TWI

(I^2C)

Two-Wire Serial Interface

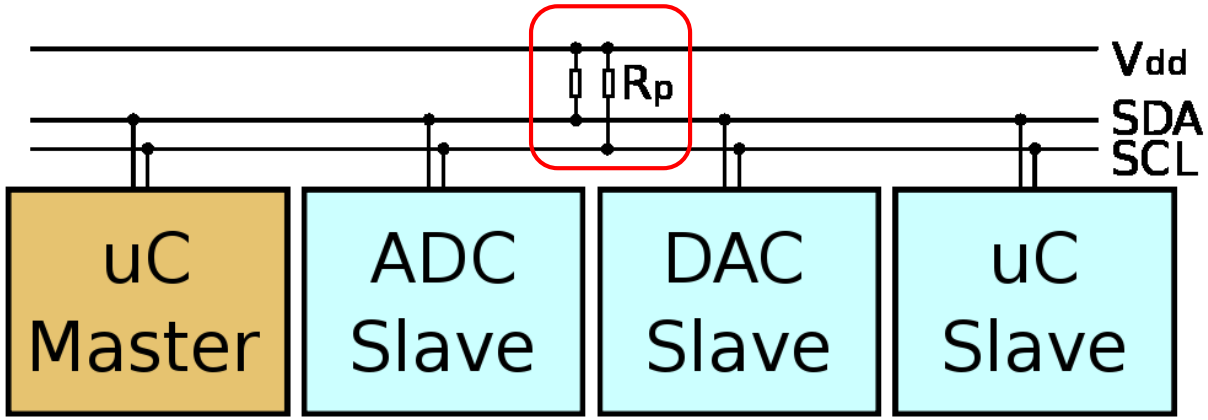
(Inter-Integrated Circuit)

I²C and TWI (1)

- I²C (Inter-Integrated Circuit), pronounced I-squared-C, is a multi-master, multi-slave, single-ended, serial computer bus.
- It is invented by Philips Semiconductor (now NXP Semiconductors).
- It is typically 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).
- TWI (Two Wire Interface) is essentially the same bus implemented on various system-on-chip processors from Atmel.
- In some cases, use of the term TWI indicates incomplete implementation of the I²C specification.
  - Not supporting arbitration or clock stretching is one common limitation, that is still useful for a single master communicating with simple slaves that never stretch the clock.

I²C and TWI (2)

- Consists of
  - SDA: Serial Data
  - SCL: Serial Clock



ATmega328PB TWI Features

- Simple, Powerful and Flexible Communication Interface with only two Bus Lines
- Both Master and Slave operation supported
- Device can operate as Transmitter or Receiver
- 7-bit Address Space allows up to 128 different Slave Addresses
- Multi-master Arbitration support
- Up to 400kHz Data Transfer Speed
- Slew-rate limited output drivers
- Noise Suppression Circuitry rejects spikes on Bus Lines
- Fully programmable Slave Address with General Call support
- Address Recognition causes Wake-up when AVR is in Sleep Mode
- Compatible with Philips' I2C protocol
- Two TWI instances `TWI0` and `TWI1` (ATmega328P has one TWI only, `TWI0`)

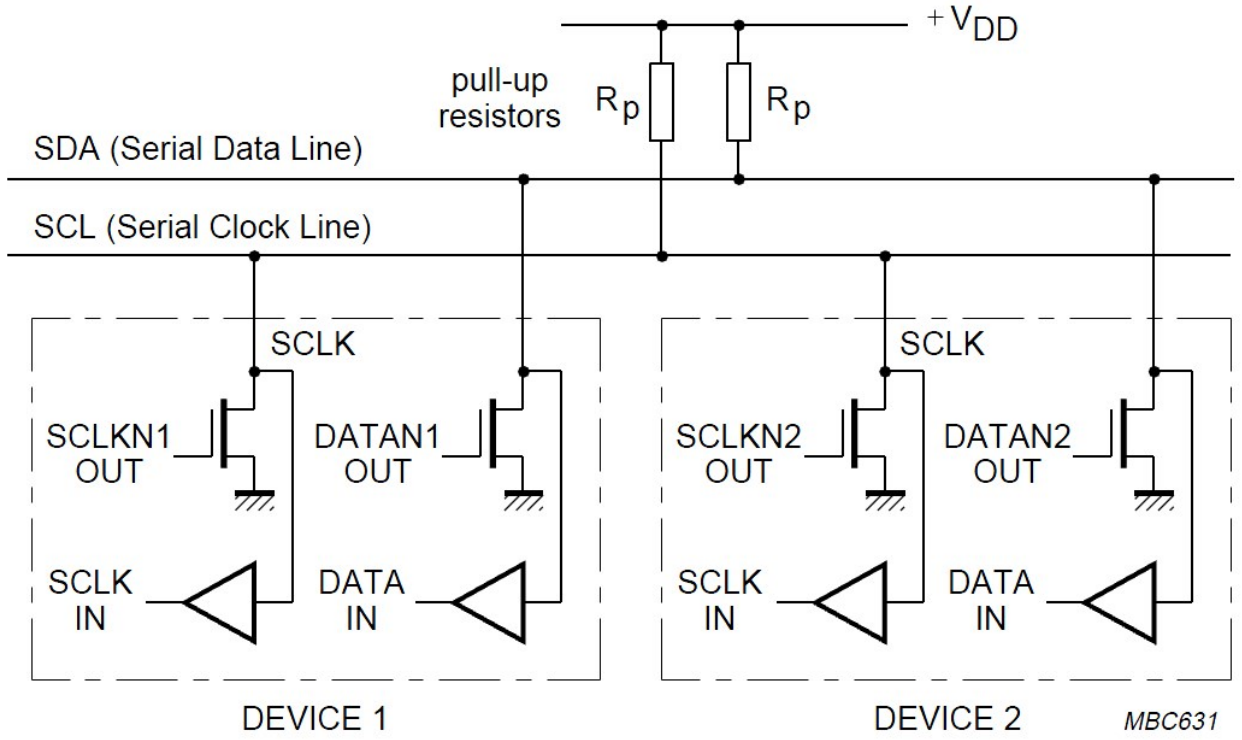
TWI Terminology

Term	Description
Master	The device that initiates and terminates a transmission. The Master also generates the SCL clock.
Slave	The device addressed by a Master.
Transmitter	The device placing data on the bus.
Receiver	The device reading data from the bus.

The Power Reduction TWI bit in the Power Reduction Register (PRRn.PRTWI) must be written to '0' to enable the two-wire Serial Interface.

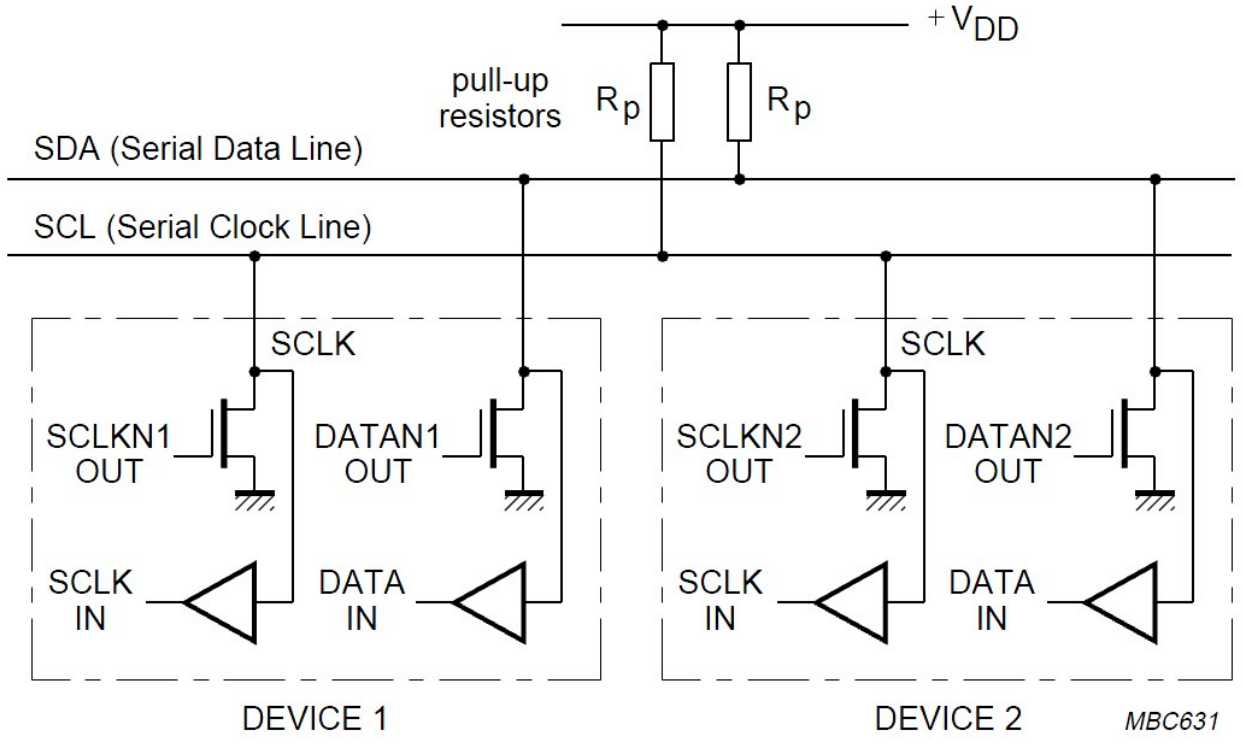
TWI Electrical Interconnection (1)

- Both bus lines are connected to the positive supply voltage through pull-up resistors.
- The bus drivers of all TWI-compliant devices are open-drain or open-collector. This implements a wired-AND function which is essential to the operation of the interface.



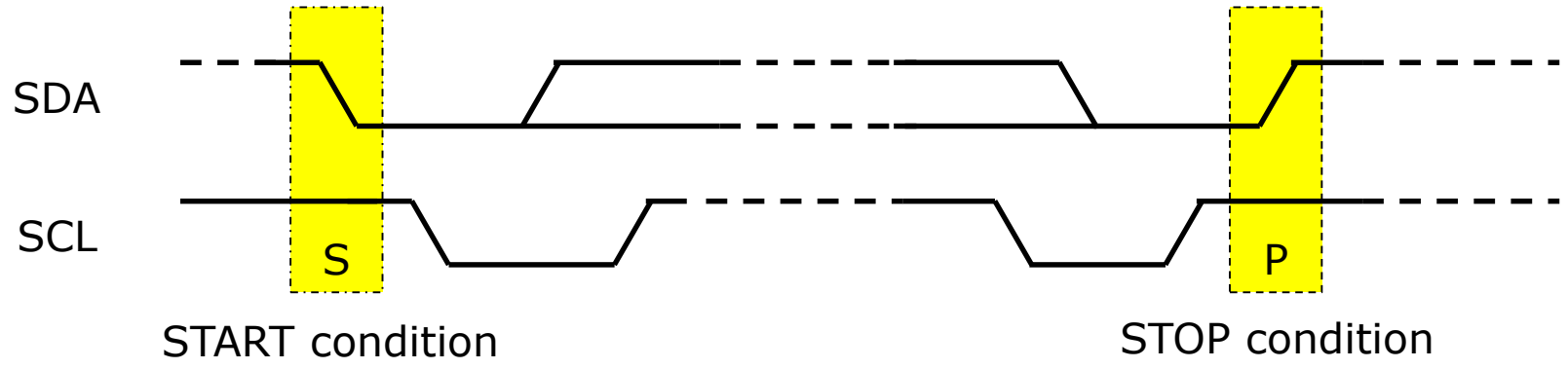
TWI Electrical Interconnection (2)

- A low level on a TWI bus line is generated when one or more TWI devices output a zero.
- A high level is output when all TWI devices tri-state their outputs, allowing the pull-up resistors to pull the line high.



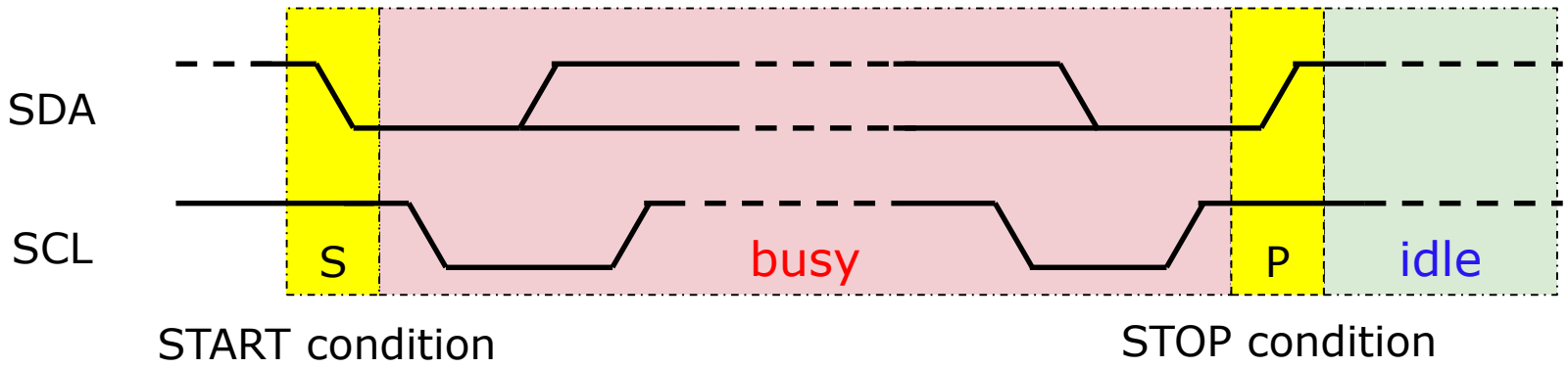
TWI START and STOP Conditions (1)

- START condition: A HIGH to LOW transition of the SDA line while SCL is HIGH.
- STOP condition: A LOW to HIGH transition of the SDA line while SCL is HIGH.
- START and STOP conditions are always generated by the master.
- Between a START and a STOP condition, the bus is considered busy, and no other master should try to seize control of the bus.



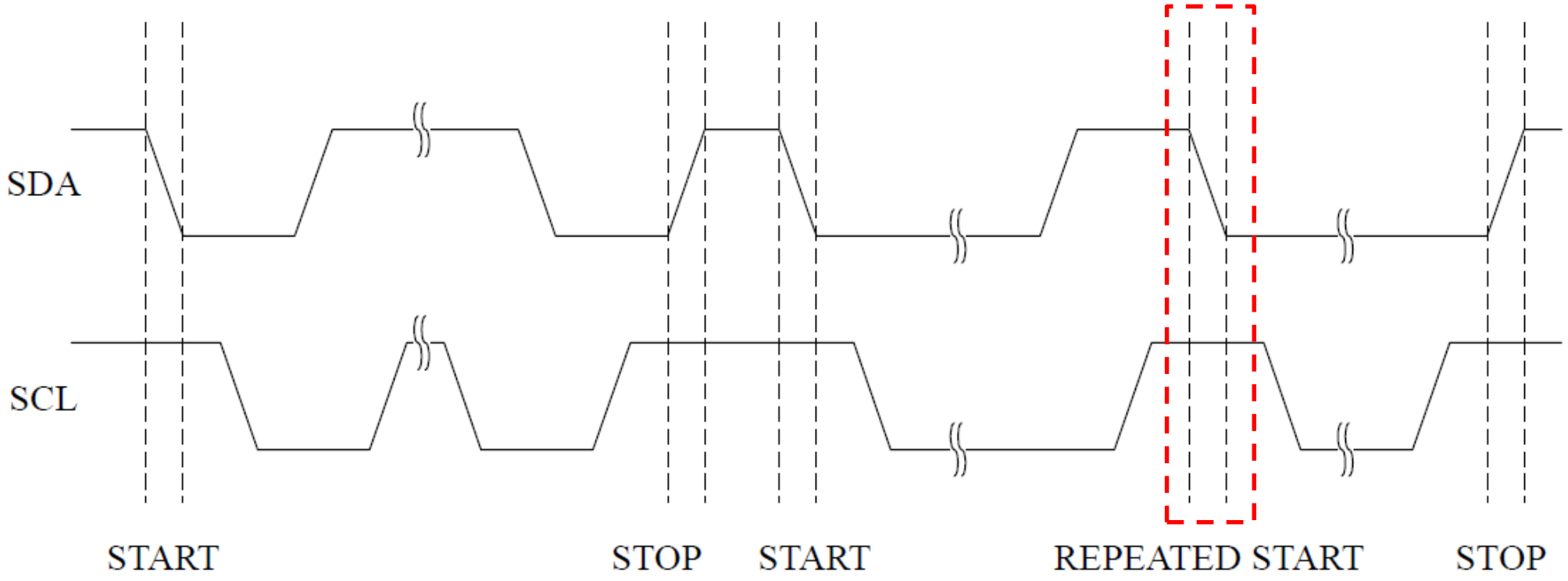
TWI START and STOP Conditions (2)

- Bus Busy:
  - After a START condition the bus is considered to be busy.
  - No other master should try to seize control of the bus.
- Bus Idle:
  - The bus becomes idle again after a STOP condition



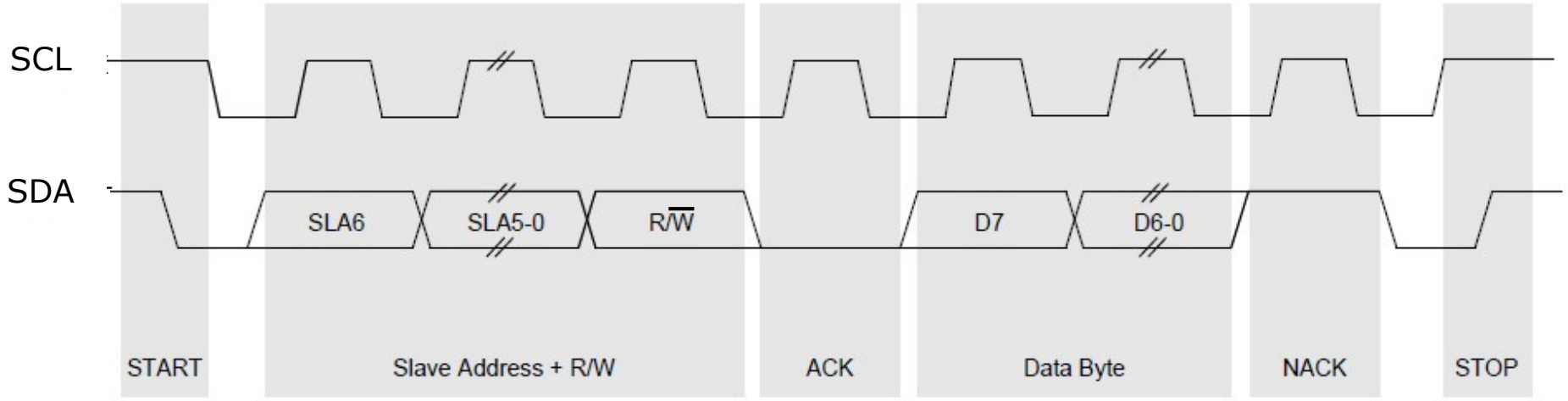
TWI REPEATED START Conditions

- REPEATED START condition: A new START condition is issued between a START and STOP condition and is used when the Master wishes to initiate a new transfer without relinquishing control of the bus.



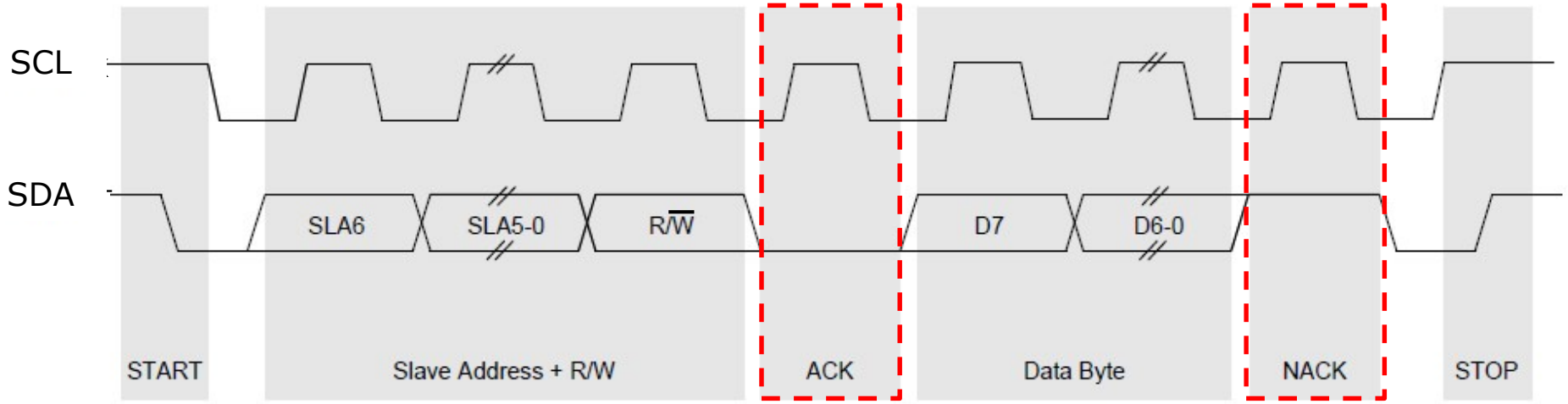
Typical TWI Transaction (1)

- A typical TWI transaction consists of
  - START condition
  - Slave address byte (Bits7-1: 7-bit slave address; Bit0: R/$\overline{W}$ direction bit)
  - One or more bytes of data
  - ACK/NAK bit
  - STOP condition



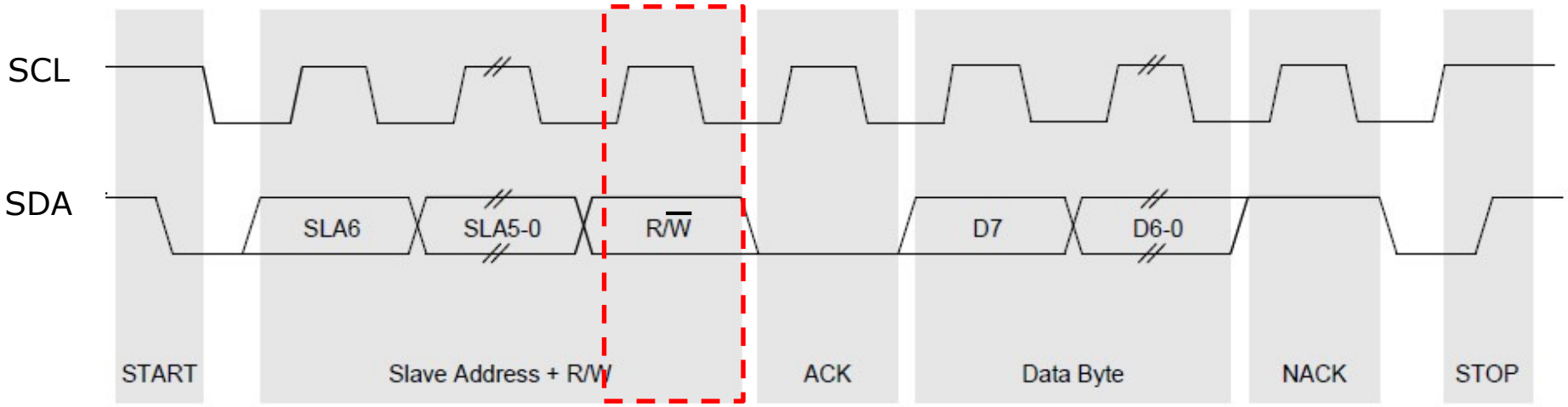
Typical TWI Transaction (2)

- ACK
  - Each byte that is received (by a master or slave) must be acknowledged (ACK) with a low SDA during a high SCL.
- NACK
  - If the receiving device does not ACK, the transmitting device will read a "not acknowledge" (NACK), which is a high SDA during a high SCL.



Typical TWI Transaction (3)

- The direction bit (R/$\overline{W}$) occupies the least-significant bit position of the address.
  - READ: The direction bit is set to logic 1
  - WRITE: The direction bit is set to logic 0



TWI: Arbitration

- Bus free:
  - After a STOP condition, or
  - After the SCL and SDA lines remain high for a specified time.
- A master may start a transfer only if the bus is free.
- What will happen if two or more devices attempt to begin a transfer at the same time?
  - An arbitration scheme is employed to force one master to give up the bus.

TWI: Arbitration Scheme

- The master devices continue transmitting until one attempts a HIGH while the other transmits a LOW.
- Since the bus is open-drain, the bus will be pulled LOW.
- The master attempting the HIGH will detect a LOW SDA and give up the bus.
- The winning master continues its transmission without interruption; the losing master becomes a slave and receives the rest of the transfer.
- This arbitration scheme is non-destructive: one device always wins, and no data is lost.

TWI: Transfer Modes

- The TWI interface may be configured to operate as a master and/or a slave.
- At any particular time, the interface will be operating in one of the following modes:
  - Master Transmitter
  - Master Receiver
  - Slave Transmitter
  - Slave Receiver

TWI: Master Transmitter Mode

S START. Transmitted by ATmega328 TWI.

Slave Addr. Slave address. Transmitted by ATmega328 TWI.

W Data direction (R/$\overline{W}$) bit. Transmitted by ATmega328 TWI. Logic `0`.

Data Byte Data byte. Transmitted by ATmega328 TWI.

A ACK. Received by ATmega328 TWI.

P STOP. Transmitted by ATmega328 TWI.

TWI: Master Receiver Mode

S	START. Transmitted by ATmega328 TWI.
Slave Addr.	Slave address. Transmitted by ATmega328 TWI.
R	Data direction (R/$\overline{W}$) bit. Transmitted by ATmega328 TWI. Logic 1
Data Byte	Data byte. Received by ATmega328 TWI.
A	ACK. Received by ATmega328 TWI.
A N	ACK or NACK. Transmitted by ATmega328 TWI depending on the state of the TWEA bit in register TWCRn.
P	STOP. Transmitted by ATmega328 TWI.

TWI: Slave Transmitter Mode

S START. Received by ATmega328 TWI.

Slave Addr. Slave address (TWARn register). Received by ATmega328 TWI.

R Data direction (R/$\overline{W}$) bit. Received by ATmega328 TWI. Logic 1

Data Byte Data byte. Transmitted by ATmega328 TWI.

A ACK. Transmitted by ATmega328 TWI.

A ACK. Received by ATmega328 TWI.

N NACK. Received by ATmega328 TWI.

P STOP. Received by ATmega328 TWI.

TWI: Slave Receiver Mode

S START. Received by ATmega328 TWI.

Slave Addr. Slave address (TWARn register). Received by ATmega328 TWI.

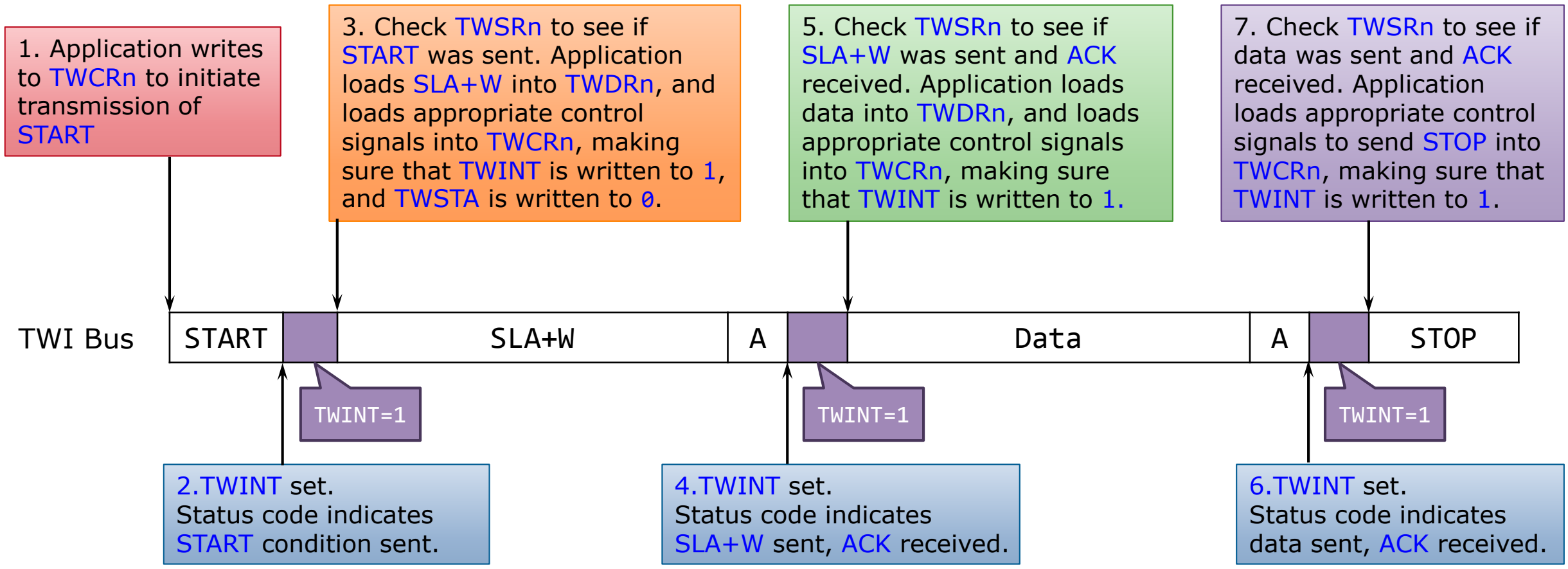
W Data direction (R/$\overline{W}$) bit. Received by ATmega328 TWI. Logic `0`

Data Byte Data byte. Received by ATmega328 TWI.

A
N ACK or NACK. Transmitted by ATmega328 TWI depending on the state of the TWEA bit in register TWCRn.

P STOP. Received by ATmega328 TWI.

Interfacing Application to the TWI in Master Transmitter (1)



Interfacing Application to the TWI in Master Transmitter (2)

1. The first step in a TWI transmission is to transmit a START condition.
   - This is done by writing a specific value into TWCRn , instructing the TWIn hardware to transmit a START condition.
   - The TWIn will not start any operation as long as the TWINT bit in TWCRn is set. Writing a 1 to TWINT clears the flag.
   - Immediately after the application has cleared TWINT, the TWIn will initiate transmission of the START condition.
2. When the START condition has been transmitted, the TWINT flag in TWCRn is set, and TWSRn is updated with a status code indicating that the START condition has successfully been sent.

Interfacing Application to the TWI in Master Transmitter (3)

3. The application software should now examine the value of TWSRn, to make sure that the START condition was successfully transmitted.
   - If TWSRn indicates otherwise, the application software might take some special action, like calling an error routine.
   - Assuming that the status code is as expected, the application must load SLA+W into TWDRn.
   - Remember that TWDRn is used both for address and data.
   - After TWDRn has been loaded with the desired SLA+W, a specific value must be written to TWCRn, instructing the TWIn hardware to transmit the SLA+W present in TWDRn.
   - The TWIn will not start any operation as long as the TWINT bit in TWCRn is set. Writing a 1 to TWINT clears the flag.
   - Immediately after the application has cleared TWINT, the TWIn will initiate transmission of the address packet.

Interfacing Application to the TWI in Master Transmitter (4)

4. When the address packet has been transmitted, the TWINT flag in TWCRn is set, and TWSRn is updated with a status code indicating that the address packet has successfully been sent.
   - The status code will also reflect whether a Slave acknowledged the packet or not.
5. The application software should now examine the value of TWSRn, to make sure that the address packet was successfully transmitted, and that the value of the ACK bit was as expected.
   - If TWSRn indicates otherwise, the application software might take some special action, like calling an error routine.
   - Assuming that the status code is as expected, the application must load a data packet into TWDRn.
   - Subsequently, a specific value must be written to TWCRn, instructing the TWIn hardware to transmit the data packet present in TWDRn.
   - The TWIn will not start any operation as long as the TWINT bit in TWCRn is set. Writing a 1 to TWINT clears the flag.
   - Immediately after the application has cleared TWINT, the TWIn will initiate transmission of the data packet.

Interfacing Application to the TWI in Master Transmitter (5)

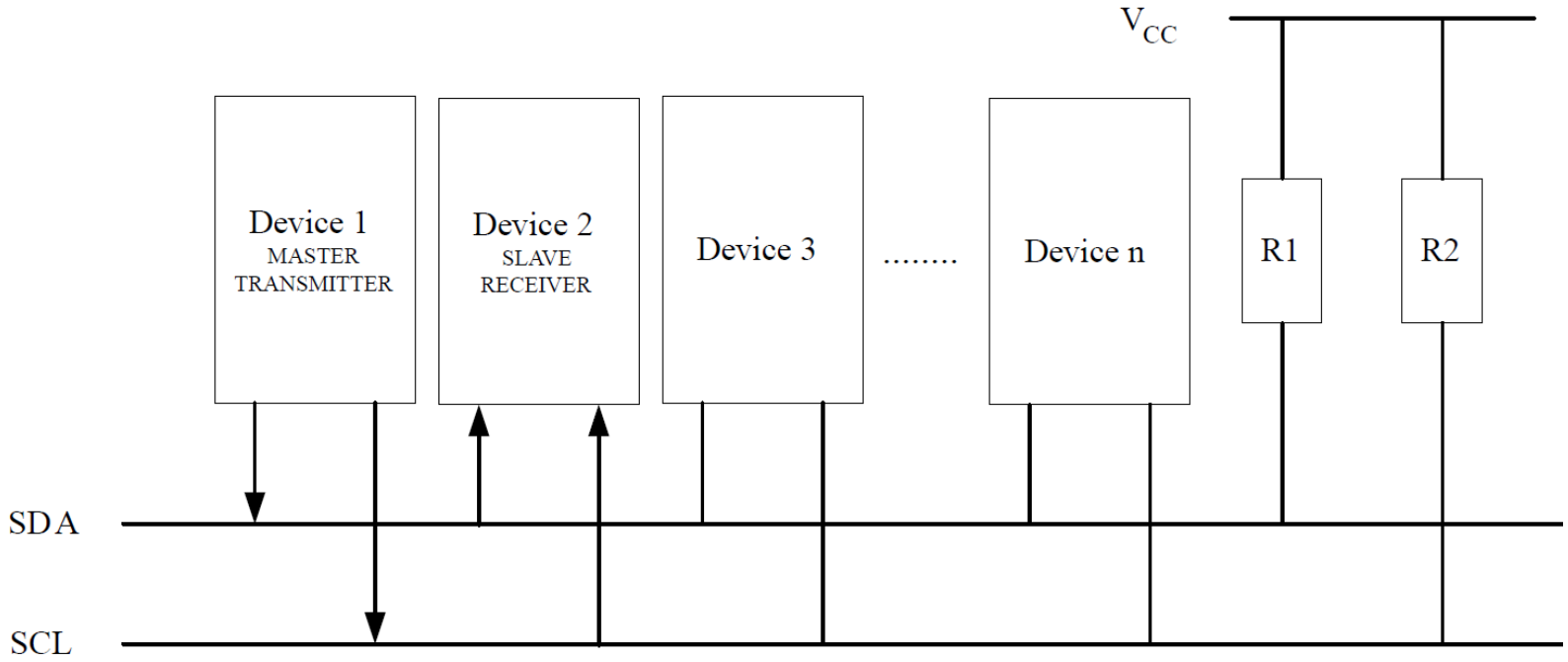
6. When the data packet has been transmitted, the TWINT flag in TWCRn is set, and TWSRn is updated with a status code indicating that the data packet has successfully been sent.
   - The status code will also reflect whether a Slave acknowledged the packet or not.
7. The application software should now examine the value of TWSRn, to make sure that the data packet was successfully transmitted, and that the value of the ACK bit was as expected.
   - If TWSR indicates otherwise, the application software might take some special action, like calling an error routine.
   - Assuming that the status code is as expected, the application must write a specific value to TWCRn, instructing the TWIn hardware to transmit a STOP condition.
   - The TWIn will not start any operation as long as the TWINT bit in TWCRn is set. Writing a 1 to TWINT clears the flag.
   - Immediately after the application has cleared TWINT, the TWI will initiate transmission of the STOP condition.
   - Note that TWINT is not set after a STOP condition has been sent.

Interfacing Application to the TWI in Master Transmitter (6)

	C Example Code	Comments
1	`TWCR0 = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN)`	Send START condition
2	`while (!(TWCR0 & (1<<TWINT)));`	Wait for TWINT Flag set. This indicates that the START condition has been transmitted.
3	`if ((TWSR0 & 0xF8) != START)` `  ERROR();`	Check value of TWI Status Register. Mask prescaler bits. If status different from START, go to ERROR.
	`TWDR0 = SLA_W;` `TWCR0 = (1 << TWINT) | (1 << TWEN);`	Load SLA_W into TWDR Register. Clear TWINT bit in TWCR to start transmission of address.
4	`while (!(TWCR0 & (1 << TWINT)));`	Wait for TWINT Flag set. This indicates that the SLA+W has been transmitted, and ACK/NACK has been received.
5	`if ((TWSR0 & 0xF8) != MT_SLA_ACK)` `  ERROR();`	Check value of TWI Status Register. Mask prescaler bits. If status different from MT_SLA_ACK go to ERROR.
	`TWDR0 = DATA;` `TWCR0 = (1 << TWINT) | (1 << TWEN);`	Load DATA into TWDR Register. Clear TWINT bit in TWCR to start transmission of data.
6	`while (!(TWCR0 & (1 << TWINT)));`	Wait for TWINT Flag set. This indicates that the DATA has been transmitted, and ACK/NACK has been received.
7	`if ((TWSR0 & 0xF8) != MT_DATA_ACK)` `  ERROR();`	Check value of TWI Status Register. Mask prescaler bits. If status different from MT_DATA_ACK go to ERROR.
	`TWCR0 = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);`	Transmit STOP condition.

TWI Master Transmitter Mode (1)

- In order to enter a Master mode, a START condition must be transmitted.
- The format of the following address packet determines whether Master Transmitter (MT) or Master Receiver (MR) mode is to be entered:
  - If SLA +W is transmitted, MT mode is entered.
  - if SLA+R is transmitted, MR mode is entered.
- All the status codes mentioned here assume that the prescaler bits are zero or are masked to zero.



TWI Master Transmitter Mode (2)

- A START condition is sent by writing a value to the TWI Control Register n (TWCRn) of the type TWCRn=1x10x10x:
  - The TWI Enable bit (TWCRn.TWEN) must be written to '1' to enable the 2-wire Serial Interface
  - The TWI Start Condition bit (TWCRn.TWSTA) must be written to '1' to transmit a START condition
  - The TWI Interrupt Flag (TWCRn.TWINT) must be written to '1' to clear the flag.
- The TWIn will then test the 2-wire Serial Bus and generate a START condition as soon as the bus becomes free.
- After a START condition has been transmitted, the TWINT flag is set by hardware, and the status code in TWSRn will be 0x08.
- In order to enter MT mode, SLA+W must be transmitted. This is done by writing SLA+W to the TWI Data Register (TWDRn).
- Thereafter, the TWCRn.TWINT flag should be cleared by writing a '1' to it to continue the transfer. This is accomplished by writing a value to TWRCn of the type TWCRn=1x00x10x.

TWI Master Transmitter Mode (3)

- When SLA+W have been transmitted and an acknowledgment bit has been received, TWINT is set again and a number of status codes in TWSRn are possible. Possible status codes in Master mode are 0x18, 0x20, or 0x38.
- The appropriate action to be taken for each of these status codes is detailed in the Status Code table below.
- When SLA+W has been successfully transmitted, a data packet should be transmitted.
  - This is done by writing the data byte to TWDRn.
  - TWDRn must only be written when TWINT is high.
  - If not, the access will be discarded, and the Write Collision bit (TWWC) will be set in the TWCRn Register.
- After updating TWDRn, the TWINT bit should be cleared by writing '1' to it to continue the transfer.
  - This is accomplished by writing again a value to TWCRn of the type TWCRn=1x00x10x.
- This scheme is repeated until the last byte has been sent and the transfer is ended, either by generating a STOP condition or a by a repeated START condition.
  - A STOP condition is generated by writing a value of the type TWCRn=1x01x10x.
  - A repeated START condition is accomplished by writing a regular START value TWCRn=1x10x10x.
- After a repeated START condition (status code 0x10), the 2-wire Serial Interface can access the same Slave again, or a new Slave without transmitting a STOP condition.
- Repeated START enables the Master to switch between Slaves, Master Transmitter mode and Master Receiver mode without losing control of the bus.

TWI Status Codes for Master Transmitter Mode (1)

Status Code (TWSRn) Prescaler bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCRn				
			STA	STO	TWINT	TWEA	
0x08	A START condition has been transmitted	Load SLA+W	0	0	1	X	SLA+W will be transmitted; ACK or NOT ACK will be received.
0x10	A repeated START condition has been transmitted	Load SLA+W	0	0	1	X	SLA+W will be transmitted; ACK or NOT ACK will be received
		Load SLA+R	0	0	1	X	SLA+R will be transmitted; Logic will switch to Master Receiver mode
0x18	SLA+W has been transmitted; ACK has been received	Load data byte	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDRn Action	1	0	1	X	Repeated START will be transmitted
		No TWDRn action	0	0	1	X	STOP condition will be transmitted and TWSTO Flag will be reset
		No TWDRn action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset

TWI Status Codes for Master Transmitter Mode (2)

Status Code (TWSRn) Prescaler bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCRn				
			STA	STO	TWINT	TWEA	
0x20	SLA+W has been transmitted; NOT ACK has been received	Load data byte	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received.
		No TWDRn action	1	0	1	X	Repeated START will be transmitted.
		No TWDRn action	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset
		No TWDRn action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
0x28	Data byte has been transmitted; ACK has been received	Load data byte	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDRn action	1	0	1	X	Repeated START will be transmitted
		No TWDRn action	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset
		No TWDRn action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset

TWI Status Codes for Master Transmitter Mode (3)

Status Code (TWSRn) Prescaler bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCRn				
			STA	STO	TWINT	TWEA	
0x30	Data byte has been transmitted; NOT ACK has been received	Load data byte	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received.
		No TWDRn action	1	0	1	X	Repeated START will be transmitted.
		No TWDRn action	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset
		No TWDRn action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
0x38	Arbitration lost in SLA+W or data bytes	No TWDRn action	0	0	1	X	2-wire Serial Bus will be released and not addressed Slave mode entered.
		No TWDRn action	1	0	1	X	A START condition will be transmitted when the bus becomes free

TWI Status Codes for Master Receiver Mode (1)

Status Code (TWSRn) Prescaler bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCRn				
			STA	STO	TWINT	TWEA	
0x08	A START condition has been transmitted	Load SLA+R	0	0	1	X	SLA+R will be transmitted; ACK or NOT ACK will be received.
0x10	A repeated START condition has been transmitted	Load SLA+R	0	0	1	X	SLA+R will be transmitted; ACK or NOT ACK will be received
		Load SLA+W	0	0	1	X	SLA+W will be transmitted; Logic will switch to Master Transmitter mode
0x38	Arbitration lost in SLA+R or NOT ACK bit	No TWDRn action	0	0	1	X	2-wire Serial Bus will be released and not addressed Slave mode will be entered
		No TWDRn action	1	0	1	X	A START condition will be transmitted when the bus becomes free.

TWI Status Codes for Master Receiver Mode (2)

Status Code (TWSRn) Prescaler bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCRn				
			STA	STO	TWINT	TWEA	
0x40	SLA+R has been transmitted; ACK has been received	No TWDRn action	0	0	1	0	Data byte will be received and NOT ACK will be returned.
		No TWDRn action	0	0	1	1	Data byte will be received and ACK will be returned.
0x48	SLA+R has been transmitted; NOT ACK has been received	No TWDRn action	1	0	1	X	Repeated START will be transmitted.
		No TWDRn action	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset.
		No TWDRn action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset.

TWI Status Codes for Master Receiver Mode (3)

Status Code (TWSRn) Prescaler bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCRn				
			STA	STO	TWINT	TWEA	
0x50	Data byte has been received; ACK has been returned.	Read data byte	0	0	1	0	Data byte will be received and NOT ACK will be returned.
		Read data byte	0	0	1	1	Data byte will be received and ACK will be returned.
0x58	Data byte has been received; NOT ACK has been returned.	Read data byte	1	0	1	X	Repeated START will be transmitted.
		Read data byte	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset.
		Read data byte	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset.

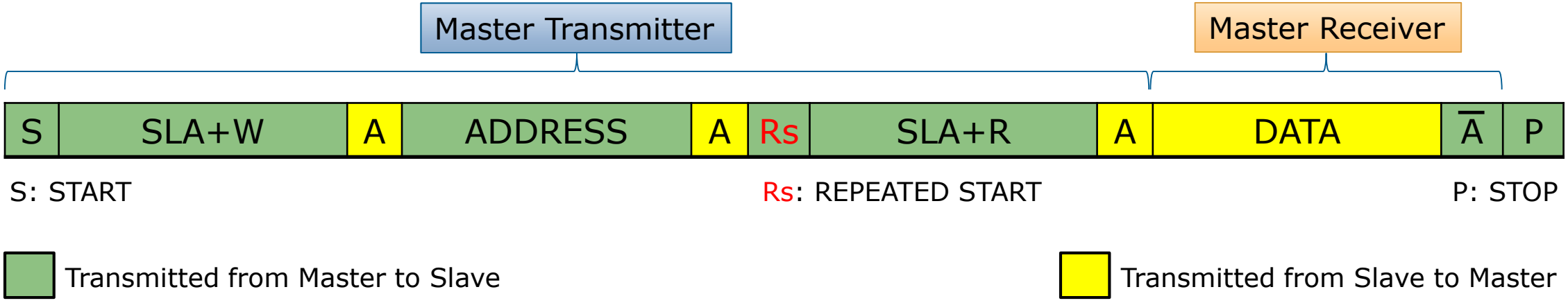
TWI Combining Several TWI Modes (1)

- In some cases, several TWI modes must be combined in order to complete the desired action.

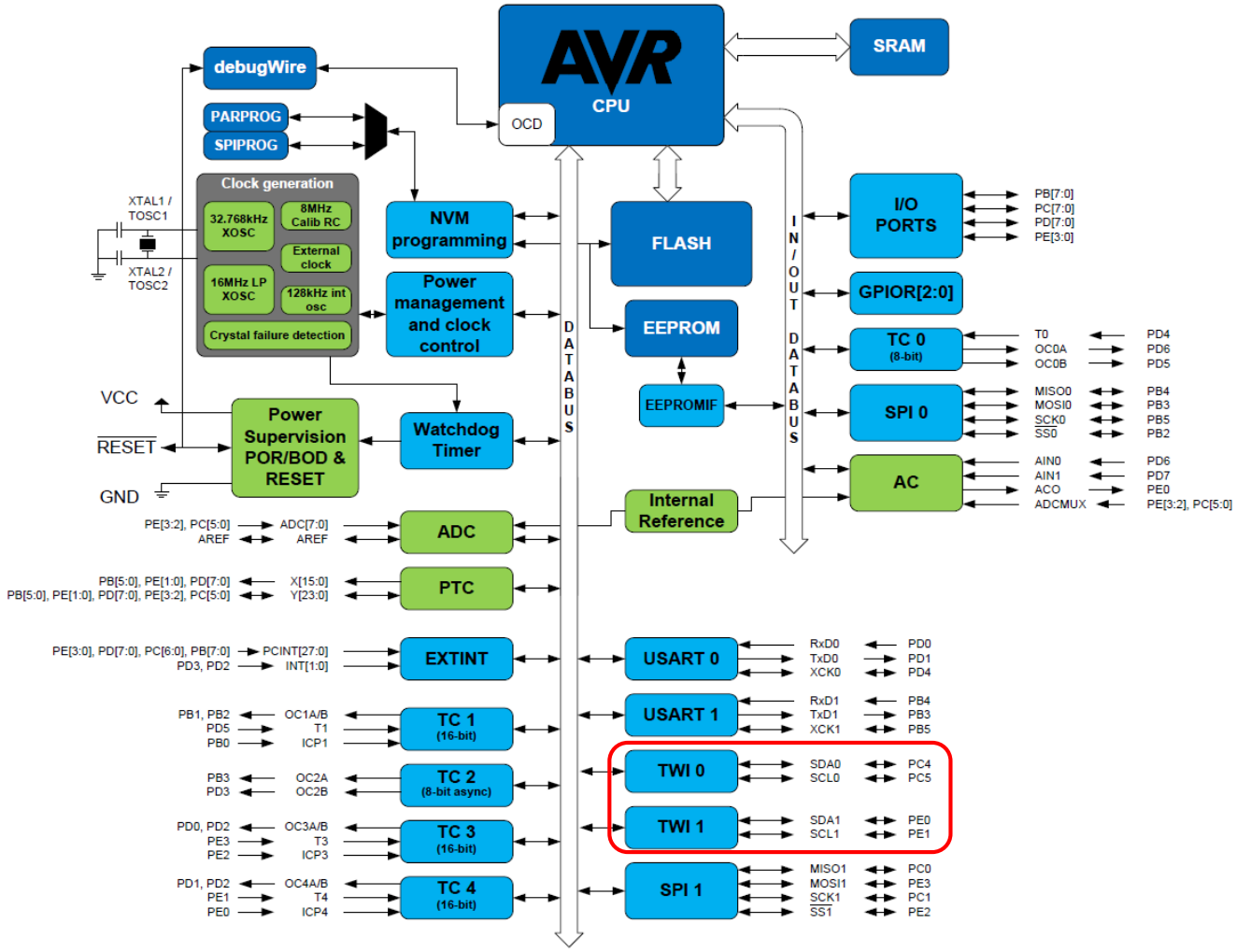
  Example: Reading data from a serial EEPROM(AT24C02, random reading) involving the following steps:

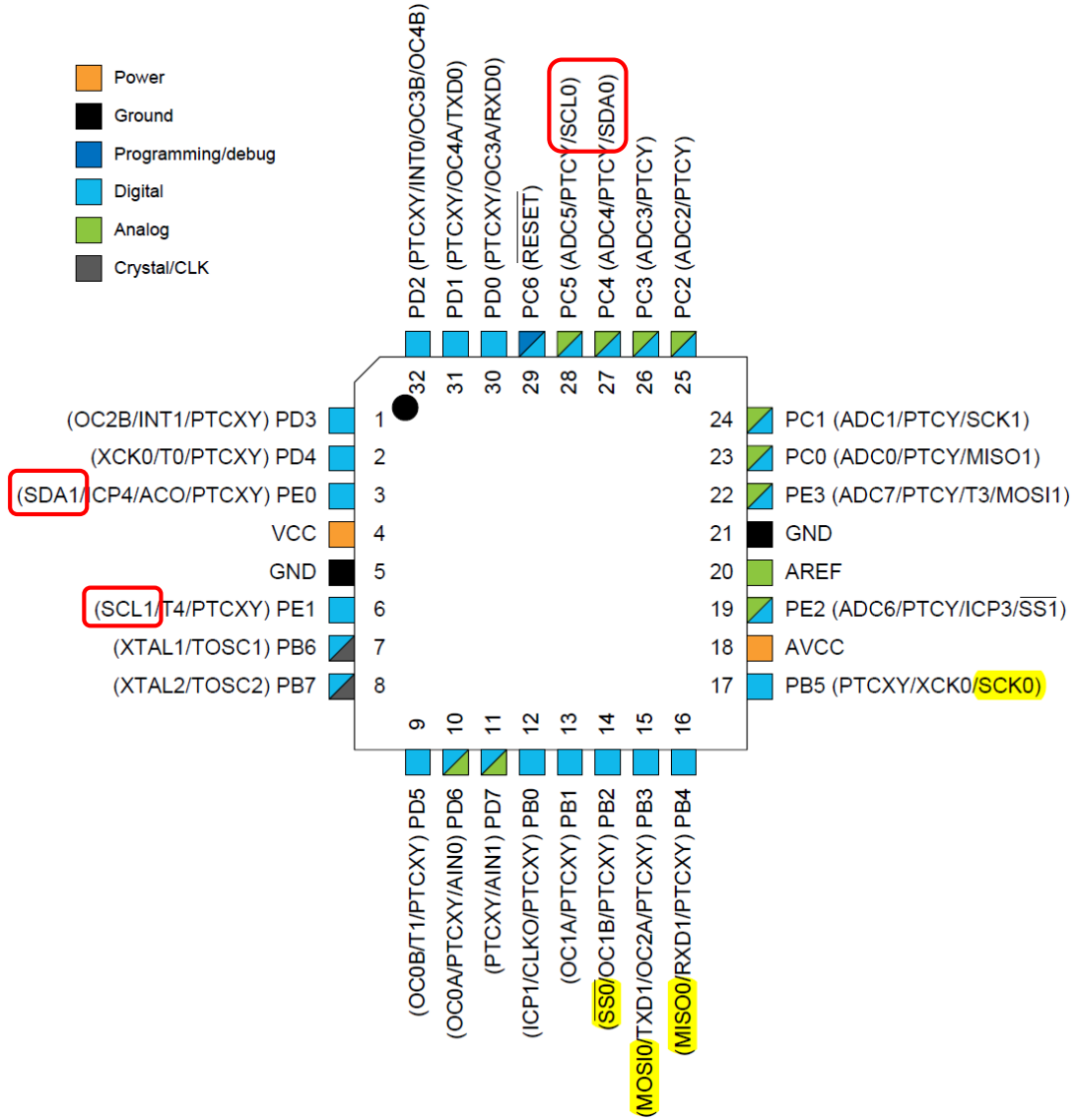
  1. The transfer must be initiated. (MT mode)
  2. The EEPROM must be instructed what location should be read. (MT mode)
  3. The reading must be performed. (MR mode)
  4. The transfer must be finished. (MT mode)

- The Master must keep control of the bus during all these steps in atomic operation.
- A change in transfer direction is accomplished by transmitting a REPEATED START between the transmission of the address byte and reception of the data.



ATmega328PB TWIn Pins





TWI Control Register (TWCRn)

TWI0 Enable Acknowledge
If the TWEA bit is written to '1', the ACK pulse is generated on the TWI0 bus if the conditions are met.

TWI STOP Condition
Writing '1' to the TWSTO bit in Master mode will generate a STOP condition on the 2-wire Serial Bus TWI0. When the STOP condition is executed on the bus, the TWSTO bit is cleared automatically.

TWI Enable
'1': TWI0 takes control over the I/O pins connected to the SCL and SDA pins.

TWI Interrupt Enable
'1': Enable TWI0 Interrupt.

TWCR0	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN		TWIE
	1	*	*	*	0	1	0	0

TWI0 Interrupt Flag
This bit is set by hardware when the TWI0 has finished its current job and expects application software response. It must be cleared by software by writing a logic '1' to it.

TWI START Condition
The application writes '1' to the TWSTA bit when it desires TWI0 to become a Master on the 2-wire Serial Bus. The TWI0 hardware checks if the bus is available, and generates a START condition on the bus if it is free. This bit must be cleared by software when the START condition has been transmitted

TWI Write Collision Flag
This bit is set when attempting to write to the TWDR0 when TWINT is low. This flag is cleared by writing the TWDR0 register when TWINT is high.

TWI Status Register (TWSRn)

TWI Status Bits
The TWS[7:3] reflect the status of the TWI0 logic and the 2-wire Serial Bus.

TWI Bit Rate Prescaler
00: Divide by 1
01: Divide by 4
10: Divide by 16
11: Divide by 64

TWSR0	TWS7	TWS6	TWS5	TWS4	TWS3		TWPS1	TWPS0
	*	*	*	*	*	0	0	0

System Clock: 16 MHz, SCL: 400 kHz
SCL freq = F_CPU/(16 + 2 * TWBR * Prescaler)
SCL freq = 16,000,000Hz/(16+2*12*1)=16,000,000Hz/40=400kHz

TWI Bit Rate Register (TWBRn)

TWI Bit Rate Register
TWBR0 selects the division factor for the bit rate generator. The bit rate generator is a frequency divider which generates the SCL clock frequency in the Master modes.

TWBR0

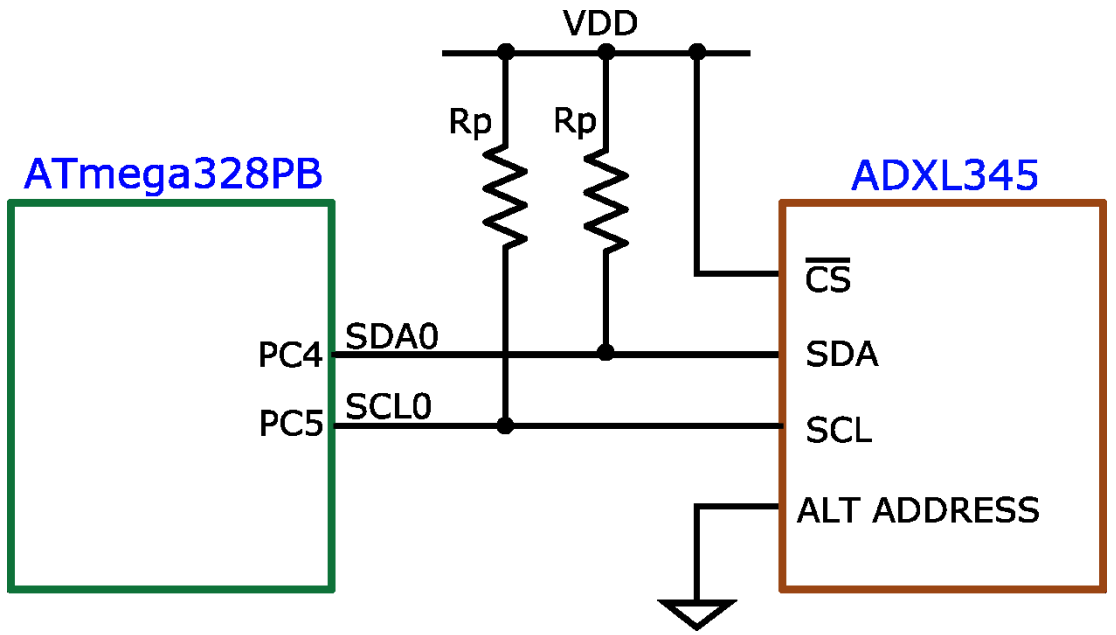
TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0
0	0	0	0	1	1	0	0

$00001100_2 = 12_{10}$

System Clock: 16 MHz, SCL: 400 kHz
SCL freq = F_CPU/(16 + 2 * TWBR * Prescaler)
SCL freq = 16,000,000Hz/(16+2*12*1)=16,000,000Hz/40=400kHz

TWI0 Example I(1)

- Specifications:
  - CPU clock: 16 MHz
  - SCL: 400 kHz
  - ADXL345 Address
    - ✓ Write (SLA_W): 0xA6
    - ✓ Read (SLA_R): 0xA7
- Make an application which reads ID and X-, Y-, Z-axis acceleration values.
  - Register addresses for
    - ✓ ID: 0x00
    - ✓ X-axis: 0x32 and 0x33
    - ✓ Y-axis: 0x34 and 0x35
    - ✓ Z-axis: 0x36 and 0x37
- Use polling method



TWI0 Example I(2)

ADXL345 I2C Device Addressing (1)

Single-Byte Write

Master	START		Slave Addr + Write		Register Address		Data		STOP
Slave				ACK		ACK		ACK	

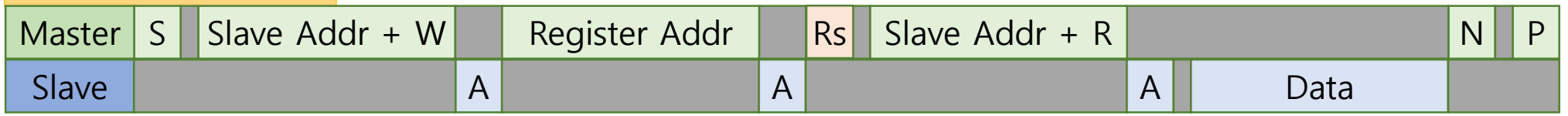
Multiple-Byte Write

Master	START		Slave Addr + Write		Register Address		Data		Data		STOP
Slave				ACK		ACK		ACK		ACK	

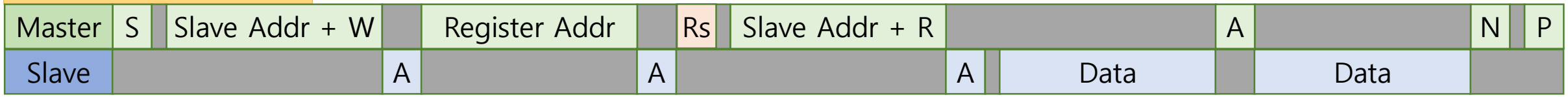
TWI0 Example I(3)

ADXL345 I2C Device Addressing (2)

Single-Byte Read



Multiple-Byte Read

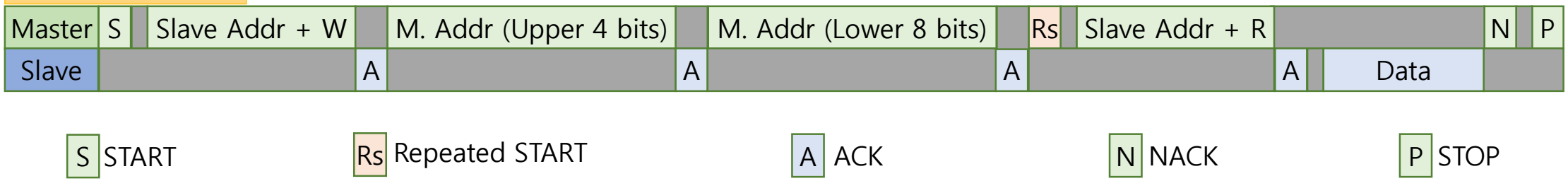


S START Rs Repeated START A A ACK N NACK P STOP

TWI0 Example I(3)

ADXL345 I2C Device Addressing (2)

Single-Byte Read



S START

Rs Repeated START

A ACK

N NACK

P STOP

TWI0 Example I(4) – Read ADXL345 ID

```
// Set SCL frequency to 400 kHz
TWSR0 = 0;    // Prescaler = 1
TWBR0 = 12;

// Send START condition
TWCR0 = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);

// Wait for the transmission of START condition
while (!(TWCR0 & (1 << TWINT)));
if ((TWSR0 & 0xF8) != 0x08)
    return -1;    // Error

TWDR0 = 0xA6;    // Load SLA_W (SLAVE ADDRESS + WRITE)

// Clear TWINT to start transmission of SLA_W
TWCR0 = (1 << TWINT) | (1 << TWEN);

// Wait for the transmission of SLA_W
while (!(TWCR0 & (1 << TWINT)));
if ((TWSR0 & 0xF8) != 0x18)
    return -1;    // Error

TWDR0 = 0x00;    // Load register address to be read
TWCR0 = (1<<TWINT) | (1<<TWEN);

// Wait for the transmission of register address
while (!(TWCR0 & (1 << TWINT)));
if ((TWSR0 & 0xF8) != 0x28)
    return -1;    // Error
```

```
// repeated START
TWCR0 = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);

// Wait for the transmission of repeated START
while (!(TWCR0 & (1 << TWINT)));
if ((TWSR0 & 0xF8) != 0x10)
    return -1;    // Error

TWDR0 = 0xA7;    // Load SLA_R (SLAVE ADDRESS + READ)

// Clear TWINT to start transmission of SLA_R
TWCR0 = (1 << TWINT) | (1 << TWEN);

// Wait for the transmission of SLA_R
while (!(TWCR0 & (1 << TWINT)));
if ((TWSR0 & 0xF8) != 0x40)
    return -1;    // Error

// Clear TWINT to start reception. NAK will be returned.
TWCR0 = (1 << TWINT) | (1 << TWEN);
while (!(TWCR0 & (1 << TWINT)));    // Wait for the reception
if ((TWSR0 & 0xF8) != 0x58)
    return -1;    // Error

// Send STOP condition
TWCR0 = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);

// Read received data (ID)
ADXL345_ID = TWDR0;
```

TWI0 Example I(5) – Read ADXL345 X-,Y-,Z-Axis Data

```
// Send START condition
TWCR0 = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);

// Wait for the transmission of START condition
while (!(TWCR0 & (1 << TWINT)));
if ((TWSR0 & 0xF8) != 0x08)
    return -1;    // Error

TWDR0 = 0xA6;    // Load SLA_W (SLAVE ADDRESS + WRITE)

// Clear TWINT to start transmission of SLA_W.
TWCR0 = (1 << TWINT) | (1 << TWEN);

// Wait for the transmission of SLA_W
while (!(TWCR0 & (1 << TWINT)));
if ((TWSR0 & 0xF8) != 0x18)
    return -1;    // Error

TWDR0 = 0x32;    // Load start address to be read
TWCR0 = (1<<TWINT) | (1<<TWEN);

// Wait for the transmission of start address
while (!(TWCR0 & (1 << TWINT)));
if ((TWSR0 & 0xF8) != 0x28)
    return -1;    // Error

// repeated START
TWCR0 = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);

// Wait for the transmission of repeated START
while (!(TWCR0 & (1 << TWINT)));
if ((TWSR0 & 0xF8) != 0x10)
    return -1;    // Error
```

```
TWDR0 = 0xA7;    // Load SLA_R (SLAVE ADDRESS + READ')

// Clear TWINT to start transmission of SLA_R .
TWCR0 = (1 << TWINT) | (1 << TWEN);

// Wait for the transmission of SLA_R
while (!(TWCR0 & (1 << TWINT)));
if ((TWSR0 & 0xF8) != 0x40)
    return -1;    // Error

for (i=0; i<6; i++)
{
    if (i < 5)
    {
        // Clear TWINT to start reception. ACK will be returned
        TWCR0 = (1 << TWINT) | (1 << TWEN) | (1 << TWEA);
        return_code = 0x50;
    }
    else
    {
        // Clear TWINT to start reception. NAK will be returned
        TWCR0 = (1 << TWINT) | (1 << TWEN);
        return_code = 0x58;
    }
    while (!(TWCR0 & (1 << TWINT)));    // Wait for the reception of data
    if ((TWSR0 & 0xF8) != return_code)
        return -1;     // Error
    }
    buff[i] = TWDR0;    // Read received data and save it
}

// Send STOP condition
TWCR0 = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);
```

End