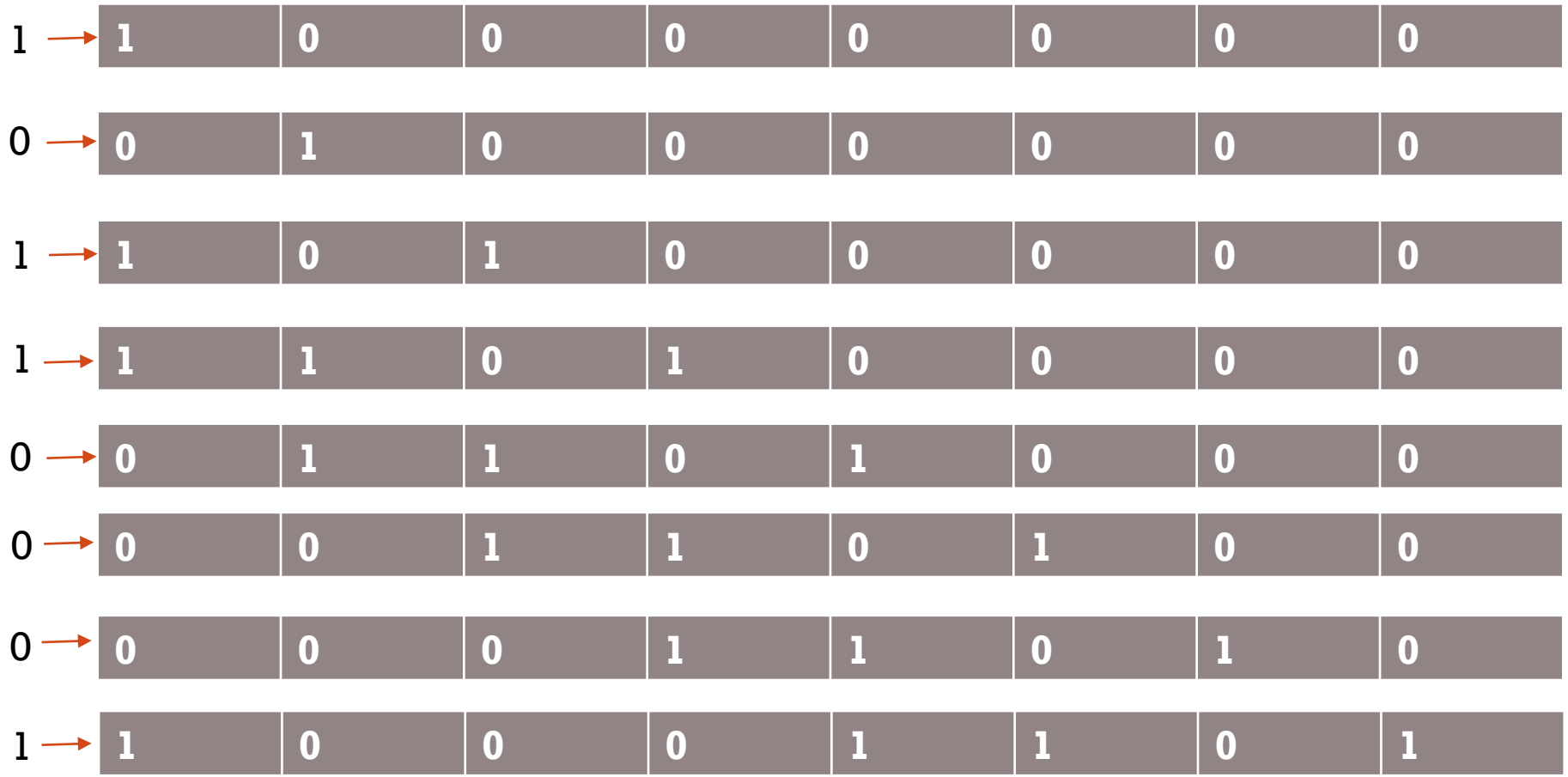
EMBEDDED SYSTEMS CMPE-453

Department of Computer Engineering

Serial Communication-5

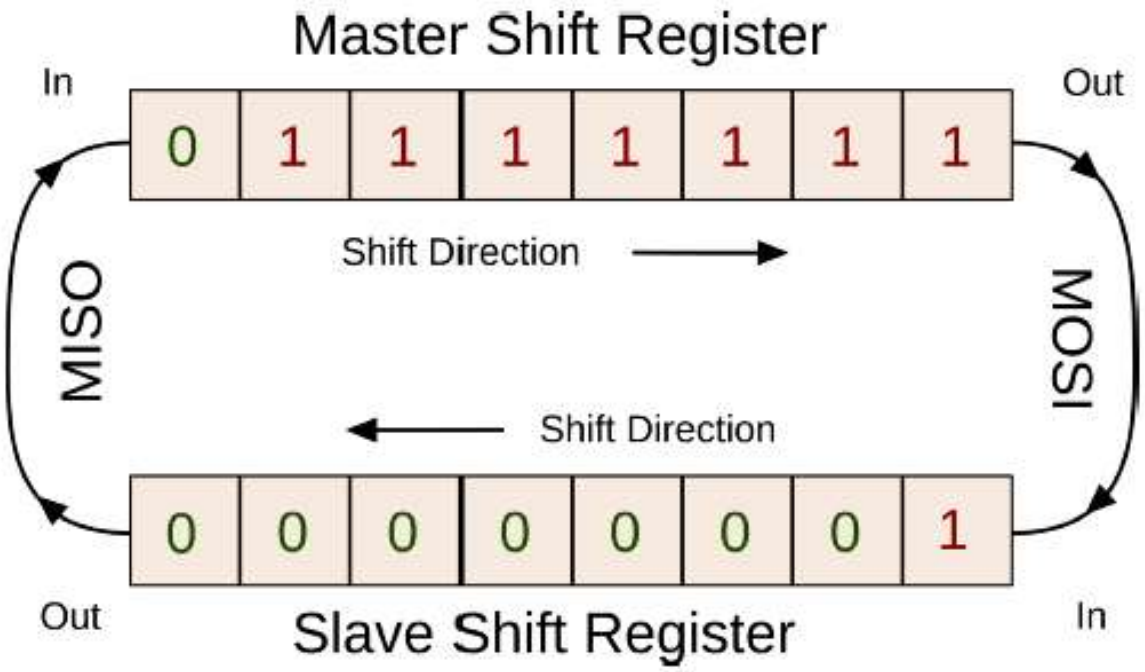
SHIFT REGISTERS



Serial in - parallel out

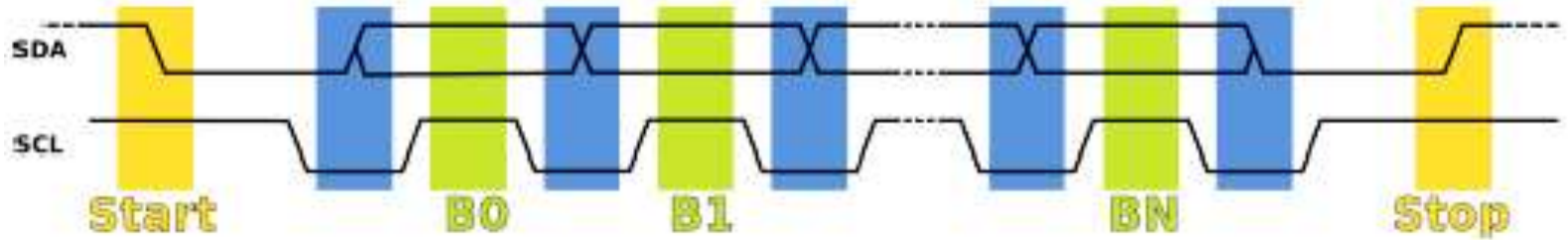
Parallel in - Serial out

SPI SHIFT REGISTERS



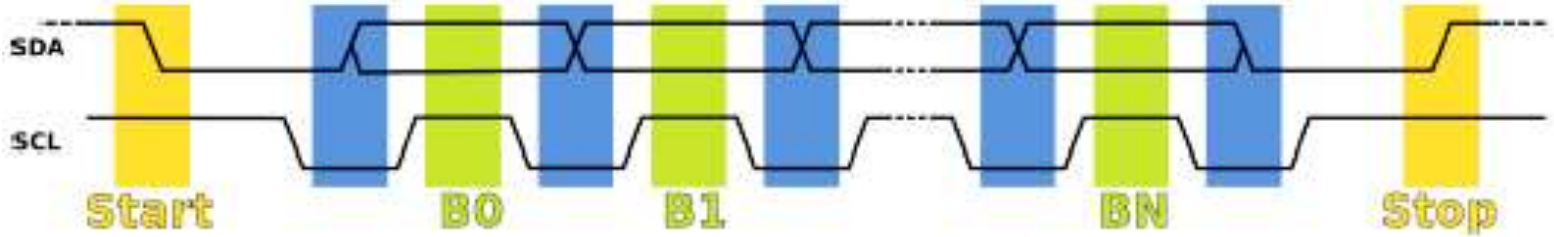
- Start : Parallel-loaded with the data
- Ex: Slave with all zeros, master with all ones
- Processors read from registers, in parallel
- Once processed, registers are reloaded.

I2C



- 2 wires
  - SCL – Serial Clock
  - SDA – Serial Data
- I2C is a synchronous protocol, a clock master that's controlling the communication by sending out the timing signal.
- I2C data line can change between high and low states only while the clock line is low
- Data is to be read only while the clock line is high

I2C

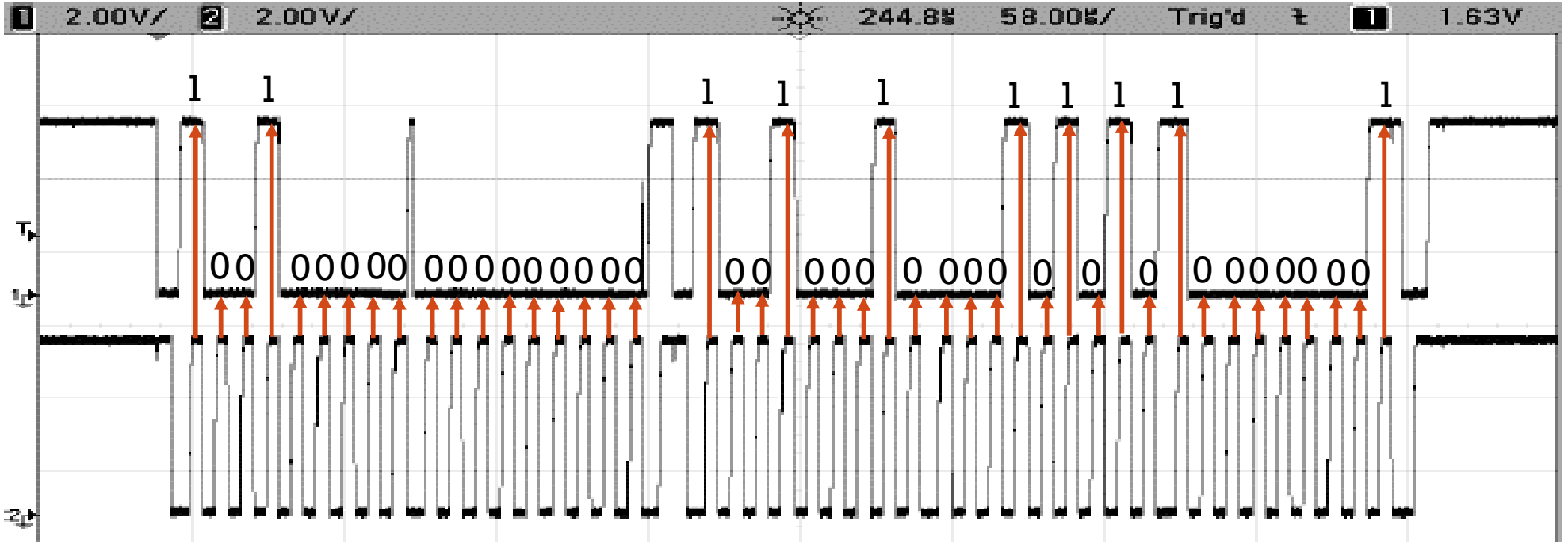


- Multiple devices use the same lines, therefore need an addressing scheme.
- Each device has specific 7-bit address
- Eight bit : whether the master or slave is going to send data over the data line next.
- If AVR sends a start, then the address of a slave device, and then a zero, the slave device knows that it's going to receive a byte from the AVR
- If the last bit after the address is a one, then the slave is responsible for sending data; the master reads data off the bus.

I2C ACK OR NACK

- Receiver device (whether it's the master or slave) is responsible for acknowledging that it has received the byte.
- A ninth bit is sent after every 8-bit byte.
- A zero or low ("ACK") indicates that the byte is acknowledged and signals the sender to continue sending.
- A high bit ("NACK") indicates
  - There was an error
  - The transmitter should stop sending
  - The whole communication is about to end, depending on the circumstances.
- Possible to detect if there's been an error in the transmission
- E.g. electrically noisy environment. (in contrast to SPI, where everything's just assumed to work.)

I2C (AVR-LM75 TEMPERATURE SENSOR)



- LM75's address (1001000)
- 0 for writing mode
- 0 for ack

- Second byte from AVR is all zeros 00000000 (temperature register) 0 for ack (from LM75)
- LM75's address (1001000), 1 for (slave is responsible), 0 for ack
- 00010101 is the data (21 degrees – Most Sign. Byte), 0 for ack (from AVR)
- 10000000 is the data (Least Sign. Byte)
- 1 for nack

```c
#include "i2c.h"
void initI2C(void) {
TWBR = 32;                          /* set bit rate, see p. 242 */
                                    /* 8MHz / (16+2*TWBR*1) ~= 100kHz */
TWCR |= (1 << TWEN);                /* enable */
}
void i2cWaitForComplete(void) {
loop_until_bit_is_set(TWCR, TWINT);
}
void i2cStart(void) {
TWCR = (_BV(TWINT) | _BV(TWEN) | _BV(TWSTA));
i2cWaitForComplete();
}
void i2cStop(void) {
TWCR = (_BV(TWINT) | _BV(TWEN) | _BV(TWSTO));
}
uint8_t i2cReadAck(void) {
TWCR = (_BV(TWINT) | _BV(TWEN) | _BV(TWEA));
i2cWaitForComplete();
return (TWDR);
}
uint8_t i2cReadNoAck(void) {
TWCR = (_BV(TWINT) | _BV(TWEN));
i2cWaitForComplete();
return (TWDR);
}
void i2cSend(uint8_t data) {
TWDR = data;
TWCR = (_BV(TWINT) | _BV(TWEN));            /* init and enable */
i2cWaitForComplete();
}
```

```c
#include <avr/io.h>
#include <util/delay.h>
#include <avr/power.h>
#include "pinDefines.h"
#include "USART.h"
#include "i2c.h"
// -------- Defines -------- //
#define LM75_ADDRESS_W 0b10010000
#define LM75_ADDRESS_R 0b10010001
#define LM75_TEMP_REGISTER 0b00000000
#define LM75_CONFIG_REGISTER 0b00000001
#define LM75_THYST_REGISTER 0b00000010
#define LM75_TOS_REGISTER 0b00000011
// -------- Functions --------- //
int main(void) {
uint8_t tempHighByte, tempLowByte;
// -------- Inits --------- //
clock_prescale_set(clock_div_1); /* 8MHz */
initUSART();
printString("\r\n==== i2c Thermometer ====\r\n");
initI2C();
// ------ Event loop ------ //
```

(_BV(BIT) : bit value

```
#include "i2c.h"
void initI2C(void) {
TWBR = 32;                          /* set bit rate, see p. 242 */
                                    /* 8MHz / (16+2*TWBR*1) ~= 100kHz */
TWCR |= (1 << TWEN);                /* enable */
}
void i2cWaitForComplete(void) {
loop_until_bit_is_set(TWCR, TWINT);
}
void i2cStart(void) {
TWCR = (_BV(TWINT) | _BV(TWEN) | _BV(TWSTA));
i2cWaitForComplete();
}
void i2cStop(void) {
TWCR = (_BV(TWINT) | _BV(TWEN) | _BV(TWSTO));
}
uint8_t i2cReadAck(void) {
TWCR = (_BV(TWINT) | _BV(TWEN) | _BV(TWEA));
i2cWaitForComplete();
return (TWDR);
}
uint8_t i2cReadNoAck(void) {
TWCR = (_BV(TWINT) | _BV(TWEN));
i2cWaitForComplete();
return (TWDR);
}
void i2cSend(uint8_t data) {
TWDR = data;
TWCR = (_BV(TWINT) | _BV(TWEN));            /* init and enable */
i2cWaitForComplete();
}
```

```
// ------ Event loop ------ //
while (1) {
/* To set register, address LM75 in write mode */
i2cStart();
i2cSend(LM75_ADDRESS_W);
i2cSend(LM75_TEMP_REGISTER);
i2cStart();
/* Setup and send address, with read bit */
i2cSend(LM75_ADDRESS_R);
/* Now receive two bytes of temperature */
tempHighByte = i2cReadAck();
tempLowByte = i2cReadNoAck();
i2cStop();
// Print it out nicely over serial for now...
printByte(tempHighByte);
if (tempLowByte & _BV(7)) {    //_BV(), defined in io.h
printString(".5\r\n");
}                                          #define _BV(bit)   (1 << (bit))
else {
printString(".0\r\n");
}
/* Once per second */
_delay_ms(1000);
} /* End event loop */
return (0); /* This line is never reached */
}
```