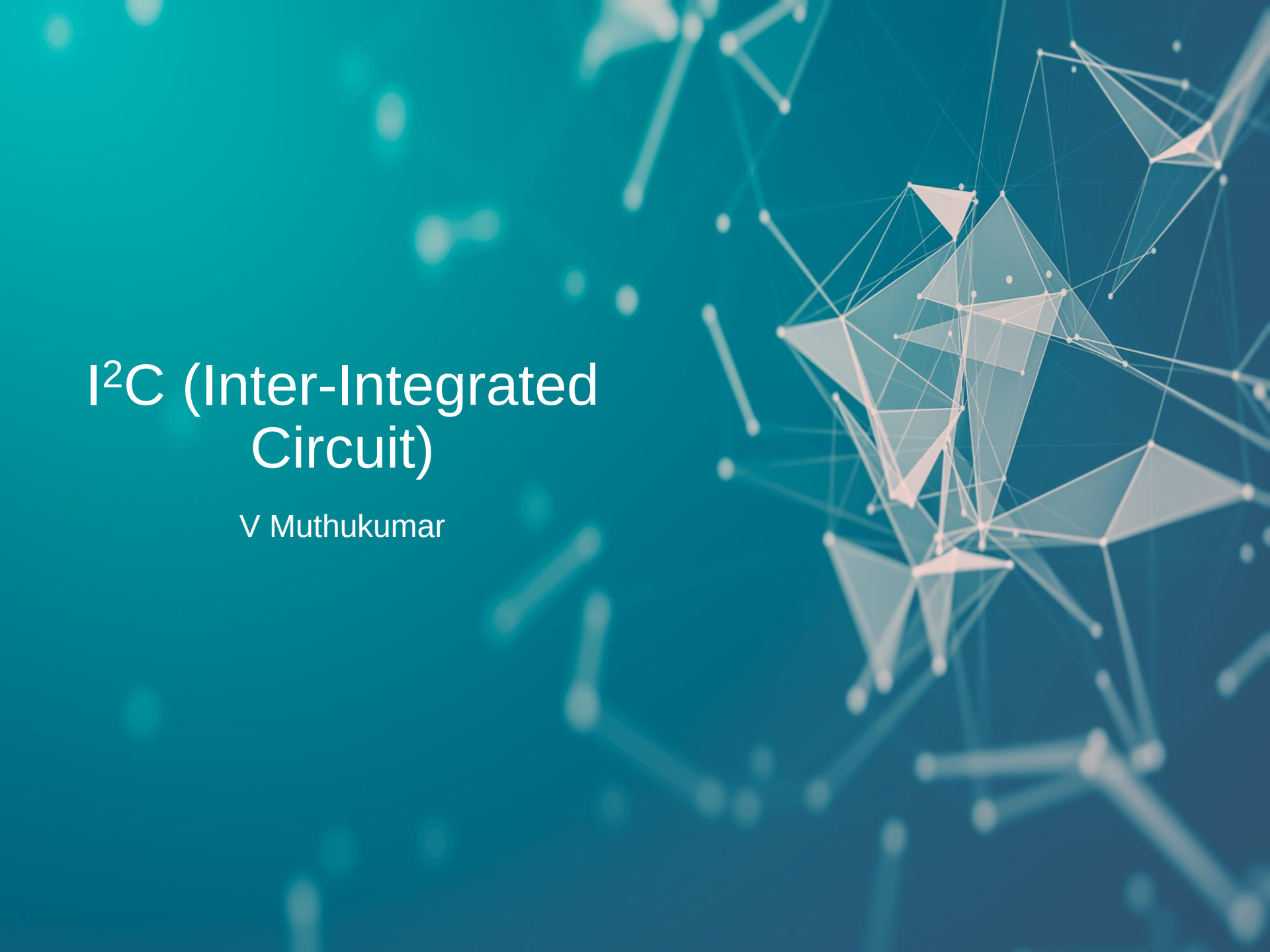
I^2C (Inter-Integrated Circuit)

V Muthukumar



Outline

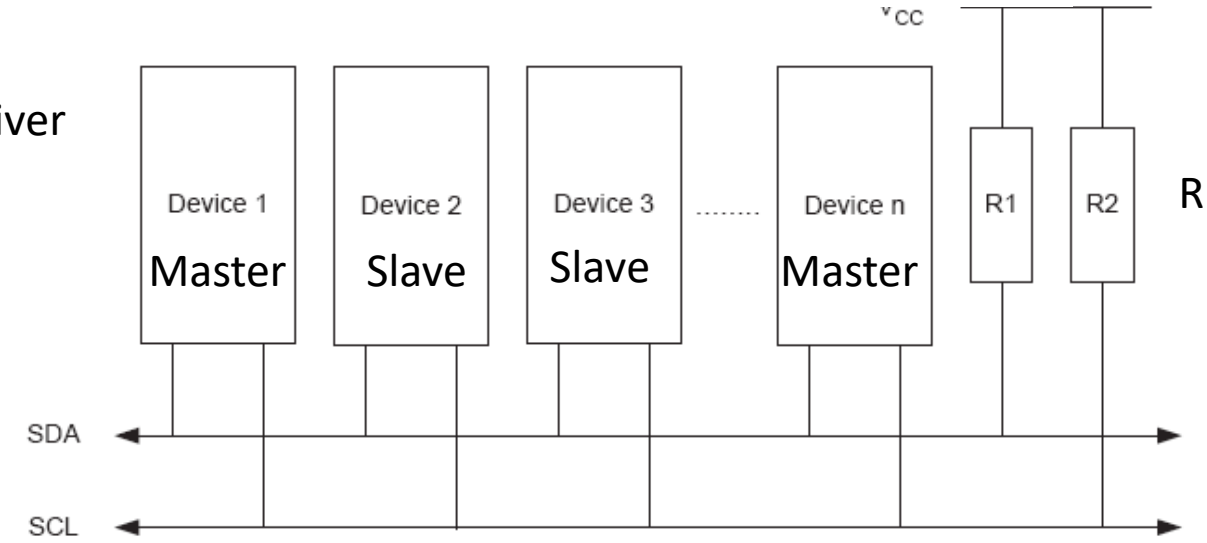
- In this module, the students will learn the following
  - Understand the fundamentals of the I^2C interface
  - Read and write data to/from an I^2C device

I²C

- started by Philips (aka TWI/Wire)
- Synchronize the data transfer between two chips
- SDA (Serial Data -Bidirectional), SCL (Serial Clock)
- About 128 devices (nodes) can share a bus

Slave = Transmitter when sending address or data to the master
Master = Receiver when receiving address or data from the slave

I2C Trans-receiver
Half-duplex



R1=R2=10k

I2C …

- Master generates and transmits CLK
- When the Master transmits the Slave can only ACK.
- ATmega328p (Master) & Sensor (Slave)

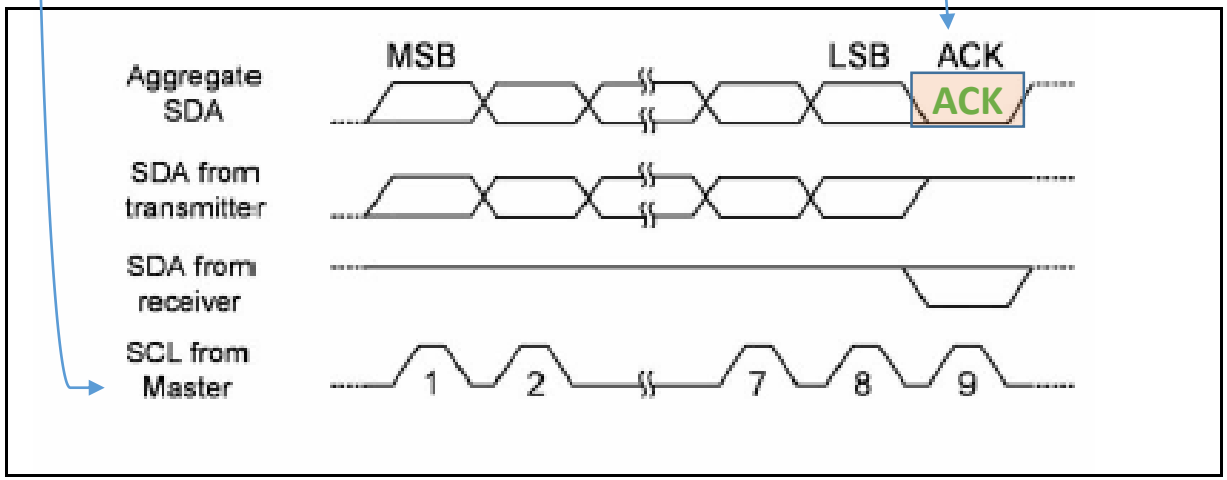


Figure 16-1. I2C

Bit Format

- Each data bit transferred on the SDA line is synchronized by the clock on the SCL line.
- Data line can change only when the clock line is low. Stop and Start conditions are exceptions.

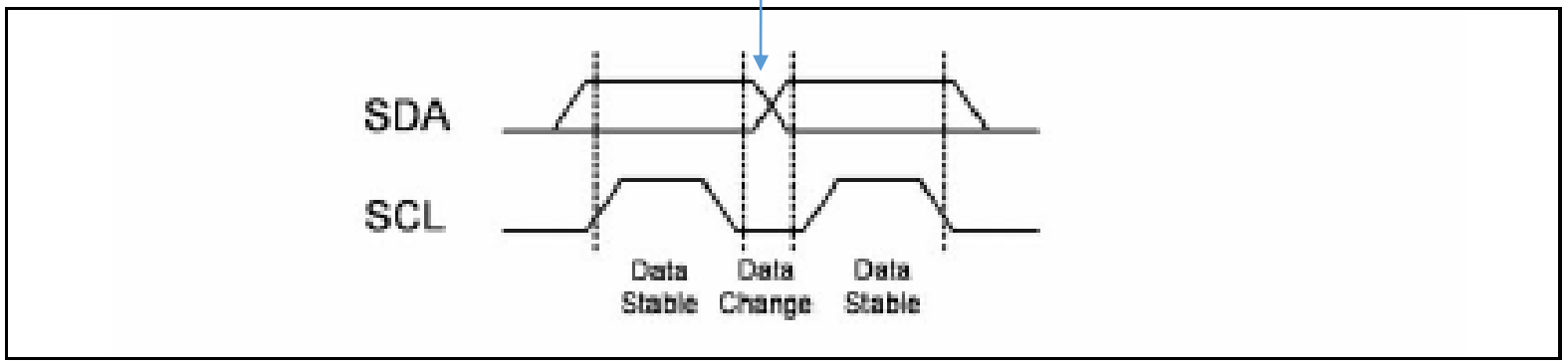


Figure 16-1. I2C

Start and Stop conditions

- START and STOP conditions – data change when the CLK is high.
- START and STOP conditions are generated by the master.

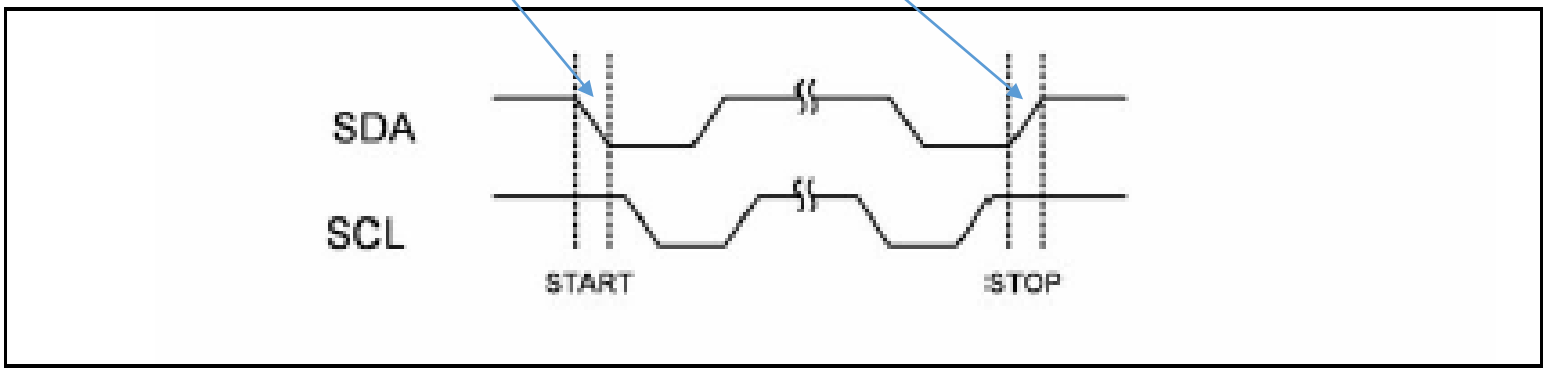


Figure 16-1. I2C

Repeated Start

- If a master, which has the control of the bus, wishes to initiate a new transfer and does not want to release the bus before starting the new transfer, it issues a new START condition (REPEAT START) between a pair of START and STOP conditions.

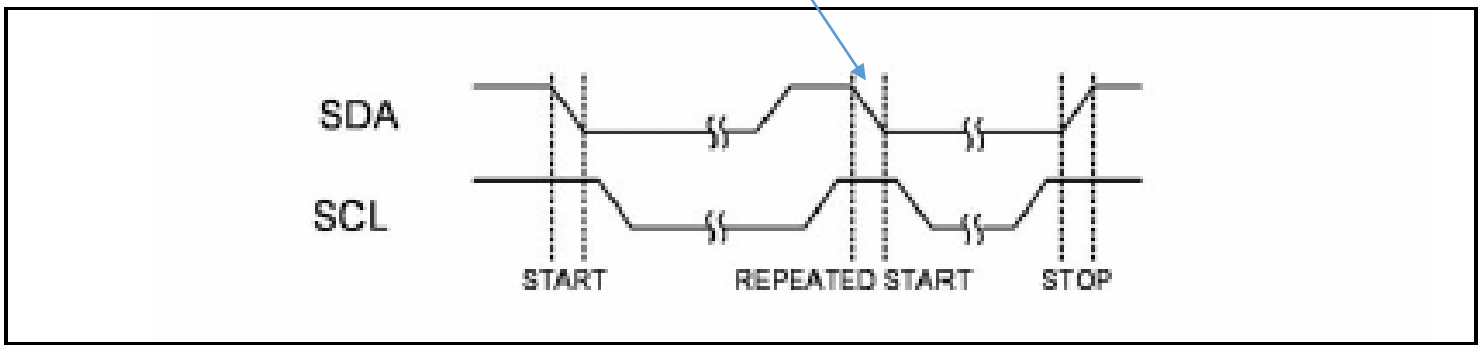


Figure 16-1. I2C

Packet Format

- Packet Types: Address Packet and Data Packet
- Each packet is 9 bits long.
- First 8 bits are put on SDA by the transmitter
- 9th bit is an acknowledge by the receiver
  - NACK (leave high) or ACK (pull down)

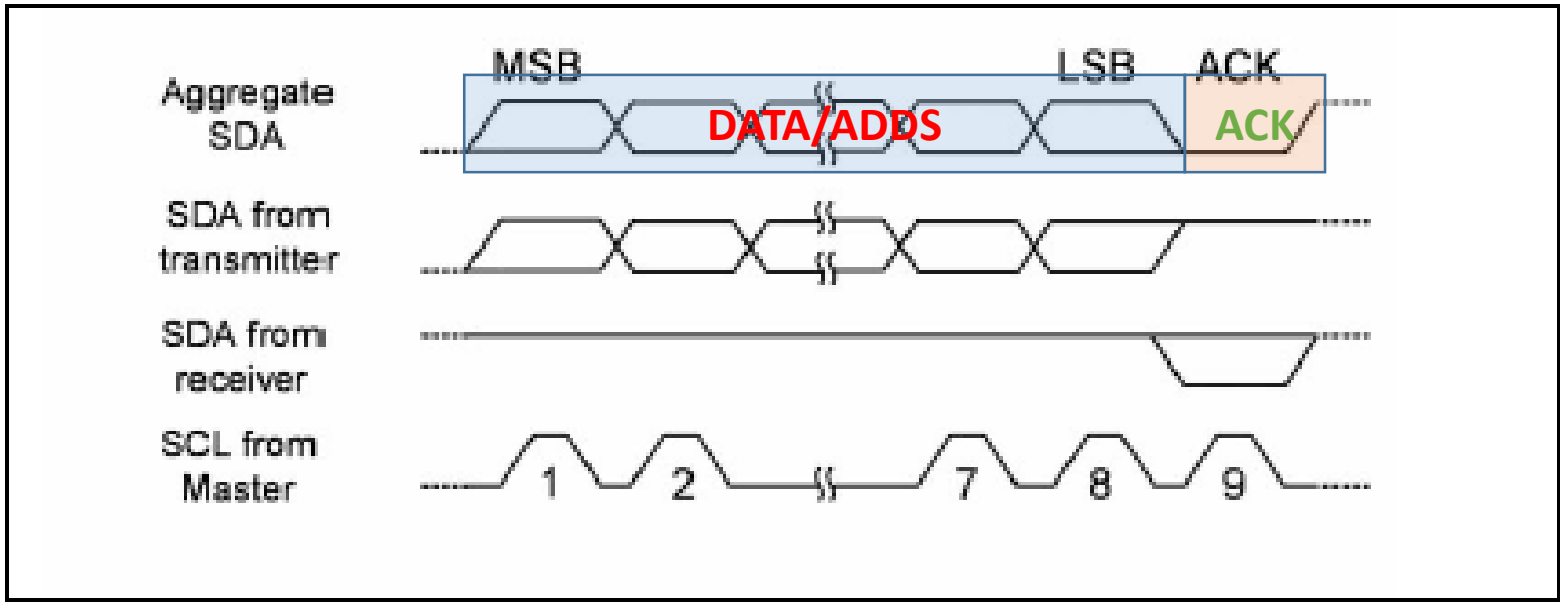
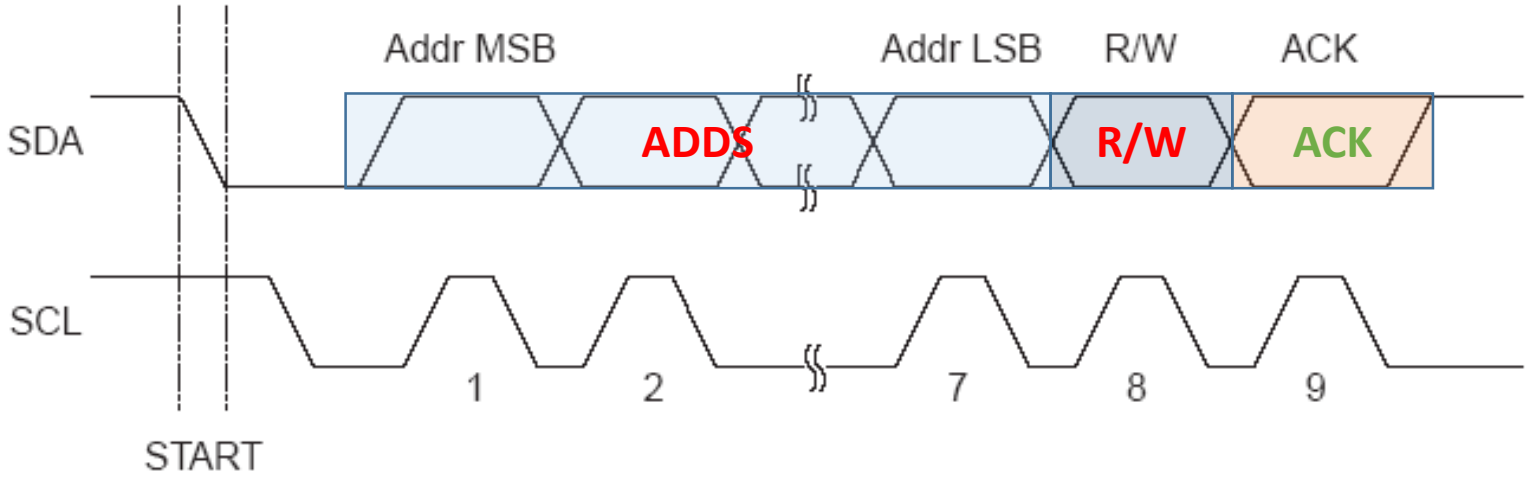


Figure 16-1. I2C

Address packet

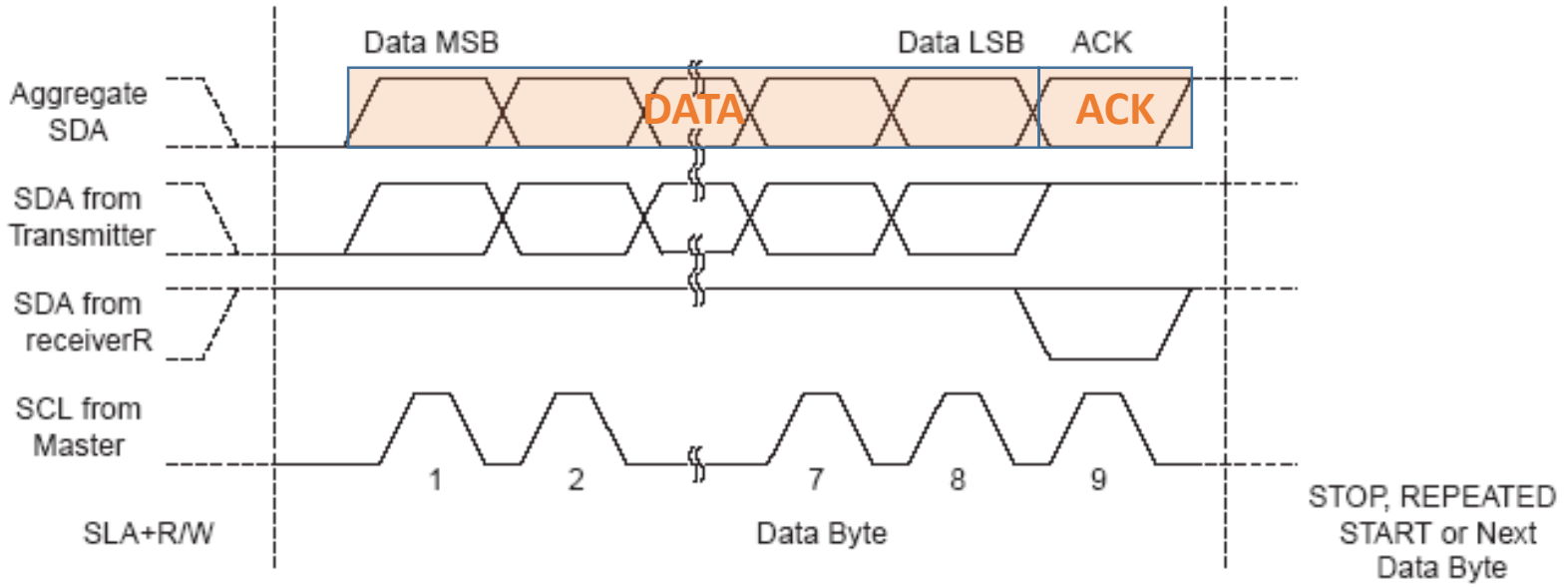
- Started by a Start Condition
- Master always selects the Slave (addresses the slave)
- 7 bits address, MSB first
- 8th bit define R/W operation
- 9th bit is the ACK or NACK

Figure 79. Address Packet Format



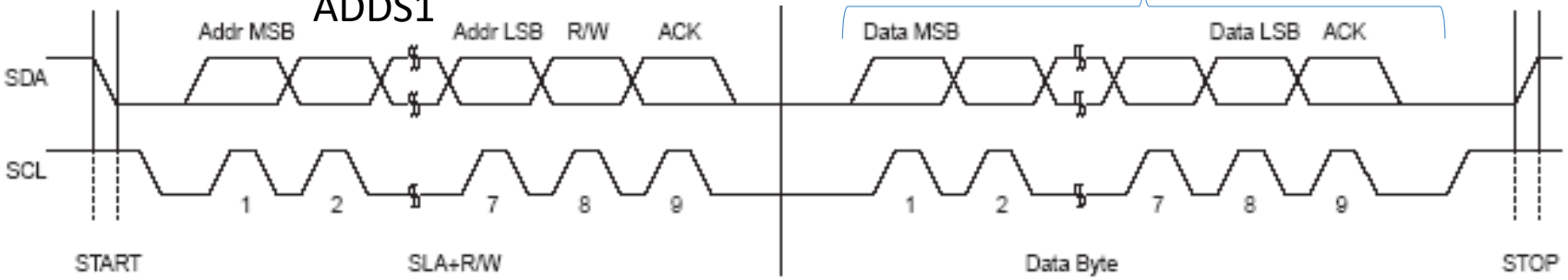
Data packet format

- Can be transmitted by either Master or Slave
- Is 8 bit long (MSB first)
- Followed by 9th bit, ACK/ NACK from receiver (slave or master)
- NACK does not necessarily indicate error

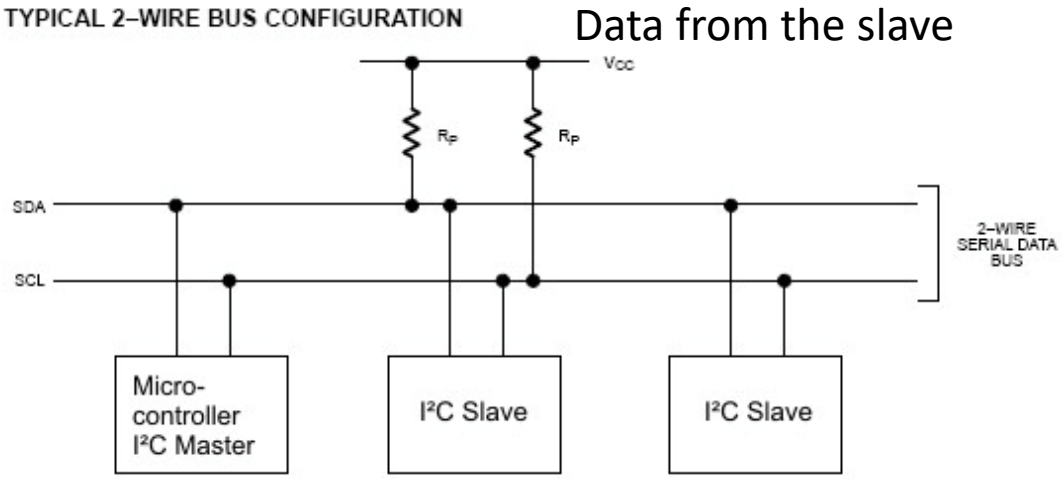


Typical data transmission

Will be repeated any # of times



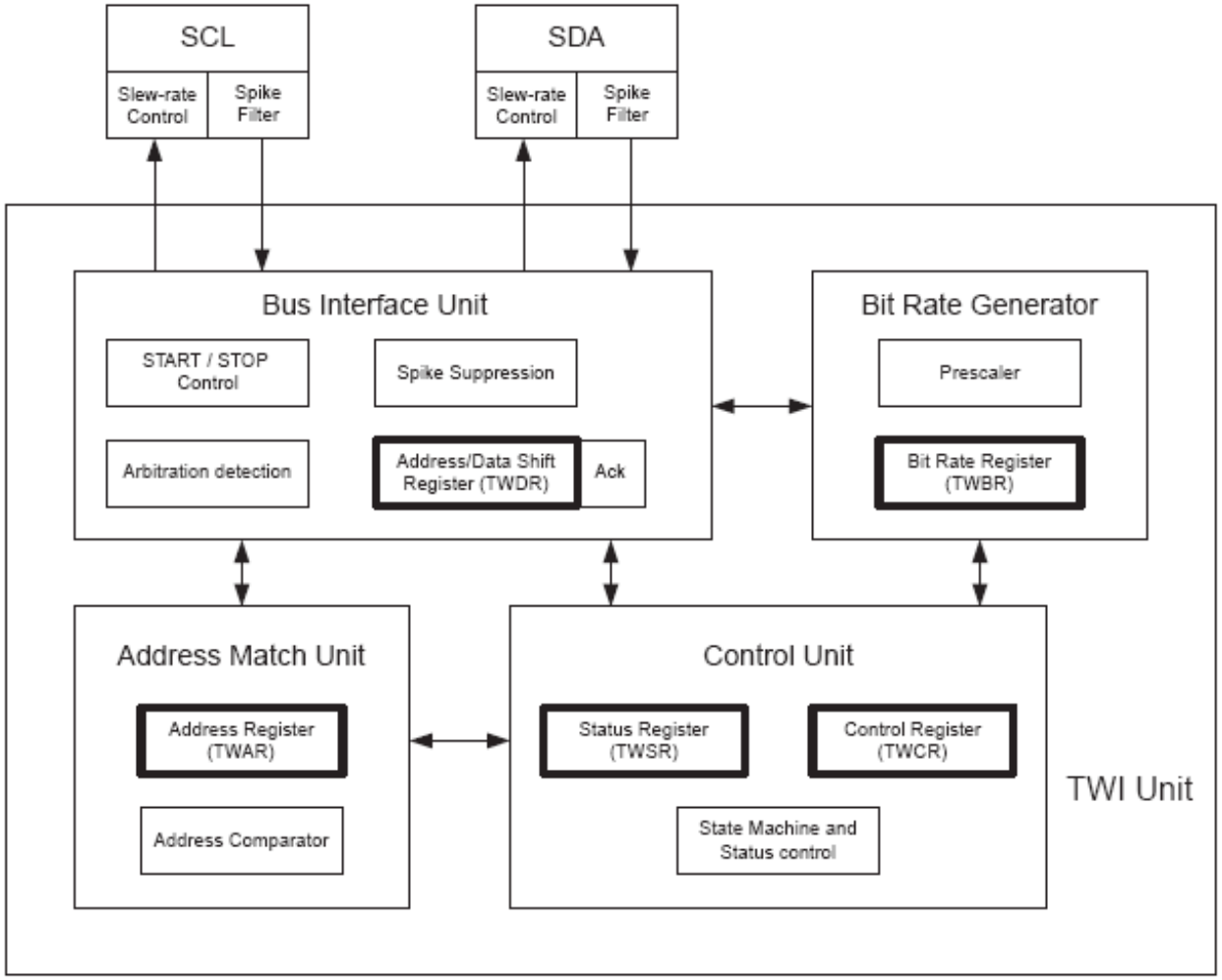
Command from the master or
Data from the slave



SENSOR#1
ADDS1

SENSOR#2
ADDS2

I²C unit in AVR



I²C Registers in AVR

- **TWI Status Register (TWSR)**

TWS7	TWS6	TWS5	TWS4	TWS3	-	TWPS1	TWPS0

Bits 7..3 – TWS: TWI Status
These five bits show the status of the TWI control and bus.

Bits 1..0 – TWPS: TWI Prescaler Bits
These bits control the bit rate prescaler.

Figure 18-12. TWSR: TWI Status Register

- **TWI Bit Rate Register (TWBR)**

TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot 4^{TWPS}}$$

TWAR (TWI Address Register)

Bit	7	6	5	4	3	2	1	0	
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	0	

- TWA6-0 (TWI slave Address)
- TWGCE (TWI General Call Recognition Enable bit)
  - 1: Answer to general call
  - All devices on the I2C bus are addressed by using the general call address. When the general call address is used, all device on the I2C bus which receive a message with this address should (at least theoretically) confirm the reception by the ACK bit. These are usually all '0' with the direction bit R/W=0.

TWI Control Register

Bit	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- TWINT (TWI Interrupt Flag)
- TWEA (TWI Enable Acknowledge Bit)
  - 1:ACK, 0:NACK
- TWSTA (TWI Start condition bit)
- TWSTO (TWI Stop condition bit)
- TWWC (TWI Write Collision flag)
- TWEN (TWI Enable bit)
- TWIEN (TWI Interrupt Enable)

TWI, Master Mode programming

- Initializing
  - Set the TWI module clock frequency by setting the values of the TWBR register and the TWPS bits in the TWSR register.
  - Set the TWEN bit in TWCR to one to enable the TWI module

```
#define I2C_SCL_CLOCK 100000L
TWBR = ((F_CPU/I2C_SCL_CLOCK)-16)/2;

TWCR = _BV(TWEN)
```

- Transmit START condition
  - Set TWEN, TWSTA, and TWINT bits of TWCR to one.

```
/* Send START condition */
TWCR = _BV(TWINT) | _BV(TWEN) | _BV(TWSTA);
```

TWI, Master Mode programming

- Send Address
  - Copy the Address byte to the TWDR
  - Set the TWEN and TWINT bits of the TWCR to one to start sending the byte.
  - Poll TWINT flag in TWCR register to see whether the byte transmitted completely & check for ACK.

```
/* Send device address */
TWDR = address | mode;
TWCR = _BV(TWINT) | _BV(TWEN);
/* Wait until transmission completed and ACK/NACK has been received */
I2C_WAIT_CLEAR(TWCR, TWINT);
/* Check value of TWI Status Register. */
twst = TW_STATUS;
if ( (twst != TW_MR_SLA_ACK) ) return twst;
```

TWI, Master Mode programming

- Send Data
  - Copy the data byte to the TWDR
  - Set the TWEN and TWINT bits of the TWCR to one to start sending the byte.
  - Poll TWINT flag in TWCR register to see whether the byte transmitted completely & check for ACK.

```
/* Send data to the previously addressed device */
TWDR = data;
TWCR = _BV(TWINT) | _BV(TWEN);
/* Wait until transmission completed */
I2C_WAIT_CLEAR(TWCR, TWINT);
```

```
/* Check Acknowledgement */
uint8_t twst;
/* Check value of TWI Status Register. */
twst = TW_STATUS;
if(twst != TW_MT_DATA_ACK)
 return twst;
```

TWI, Master Mode programming

- Receive Data
  - Set TWEN and TWINT bits of TWCR to one to start receiving a byte. [optional TWEA for Automatically send acknowledge bit].
  - Poll TWINT flag in TWCR to see whether a byte has been received completely
  - read the received byte from the TWDR

```
/* Read one byte from the I2C device*/
TWCR = _BV(TWINT) | _BV(TWEN) | _BV(TWEA);
I2C_WAIT_CLEAR(TWCR, TWINT);
return TWDR;
```

- Transmit STOP condition
  - Set TWEN, TWSTO, and TWINT bits of TWCR to one
    - Note: we cannot poll the TWINT flag after transmitting the STOP condition

```
/* Send STOP condition. */
TWCR = _BV(TWINT) | _BV(TWEN) | _BV(TWSTO);
```

TWI, Slave Mode programming

- Initializing
  - Set the slave address by setting the values for the TWAR register.
    - 7 bits for address
    - 8th bit is TWGCE (1 = answer general calls)
  - Set the TWEN bit in TWCR to one to enable the TWI module
  - Set the TWEN, TWINT, and TWEA bits of TWCR to one to enable the TWI and acknowledge generation

```
TWAR = address | (gcall & 0x01);
TWAMR = address_mask;
TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWINT) | _BV(TWEA);
```

TWI, Slave Mode programming

- Listening
  - poll the TWINT flag to see when the slave is addressed by a master device or use its interrupt.

```
/* Wait until transmission completed and ACK/NACK has been received */
I2C_WAIT_CLEAR(TWCR, TWINT);
```

- Send Data
  - Copy the data byte to the TWDR
  - Set the TWEN, TWEA, and TWINT bits of the TWCR register to one to start sending the byte.
  - Poll TWINT flag in TWCR register to see whether the byte transmitted completely

```
/* Send data to the previously addressed device */
TWDR = data;
TWCR = _BV(TWINT) | _BV(TWEN);
/* Wait until transmission completed */
I2C_WAIT_CLEAR(TWCR, TWINT);
```

TWI, Slave Mode programming

- Receive Data
  - Set TWEN and TWINT bits of TWCR to one to start receiving a byte. [optional TWEA for Automatically send acknowledge bit].
  - Poll TWINT flag in TWCR to see whether a byte has been received completely
  - read the received byte from the TWDR

```
/* Read one byte from the I2C device*/
TWCR = _BV(TWINT) | _BV(TWEN) | _BV(TWEA);
I2C_WAIT_CLEAR(TWCR, TWINT);
return TWDR;
```

I2C Master - C

```
#include <avr/io.h>

void i2c_write(unsigned char data)
{
   TWDR = data ;
   TWCR = (1<< TWINT)|(1<<TWEN);
   while ((TWCR & (1 <<TWINT)) == 0);
}

//*************************************************************

void i2c_start(void)
{
   TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
   while ((TWCR & (1 << TWINT)) == 0);
}

//*************************************************************

void i2c_stop()
{
   TWCR = (1<< TWINT)|(1<<TWEN)|(1<<TWSTO);
}
```

I2C Master – write C

```
void i2c_init(void)
{
   TWSR=0x00;                    //set prescaler bits  to zero
   TWBR=0x47;                    //SCL frequency is 50K for XTAL = 8M
   TWCR=0x04;                    //enable the TWI module
}

//*************************************************************************

int main (void)
{
   i2c_init();
   i2c_start();                  //transmit START condition
   i2c_write(0b11010000);        //transmit SLA + W(0)
   i2c_write(0b11110000);        //transmit data
   i2c_stop();                   //transmit STOP condition
   while(1);                     //stay here forever
   return 0 ;
}
```

Address of SLA=0xD0

Transmit data 0xF0 to slave

I2C Master - read - C

```
#include <avr/io.h>

void i2c_init(void)
{
   TWSR=0x00;                    //set prescaler bits  to zero
   TWBR=0x47;                    //SCL Frequency is 50K for XTAL=8M
   TWCR=0x04;                    //enable the TWI module
}
//*************************************************************
void i2c_start(void)
{
   TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
   while ((TWCR & (1 << TWINT)) == 0);
}
//*************************************************************
void i2c_write(unsigned char data)
{
   TWDR = data ;
   TWCR = (1<< TWINT)|(1<<TWEN);
   while ((TWCR & (1 <<TWINT)) == 0);
}
```

I2C Master - read - C

```
unsigned char i2c_read(unsigned char isLast)
{
   if (isLast == 0)           //if want to read more than 1 byte
     TWCR = (1<< TWINT)|(1<<TWEN)|(1<<TWEA);
   else                       //if want to read only one byte
     TWCR = (1<< TWINT)|(1<<TWEN);
   while ((TWCR & (1 <<TWINT)) == 0);
   return TWDR  ;
}
//*************************************************************
void i2c_stop()
{
   TWCR = (1<< TWINT)|(1<<TWEN)|(1<<TWSTO);
}
```

I2C Master - read - C

```
int main (void)
{
   unsigned char i = 0 ;
   DDRA = 0xFF;                    //Port A is output
   i2c_init();                     //initialize TWI for master mode
   i2c_start();                    //transmit START condition
   i2c_write(0b11010001);          //transmit SLA + R(1)
   i=i2c_read(1);                  //read only one byte of data
   PORTB= i;                       //show the byte on Port B
   i2c_stop();                     //transmit STOP condition
   while(1);                       //stay here forever
   return 0 ;
}
```

Address of SLA=0xD0

Summary

- On completion of this module student should be able to
  - Understand I^2C module in AVRs
  - Program using assembly and C program to store and retrieve data in/from an I^2C device