philips Symbol
		Min	Max	Min	Max	
SCL clock frequency	$f_{CL}$	0	100	0	400	kHz
Low period of SCL clock	$t_{LC}$	4.7	-	1.3	-	$t_{LOW}$
High period of SCL clock	$t_{HC}$	4.0	-	0.6	-	$t_{HIGH}$
Bus free time between stop & start condition	$t_{BUF}$	4.7	-	1.3	-	$t_{BUF}$
Time SCL must be high before start or repeated start	$t_{AS}$	4.7	-	0.6	-	$t_{SU.STA}$
Hold time SCL must be high after start or repeated start	$t_{BS}$	4.0	-	0.6	-	$t_{HD.STA}$
Time SDA must be stable before rising edge of SCL	$t_{BC}$	300	-	300	-	$t_{HD.DAT}$
Time SDA must be stable after falling edge of SCL	$t_{AC}$	250	-	100	-	$t_{SU.DAT}$
Time SDA must be low after a rising edge on SCL prior to a stop (rising edge on SDA)	$t_{BP}$	4.0	-	0.6	-	$t_{SU.STO}$
Rise time of both SDA and SCL signals	$t_R$	0	1000	0	300	ns
Fall time of both SDA and SCL signals	$t_F$	0	300	0	300	ns
Capacitance load for each line	$C_b$	0	400	0	400	pF



Conventional I<sup>2</sup>C devices handle clock signals and bit rates to 100kHz but fast mode devices are now appearing that are capable of working at 400kHz. These timings cover both fast and slow modes.

defined as a falling edge on the SDA lead when SCL is high. A stop condition is defined as a rising edge on SDA when the SCL is high. It follows that to avoid false start and stop conditions being generated during data transfer, the state of the SDA lead must be stable while the SCL lead is high.

The generation of a start condition indicates to all other devices that the bus is busy until a stop condition is generated. masters wait for this stop condition before attempting to send a start. All slaves, on detecting a start, will reset their hardware and prepare to receive the slave address. A slave recognising its own address will generate an ACK signal.

The ACK signal is a logic-low signal during the ninth clock pulse. The NACK signal is, therefore, a logic-high. The generation of a non-existent slave device address automatically generates a NACK because the bus is inherently in the high state.

It is possible that two masters could simultaneously generate a start followed by a slave address. For this reason all masters must always check the SDA lead for arbitration. As two or more masters could attempt to address the same slave, the check for arbitration must continue for the whole of the data transfer. (Until the stop condition is generated.)

### Slave addressing

All slave devices are designed with a unique address which, when recognised and accepted, sets the slave in data transfer mode. There are two modes of addressing, both using the same protocol. The "standard" seven bit address is used by most of the devices available at present. The ten bit address mode will be provided on some future devices.

To address a slave device, a master will generate a start followed by a nine bit word. This

word comprises seven data bits (ADDRESS), a read/write (W) bit and an acknowledge (ACK) bit. The read/write bit determines the data direction. A logic-high (read) sets the slave as a transmitter, a logic-low (write) makes the slave a receiver.

If an addressed slave device is capable of responding to the master it will generate an ACK signal (a low level on the SDA lead during the ninth clock pulse) and set its internal hardware or software for the data transfer. Any slave not addressed will ignore any further action on the bus until another start condition is generated.

The seven bit address has several reserved codes used for special purposes.

### Slave address byte

	Bit no.
Allocation	6543 210 W
General call	0000 000 0
Start byte	0000 000 1
CBUS	0000 001 X
Reserved	0000 1XX X
10 bit addressing	1111 0XX X
Reserved	1111 XXX X

X = any state

### Data transfer

All currently available devices perform data transfer in a 9-bit word comprising eight data bits plus a ninth ACK bit. An astute reader will observe that the I<sup>2</sup>C bus protocol does not necessarily require an eight bit format for addressing and data transfer. Provided that a master is capable of generating the clock pulses and the slave is configured to receive them, then the word format can be any number of bits. Early bus formats, particularly systems using the

8048 type microcontroller, were open and allowed the user to choose the word length.

Transmitters send data on the SDA lead, receivers read data from the SDA lead and generate the ACK signal. Masters generate clock pulses on the SCL lead and control the bus by generating start and stop conditions.

A data transfer can be of any number of data words. The transfer is terminated when a (repeated) start or a stop condition is sent by the master. The bus is considered to be busy during the period between an initial start condition and a stop condition. A receiver can indicate that the transfer is over by sending a NACK signal but it is the responsibility of the master to send a stop.

### General calls

The slave address 0000 0000 (a write to slave address 00) is reserved for a general call to devices that require "broadcast" information. The second byte of the transfer will indicate what type of information is being transmitted. General calls are used to globally set slaves to a defined state or to send global configuration data. A full discussion is beyond the scope of this article. Interested readers should obtain the relevant Philips data sheets.

### Other modes

**Low speed mode.** This mode is an extension of the bus protocol to allow relatively slow slave devices to respond to a "normal" master using an optional lower clock rate, preceded by a longer start procedure. The start procedure is as follows:

- A standard start condition.
  - A start-byte 0000 0001. (This is equivalent to "read address 0")
  - A repeated START condition.
- The start-word is seven clock pulses long

## Putting in an extra feature

We had a requirement to add an auxiliary keypad to one of our existing designs. This product used a PCB80C552 micro controller with I<sup>2</sup>C bus software drivers already installed (clock timer and a led display). Expecting future enhancements and modifications we arranged the original circuit layout such that all spare '552 port leads were made accessible on suitable connecting points.

The PCF8574, a remote 8-bit i/o expander, has the necessary functions. It is an 8-bit quasi-bidirectional port similar in function to the 8051 microcontroller ports. It has an interrupt facility which is activated when the input to one or more of the port leads changes state. The interrupt signal is cleared when a bus read or write is sent to the device. There are two versions of the PCF8574; one version has an allocated slave address 0100 XXX, the other (PCF8574A) 0111 XXX.

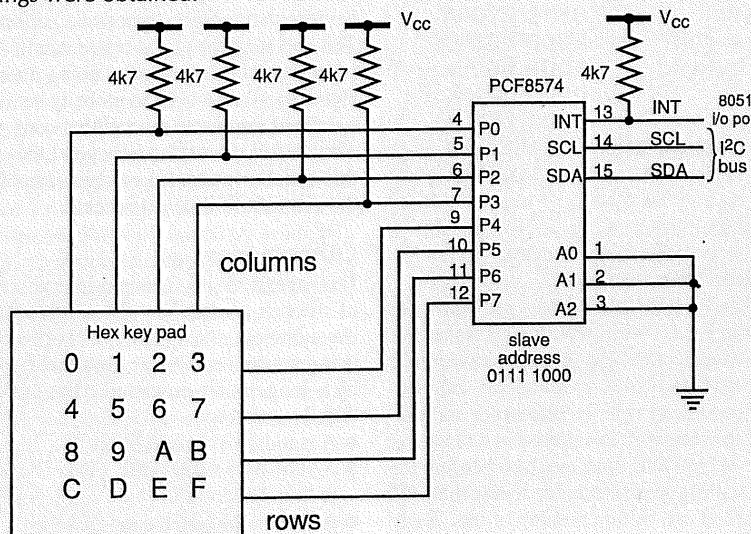
The PCF8574 was mounted on a small daughter board attached to the hex keypad. The eight wires from the keypad were connected to the device ports. The 5V supply, ground, SCL, SDA and interrupt leads were wired to a suitable connector. Hardwire links set the address to 0100 000. In the idle (waiting for a key depression) state the PCF8574 port bits 4-7 are set to binary 0000, which applies a ground potential to the four row pins on the keyboard. Bits 0-3 are set to binary 1111. When a key is pressed one of the column pins (bits 0-3) is pulled to ground via the key connection. This change of input causes the internal logic in the PCF8574 to apply a ground potential on the "int" pin 13 which is detected by the microcontroller. The software then performed the following I<sup>2</sup>C bus transfer.

Slave address		Transferred data			
	bit no.		bit no.		
S	6543 210	W	7654 3210		
R	0100 000	0	A	1110 1111	A start & select column "0"
R	0100 000	1	A	1110 KKKK	A Read state of keys 048C
R	0100 000	0	A	1101 1111	A Select Column "1"
R	0100 000	1	A	1101 KKKK	A Read state of keys 159D
R	0100 000	0	A	1011 1111	A Select Column "2"
R	0100 000	1	A	1011 KKKK	A Read state of keys 26AE
R	0100 000	0	A	0111 1111	A Select Column "3"
R	0100 000	1	A	0111 KKKK	A Read state of keys 37BF
R	0100 000	1	A P	0000 1111	Set "idle" state and STOP

S start; P stop sent or R repeated start; A ack; W read/write 1=read, 0= write

All of the bits (K) in the received data bits 0-3 will be logic-high except for the bit(s) corresponding to the pressed key(s). Note that only one stop condition is sent. The repeated start feature was used to prevent other masters from interfering, and thus delaying, the bus transfer. When the key is released the PCF8574 will detect another change on its inputs and present a further signal on the "int" pin. The software must now perform another read or write to the device to clear this signal.

Initial concern whether the bus would be fast enough to detect and process the key depressions was soon dispelled by a calculation. With a baud rate of 100kHz and nine bits required for each data byte and with a total of 18 bytes, the maximum time to scan the keyboard was  $9 \times 18 / 100 = 1.6\text{ms}$ . This time was less than the contact bounce period of the keys and appropriate software delay routines were needed to insure that valid readings were obtained.



which, combined with a slower clock rate, gives ample time for microprocessor to respond and prepare for data transfer. After the repeated start condition the "real" slave address is sent. No slave device is allowed to acknowledge this start byte.

**Fast mode.** In fast-mode the I<sup>2</sup>C bus protocol remains unchanged. The maximum baud rate has been increased to 400kHz thus tightening the timing specification for the SDA and SCL leads. Devices designed for the fast-mode will still perform satisfactorily at standard-mode baud rates.

**Ten bit slave addresses.** The 10-bit addressing has been introduced because most of the 112 addresses allowed by the 7-bit scheme have been allocated more than once. The bus protocol and byte length remain the same. The reserved slave address 1111 0XX is used to provide an extra two bits for the address. The remaining eight bits are sent in the next byte. Full details of this mode are outside the scope of this article and, as I understand from Philips, there are no devices yet using this mode. Further information can be obtained from Philips Components and the handbook "I<sup>2</sup>C Peripherals for Microcontrollers" gives full details on this – at present fairly academic – mode.

A future article will deal with practical applications such as the use of an I<sup>2</sup>C bus controller to adapt an existing micro system and a list of available devices. ■

## Bibliography

- Philips Components Data Handbooks:
- I<sup>2</sup>C bus specification
- I<sup>2</sup>C bus compatible ICs, Book 4 Parts 12a & 12b 1989
- I<sup>2</sup>C Peripherals for Microcontrollers, 1992
- Single Chip 8-bit Microcontrollers PCB83C552
- Integrated Circuits Designers Guide.

Help and further information may be obtained from the author at TDR Ltd. Tel 0666 577464.

## Acknowledgements

The author is grateful for the help and information, given over several years, from Philips Components, Gothic Crellon and Quarndon Ltd.