TWI

(I^2C)

Two-Wire Serial Interface

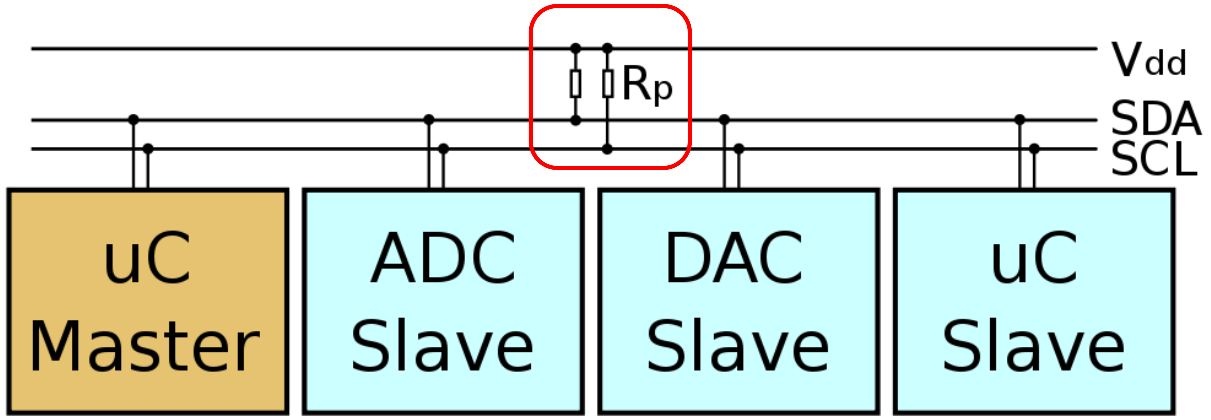
(Inter-Integrated Circuit)

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.

I^{2}C 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`)

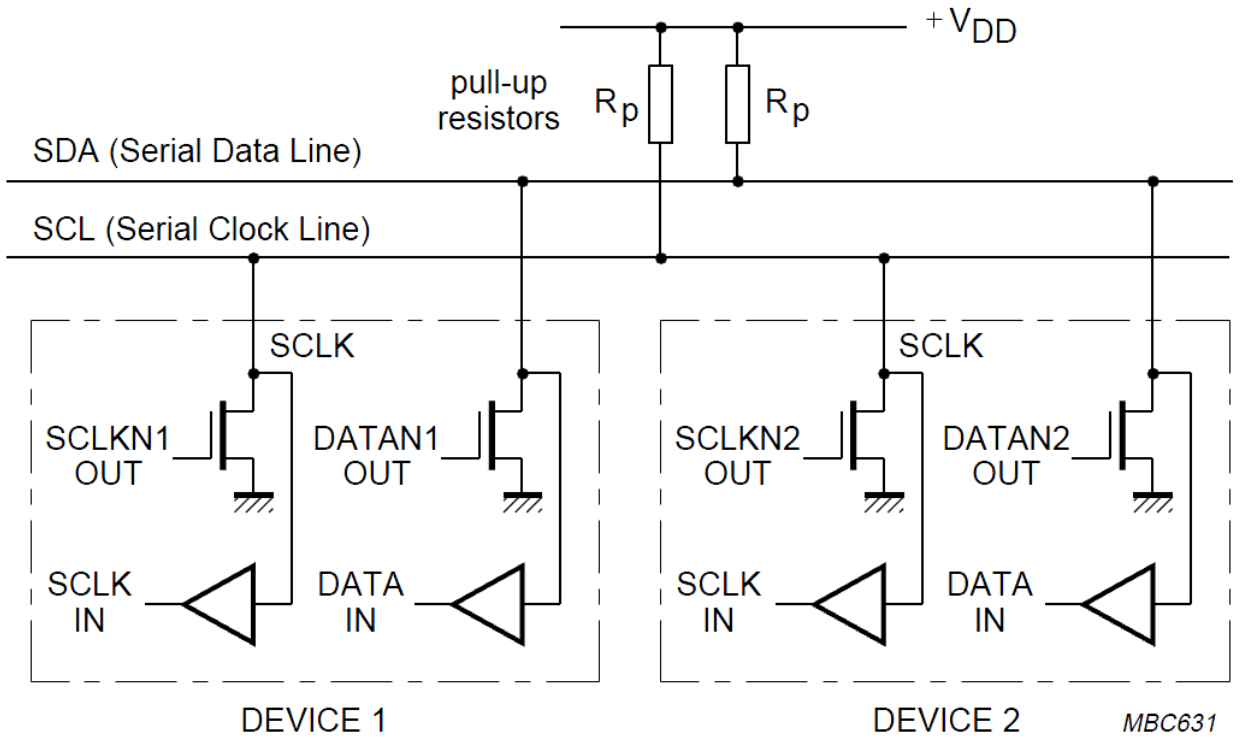
TWI Terminology

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.

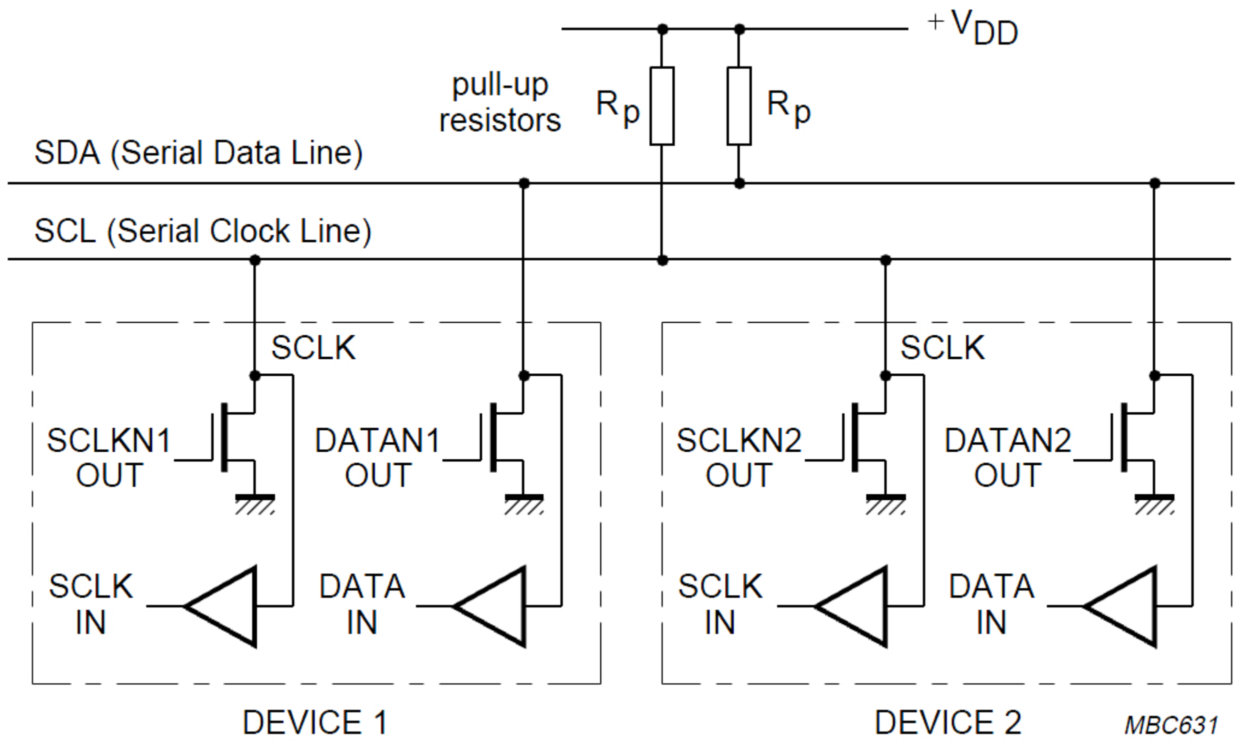
TWI Electrical Interconnection (1)

- 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.



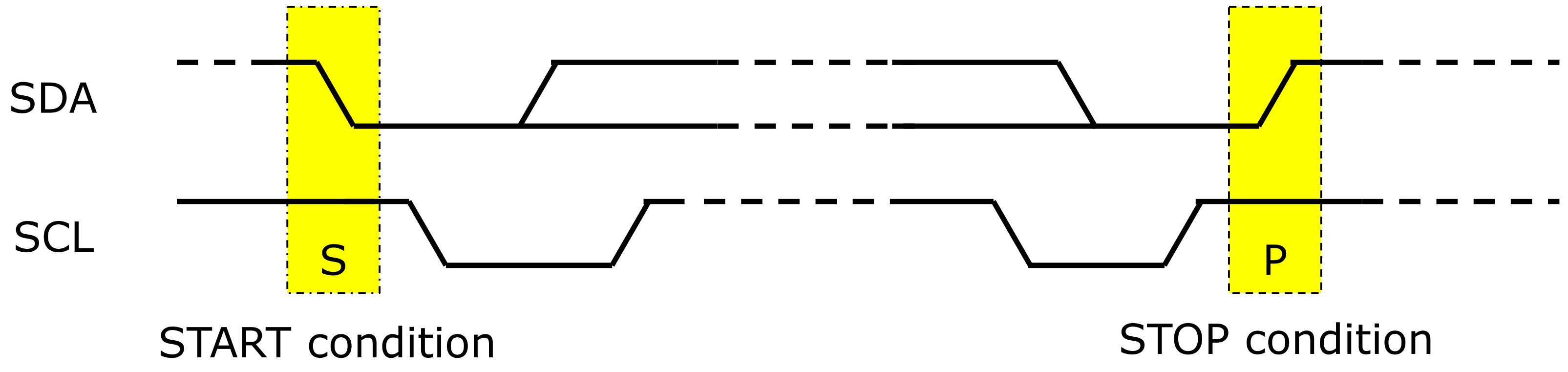
TWI Electrical Interconnection (2)

- 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.



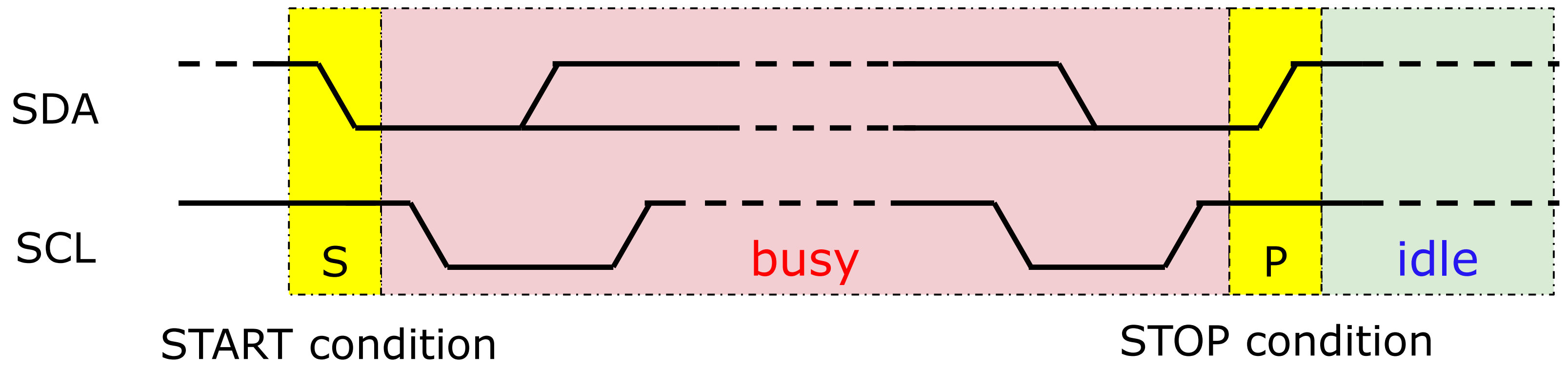
TWI START and STOP Conditions (1)

- 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.



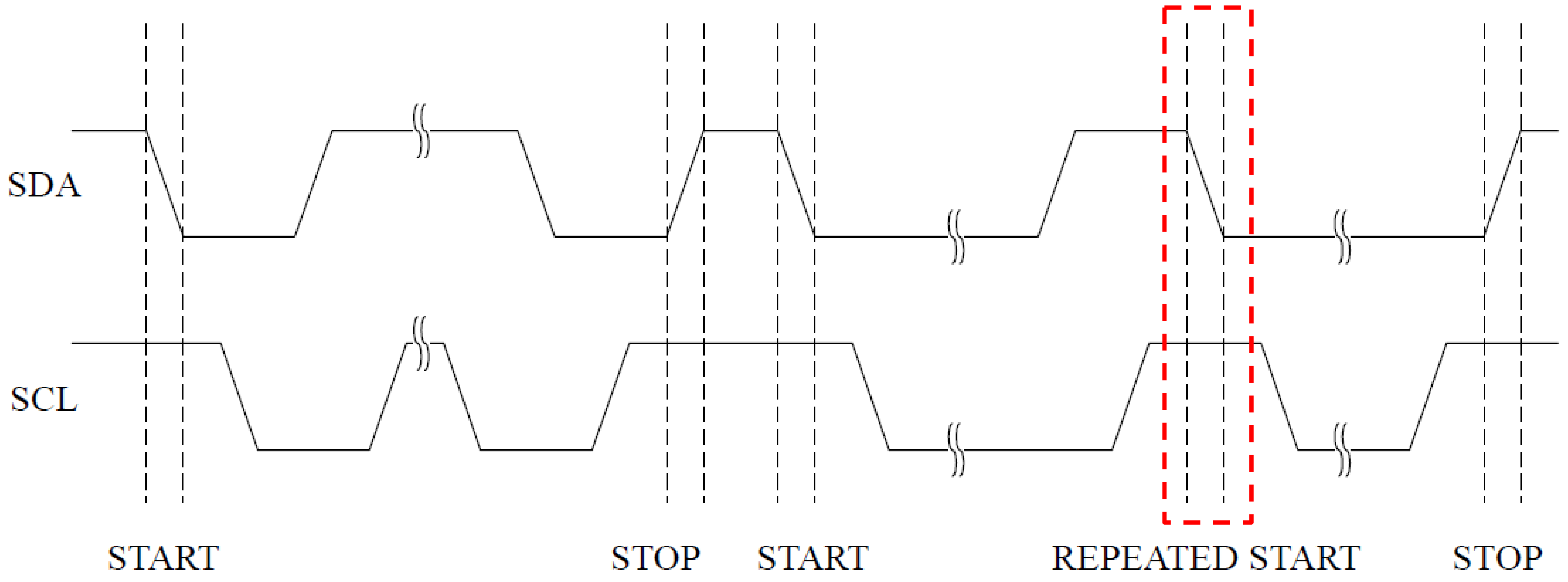
TWI START and STOP Conditions (2)

- 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



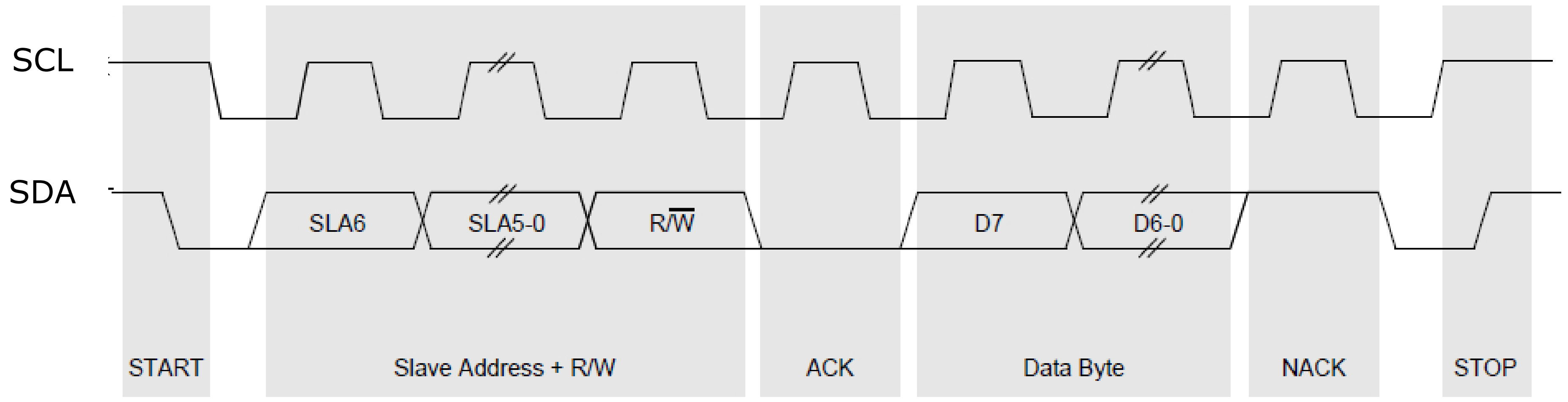
TWI REPEATED START Conditions

- 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.



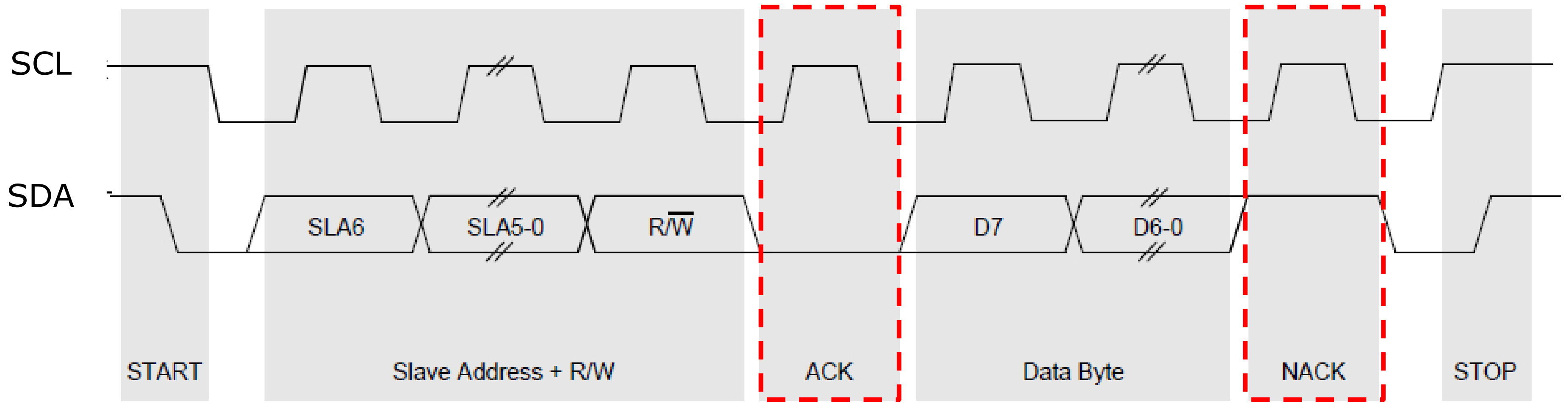
Typical TWI Transaction (1)

- 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



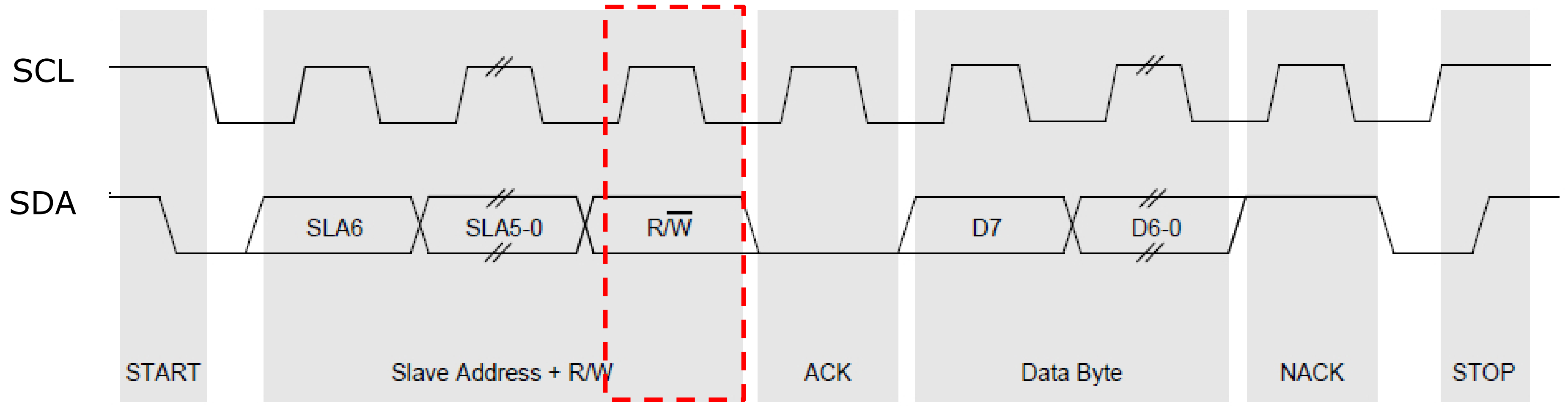
Typical TWI Transaction (2)

- 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)

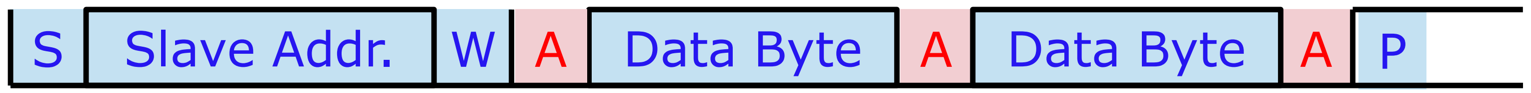
- The direction bit ($R/\overline{W}$) occupies the least-significant bit position of the address.
  - READ: The direction bit is set to logic 1
  - WRITE: The direction bit is set to logic 0



TWI: Transfer Modes

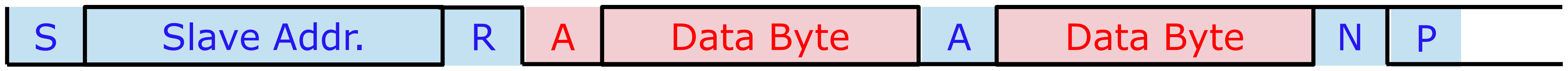
- The TWI interface may be configured to operate as a master and/or a slave.
- At any particular time, the interface will be operating in one of the following modes:
  - Master Transmitter
  - Master Receiver
  - Slave Transmitter
  - Slave Receiver

TWI: Master Transmitter Mode



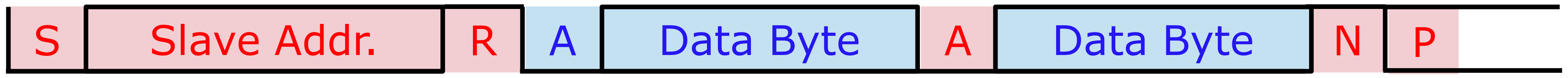
S	START. Transmitted by ATmega328 TWI.
Slave Addr.	Slave address. Transmitted by ATmega328 TWI.
W	Data direction ($R/\overline{W}$) bit. Transmitted by ATmega328 TWI. Logic `0`.
Data Byte	Data byte. Transmitted by ATmega328 TWI.
A	ACK. Received by ATmega328 TWI.
P	STOP. Transmitted by ATmega328 TWI.

TWI: Master Receiver Mode



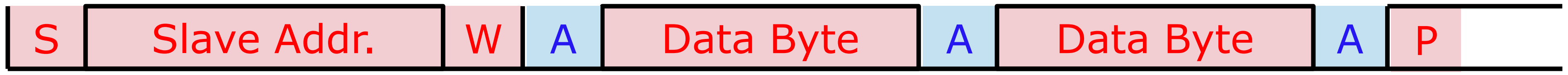
S	START. Transmitted by ATmega328 TWI.
Slave Addr.	Slave address. Transmitted by ATmega328 TWI.
R	Data direction ($R/\overline{W}$) bit. Transmitted by ATmega328 TWI. Logic 1
Data Byte	Data byte. Received by ATmega328 TWI.
A	ACK. Received by ATmega328 TWI.
A N	ACK or NACK. Transmitted by ATmega328 TWI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

TWI: Slave Transmitter Mode



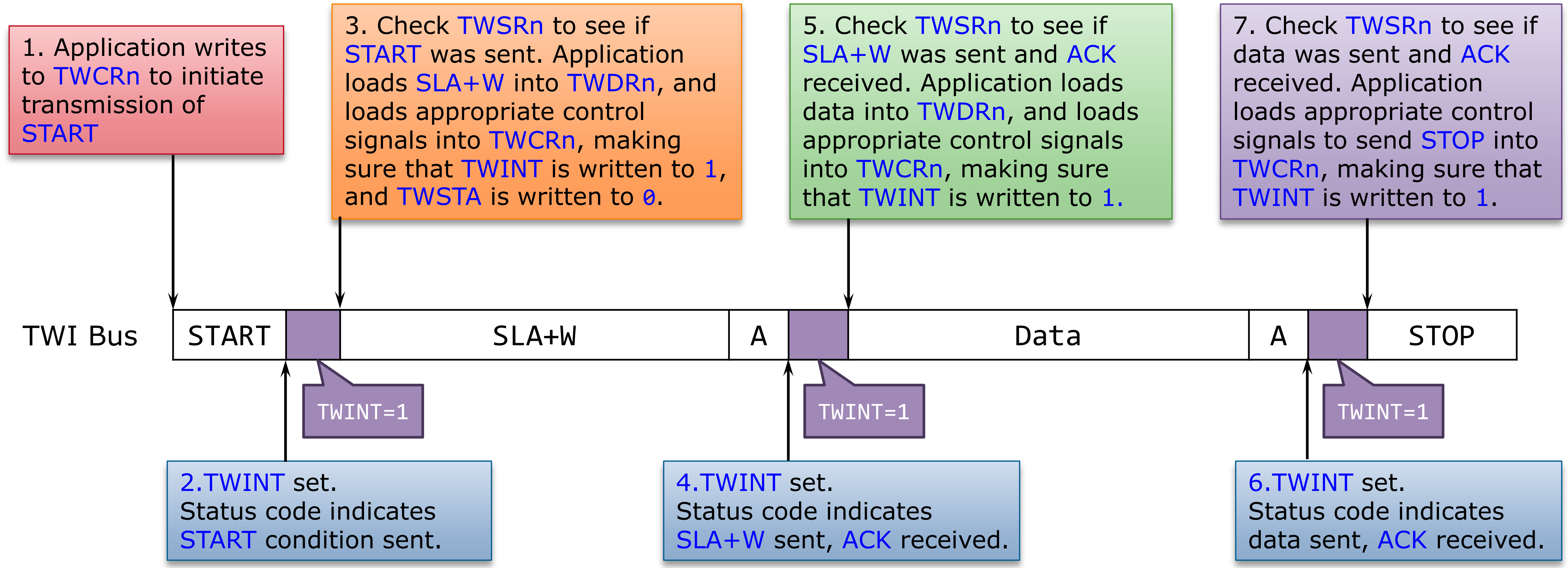
S	START. Received by ATmega328 TWI.
Slave Addr.	Slave address (TWARn register). Received by ATmega328 TWI.
R	Data direction ($R/\overline{W}$) bit. Received by ATmega328 TWI. Logic 1
Data Byte	Data byte. Transmitted by ATmega328 TWI.
A	ACK. Transmitted by ATmega328 TWI.
A	ACK. Received by ATmega328 TWI.
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)


1. Application writes to TWCRn to initiate transmission of START
3. Check TWSRn to see if START was sent. Application loads SLA+W into TWDRn, and loads appropriate control signals into TWCRn, making sure that TWINT is written to 1, and TWSTA is written to 0.
5. Check TWSRn to see if SLA+W was sent and ACK received. Application loads data into TWDRn, and loads appropriate control signals into TWCRn, making sure that TWINT is written to 1.
7. Check TWSRn to see if data was sent and ACK received. Application loads appropriate control signals to send STOP into TWCRn, making sure that TWINT is written to 1.
TWI Bus
START
SLA+W
A
Data
A
STOP
TWINT=1
TWINT=1
TWINT=1
2.TWINT set. Status code indicates START condition sent.
4.TWINT set. Status code indicates SLA+W sent, ACK received.
6.TWINT set. Status code indicates data sent, ACK received.

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.

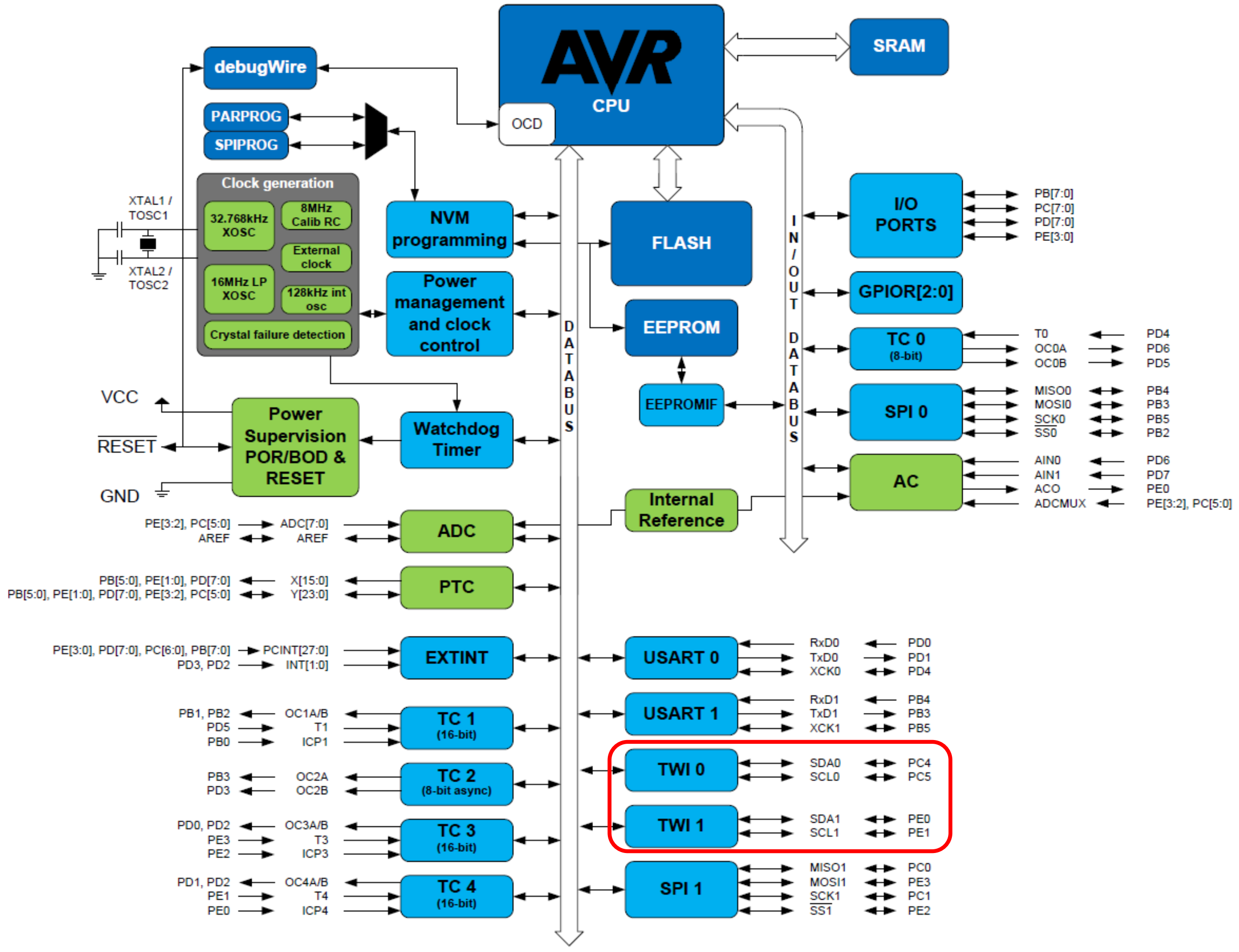
TWI Status Codes for Master Receiver Mode (2)

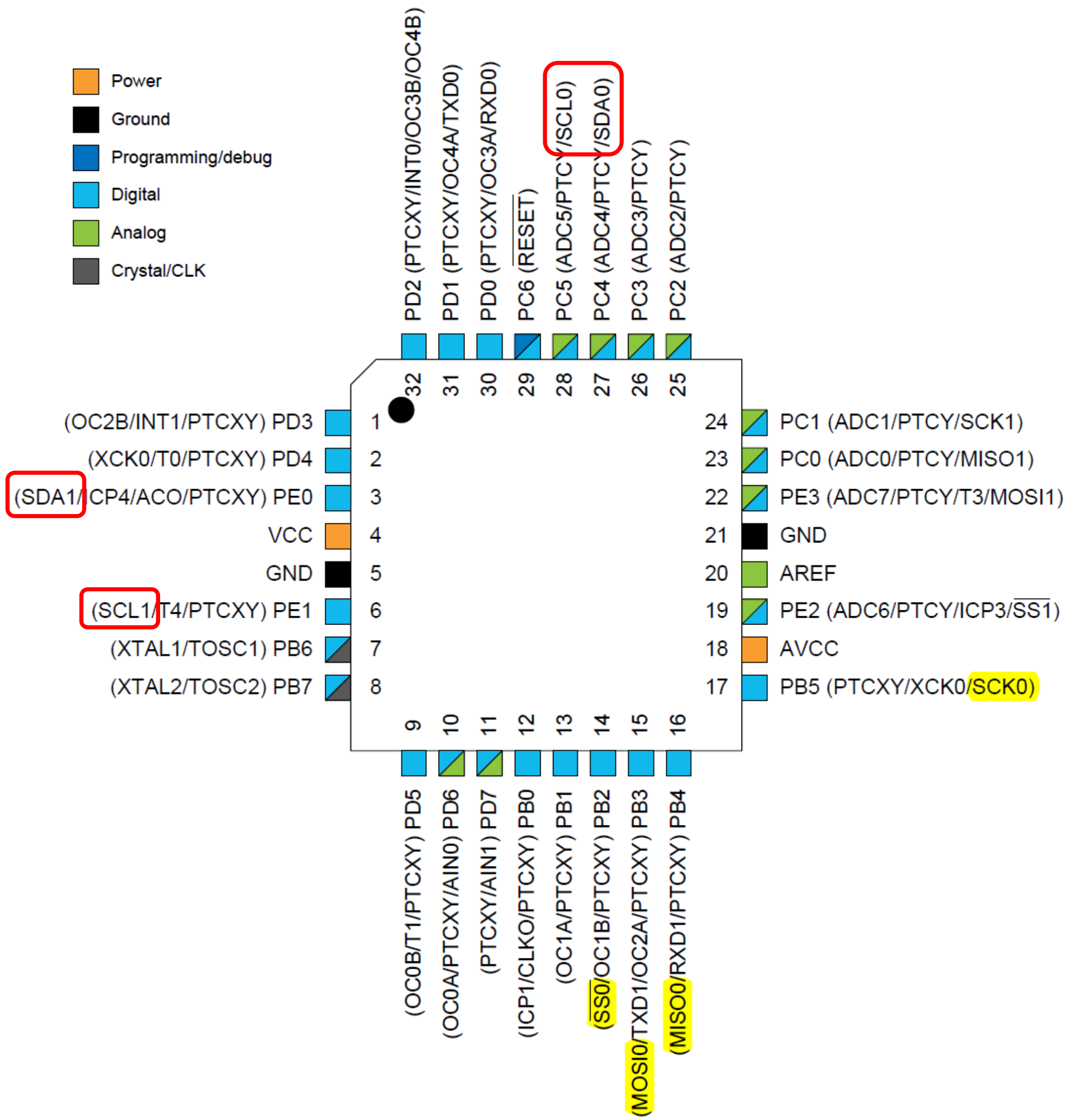
Status Code (TWSRn) Prescaler bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCRn				
			STA	STO	TWINT	TWEA	
0x40	SLA+R has been transmitted; ACK has been received	No TWDRn action	0	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.

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

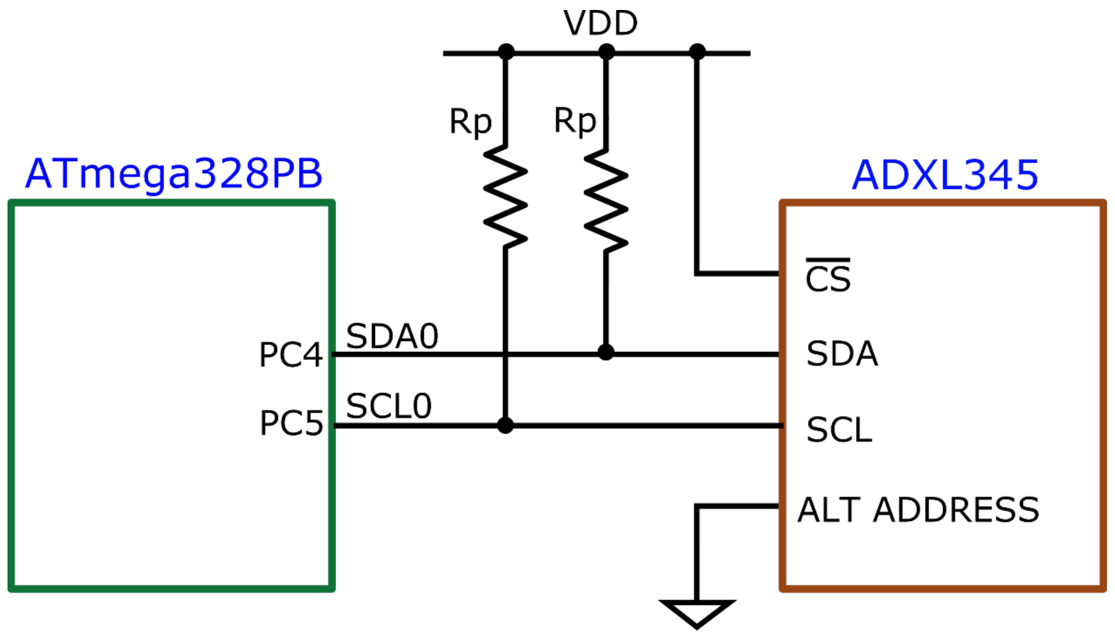
TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0
0	0	0	0	1	1	0	0

$00001100_2 = 12_{10}$

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 (1)

- Specifications:
  - ➢ CPU clock: 8 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 (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 (3)

ADXL345 I2C Device Addressing (2)

Single-Byte Read

Master	S		Slave Addr + W		Register Addr		Rs		Slave Addr + R				N		P
Slave				A		A				A		Data			

Multiple-Byte Read

Master	S		Slave Addr + W		Register Addr		Rs		Slave Addr + R				A		N		P
Slave				A		A				A		Data		Data			

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

TWI0 Example (4) – Initialize ATmega328PB TWI0

```c
#define SLA_W  0xA6
#define SLA_R  0xA7

int main(void)
{
  uint8_t id;
  uint8_t buff[6];
  int accelX, accelY, accelZ;

  uart0_init(38400UL);  // 38,400 bps
  twi0_init();

  // Read ADXL345 ID_Register (reg. addr = 0x00)
  id = twi0_read_adxl345_reg(0x00);
  printf("ADXL345 Chip ID = %X\n", id);

  // Set BW_RATE
  twi0_write_adxl345_reg(0x2C, 0x0A);  // 100 sampleing per sec

  // Set DATA_FORMAT
  twi0_write_adxl345_reg(0x31, 0x08);  // +/-2g (4mg/LSB)

  // Enter MEASUREMENT mode
  twi0_write_adxl345_reg(0x2D, 0x08);
```

```c
  while (1)
  {
    while (!(twi0_read_adxl345_reg(0x30) & 0x80));

    twi0_read_adxl345_reg_multi(6, 0x32, buff);
    accelX = (int)(buff[0] + (buff[1] << 8));
    accelY = (int)(buff[2] + (buff[3] << 8));
    accelZ = (int)(buff[4] + (buff[5] << 8));
    printf("accelX=%d, \taccelY=%d, \taccelZ=%d\n",
              accelX, accelY, accelZ);
  }
}
```

```c
void twi0_init(void)
{
    // Set Bit Rate (400 kHz)
    // SCL freq = F_CPU/(16 + 2 * TWBR * Prescaler)
    // 8,000,000Hz/(16+2*2*1)=8,000,000Hz/20=400kHz
    TWSR0 = 0;  // Prescaler = 1
    TWBR0 = 2;
}
```

TWI0 Example (5) – Read single byte from ADXL345

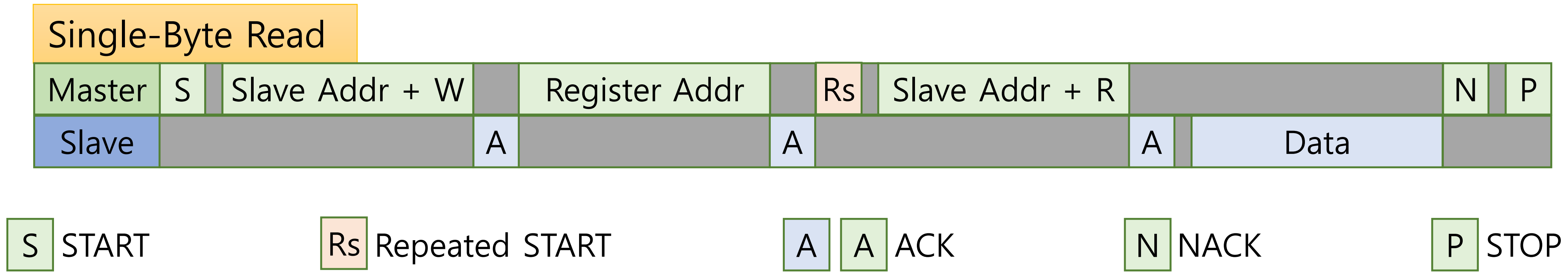
```
uint8_t twi0_read_adxl345_reg(uint8_t reg_addr)
{
   // Send START condition
   TWCR0 = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
   // Wait for the transmission of START condition
   while (!(TWCR0 & (1 << TWINT)));
   if ((TWSR0 & 0xF8) != 0x08)  // Error (Refer to Table 26-3 of Datasheet)
   {
      display_error_code(TWSR0);
      return 0;
   }

   TWDR0 = SLA_W;  // Load SLA_W into TWDR0 Register
   // 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)  // Error (Refer to Table 26-3 of Datasheet)
   {
      display_error_code(TWSR0);
      return 0;
   }

   TWDR0 = reg_addr;  // Load reg_addr to be read into TWDR0 Register
   // Clear TWINT to start transmission of reg_addr.
   TWCR0 = (1<<TWINT) | (1<<TWEN);
   // Wait for the transmission of reg_addr
   while (!(TWCR0 & (1 << TWINT)));
   if ((TWSR0 & 0xF8) != 0x28)  // Error (Refer to Table 26-3 of Datasheet)
   {
      display_error_code(TWSR0);
      return 0;
   }
```

```
   TWCR0 = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);  // Send Repeated START
   // Wait for the transmission of Repeated START
   while (!(TWCR0 & (1 << TWINT)));
   if ((TWSR0 & 0xF8) != 0x10)  // Error (Refer to Table 26-4 of Datasheet)
   {
      display_error_code(TWSR0);
      return 0;
   }

   TWDR0 = SLA_R;  // Load SLA_R into TWDR0 Register
   // 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)  // Error (Refer to Table 26-4 of Datasheet)
   {
      display_error_code(TWSR0);
      return 0;
   }

   // 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)  // Error (Refer to Table 26-4 of Datasheet)
   {
      display_error_code(TWSR0);
      return 0;
   }

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

   return TWDR0;
}
```

TWI0 Example (5-1) – Read single byte from ADXL345

```c
uint8_t twi0_read_adxl345_reg(uint8_t reg_addr)
{
   // 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)  // Error (Refer to Table 26-3 of Datasheet)
   {
      display_error_code(TWSR0);
      return 0;
   }

   TWDR0 = SLA_W;  // Load SLA_W into TWDR0 Register
   // 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)  // Error (Refer to Table 26-3 of Datasheet)
   {
      display_error_code(TWSR0);
      return 0;
   }
```

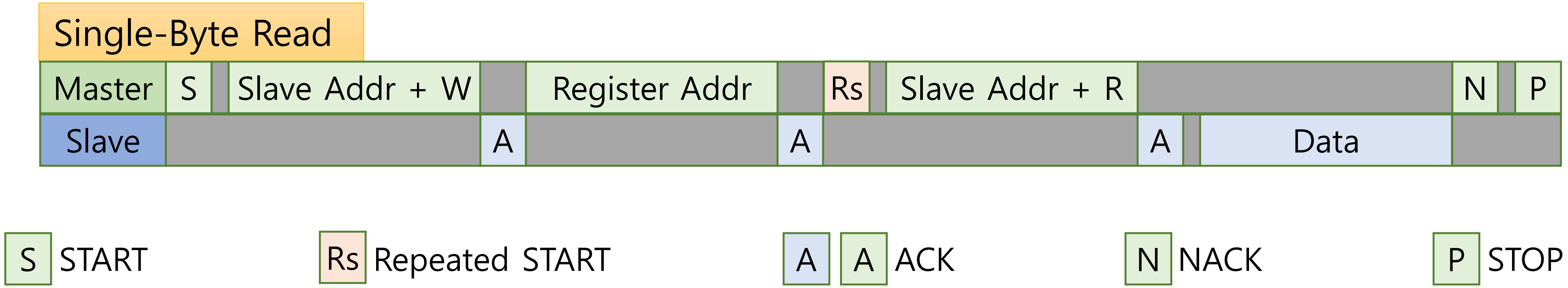
```c
   TWDR0 = reg_addr;  // Load reg_addr to be read into TWDR0 Register
   // Clear TWINT to start transmission of reg_addr.
   TWCR0 = (1<<TWINT) | (1<<TWEN);
   // Wait for the transmission of reg_addr
   while (!(TWCR0 & (1 << TWINT)));
   if ((TWSR0 & 0xF8) != 0x28)  // Error (Refer to Table 26-3 of Datasheet)
   {
      display_error_code(TWSR0);
      return 0;
   }

   TWCR0 = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);  // Send Repeated START
   // Wait for the transmission of Repeated START
   while (!(TWCR0 & (1 << TWINT)));
   if ((TWSR0 & 0xF8) != 0x10)  // Error (Refer to Table 26-4 of Datasheet)
   {
      display_error_code(TWSR0);
      return 0;
   }
```



TWI0 Example (5-2) – Read single byte from ADXL345

```c
   TWDR0 = SLA_R;  // Load SLA_R into TWDR0 Register
   // 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)  // Error (Refer to Table 26-4 of Datasheet)
   {
      display_error_code(TWSR0);
      return 0;
   }
```

```c
   // 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)  // Error (Refer to Table 26-4 of Datasheet)
   {
      display_error_code(TWSR0);
      return 0;
   }

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

   return TWDR0;
}
```



TWI0 Example (6) – Write single byte to ADXL345

```c
void twi0_write_adxl345_reg(uint8_t reg_addr, uint8_t 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)       // Error
   {
      display_error_code(TWSR0);
      return;
   }


   TWDR0 = SLA_W;

   // 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)       // Error
   {
      display_error_code(TWSR0);
      return;
   }
```

```c
   TWDR0 = reg_addr;     // Load reg_addr into TWDR0 Register

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

   // Wait for the transmission of reg_addr
   while (!(TWCR0 & (1 << TWINT)));

   if ((TWSR0 & 0xF8) != 0x28)       // Error
   {
      display_error_code(TWSR0);
      return;
   }


   TWDR0 = data;         // Load data into TWDR0 Register

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

   // Wait for the transmission of data
   while (!(TWCR0 & (1 << TWINT)));

   if ((TWSR0 & 0xF8) != 0x28)       // Error
   {
      display_error_code(TWSR0);
      return;
   }

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

TWI0 Example (6-1) – Write single byte to ADXL345

```c
void twi0_write_adxl345_reg(uint8_t reg_addr, uint8_t 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)        // Error
   {
      display_error_code(TWSR0);
      return;
   }
```

```c
   TWDR0 = SLA_W;

   // 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)        // Error
   {
      display_error_code(TWSR0);
      return;
   }
```

Single-Byte Write

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

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

TWI0 Example (6-2) – Write single byte to ADXL345

```c
  TWDR0 = reg_addr;     // Load reg_addr into TWDR0 Register

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

  // Wait for the transmission of reg_addr
  while (!(TWCR0 & (1 << TWINT)));

  if ((TWSR0 & 0xF8) != 0x28)       // Error
  {
    display_error_code(TWSR0);
    return;
  }
```

```c
  TWDR0 = data;          // Load data into TWDR0 Register

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

  // Wait for the transmission of data
  while (!(TWCR0 & (1 << TWINT)));

  if ((TWSR0 & 0xF8) != 0x28)       // Error
  {
    display_error_code(TWSR0);
    return;
  }

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

Single-Byte Write

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

S START

Rs Repeated START

A A ACK

N NACK

P STOP

TWI0 Example (7) – Read multiple bytes from ADXL345

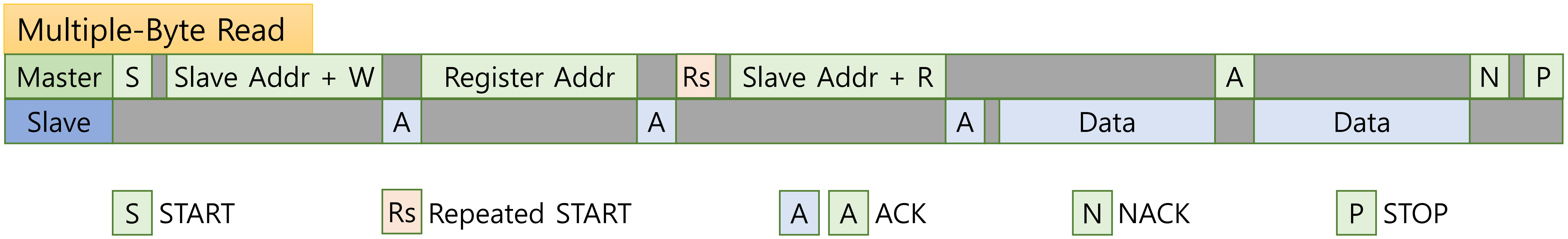
```c
void twi0_read_adxl345_reg_multi(uint8_t num, uint8_t start_addr, uint8_t buff[])
{
   uint8_t i, return_code;

   TWCR0 = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);  // Send START condition
   while (!(TWCR0 & (1 << TWINT)));  // Wait for the transmission of START condition
   if ((TWSR0 & 0xF8) != 0x08)       // Error
   {
      display_error_code(TWSR0);
      return;
   }

   TWDR0 = SLA_W;              // Load SLA_W into TWDR0 Register
   TWCR0 = (1 << TWINT) | (1 << TWEN);      // Start sending of SLA_W.
   while (!(TWCR0 & (1 << TWINT)));         // Wait for the transmission of SLA_W
   if ((TWSR0 & 0xF8) != 0x18)              // Error
   {
      display_error_code(TWSR0);
      return;
   }

   TWDR0 = start_addr;       // Load start_addr into TWDR0 Register
   TWCR0 = (1<<TWINT) | (1<<TWEN);          // Start sending of start_addr
   while (!(TWCR0 & (1 << TWINT)));         // Wait for the transmission of start_addr
   if ((TWSR0 & 0xF8) != 0x28)              // Error
   {
      display_error_code(TWSR0);
      return;
   }

   TWCR0 = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);  // repeated START
   while (!(TWCR0 & (1 << TWINT)));    // Wait for the transmission of repeated START
   if ((TWSR0 & 0xF8) != 0x10)              // Error
   {
      display_error_code(TWSR0);
      return;
   }
```

```c
   TWDR0 = SLA_R;                               // Load SLA_R into TWDR0 Register
   TWCR0 = (1 << TWINT) | (1 << TWEN);          // Start sending of SLA_R
   while (!(TWCR0 & (1 << TWINT)));             // Wait for the transmission of SLA_R
   if ((TWSR0 & 0xF8) != 0x40)                  // Error
   {
      display_error_code(TWSR0);
      return;
   }

   for (i=0; i<num; i++)
   {
      if (i < num-1)
      {
         // 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
      if ((TWSR0 & 0xF8) != return_code)    // Error
      {
         display_error_code(TWSR0);
         return;
      }

      buff[i] = TWDR0;        // Store read data
   }

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

TWI0 Example (7-1) – Read multiple bytes from ADXL345

```c
void twi0_read_adxl345_reg_multi(uint8_t num, uint8_t start_addr, uint8_t buff[])
{
   uint8_t i, return_code;

   TWCR0 = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);  // Send START condition
   while (!(TWCR0 & (1 << TWINT)));  // Wait for the transmission of START condition
   if ((TWSR0 & 0xF8) != 0x08)       // Error
   {
      display_error_code(TWSR0);
      return;
   }

   TWDR0 = SLA_W;              // Load SLA_W into TWDR0 Register
   TWCR0 = (1 << TWINT) | (1 << TWEN);      // Start sending of SLA_W.
   while (!(TWCR0 & (1 << TWINT)));         // Wait for the transmission of SLA_W
   if ((TWSR0 & 0xF8) != 0x18)              // Error
   {
      display_error_code(TWSR0);
      return;
   }
```

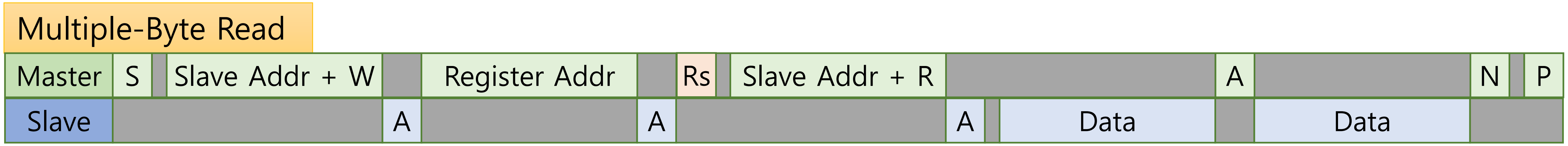
```c
   TWDR0 = start_addr;       // Load start_addr into TWDR0 Register
   TWCR0 = (1<<TWINT) | (1<<TWEN);          // Start sending of start_addr
   while (!(TWCR0 & (1 << TWINT)));         // Wait for the transmission of start_addr
   if ((TWSR0 & 0xF8) != 0x28)              // Error
   {
      display_error_code(TWSR0);
      return;
   }

   TWCR0 = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);  // repeated START
   while (!(TWCR0 & (1 << TWINT)));     // Wait for the transmission of repeated START
   if ((TWSR0 & 0xF8) != 0x10)              // Error
   {
      display_error_code(TWSR0);
      return;
   }
```



TWI0 Example (7-2) – Read multiple bytes from ADXL345

```c
   TWDR0 = SLA_R;                            // Load SLA_R into TWDR0 Register
   TWCR0 = (1 << TWINT) | (1 << TWEN);      // Start sending of SLA_R
   while (!(TWCR0 & (1 << TWINT)));          // Wait for the transmission of SLA_R
   if ((TWSR0 & 0xF8) != 0x40)               // Error
   {
      display_error_code(TWSR0);
      return;
   }

   for (i=0; i<num; i++)
   {
      if (i < num-1)
      {
         // Clear TWINT to start reception. ACK will be returned
         TWCR0 = (1 << TWINT) | (1 << TWEN) | (1 << TWEA);
         return_code = 0x50;
      }
```

```c
      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
      if ((TWSR0 & 0xF8) != return_code)    // Error
      {
         display_error_code(TWSR0);
         return;
      }

      buff[i] = TWDR0;        // Store read data
   }

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



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

End