Introduction to Design (2)

Microcontrollers and Interfacing

Week 13

Serial communication with devices:
Inter-Integrated Circuit (I^2C) and
Serial Peripheral Interconnect (SPI) protocols

京都先端科学大学
KYOTO UNIVERSITY of ADVANCED SCIENCE
Department of Mechanical and Electrical System Engineering

this week

history of board-level serial communications

I^2C: topology, messages

SPI: topology, data exchange

example device: MCP3204 quad 12-bit ADC with SPI

- timing
- circuit
- layout

the SPI library

history

1982: Philips was putting digital integrated circuits (ICs) into TV sets

- TV control (channel buttons on the front, etc.) had to communicate with the ICs
- ICs had to communicate with each other
- ‘mini serial network’ developed: Inter-Integrated Circuit (I^2C) protocol
- very good for configuring devices with control registers

1985: Motorola released a microcontroller based on the 68000 architecture

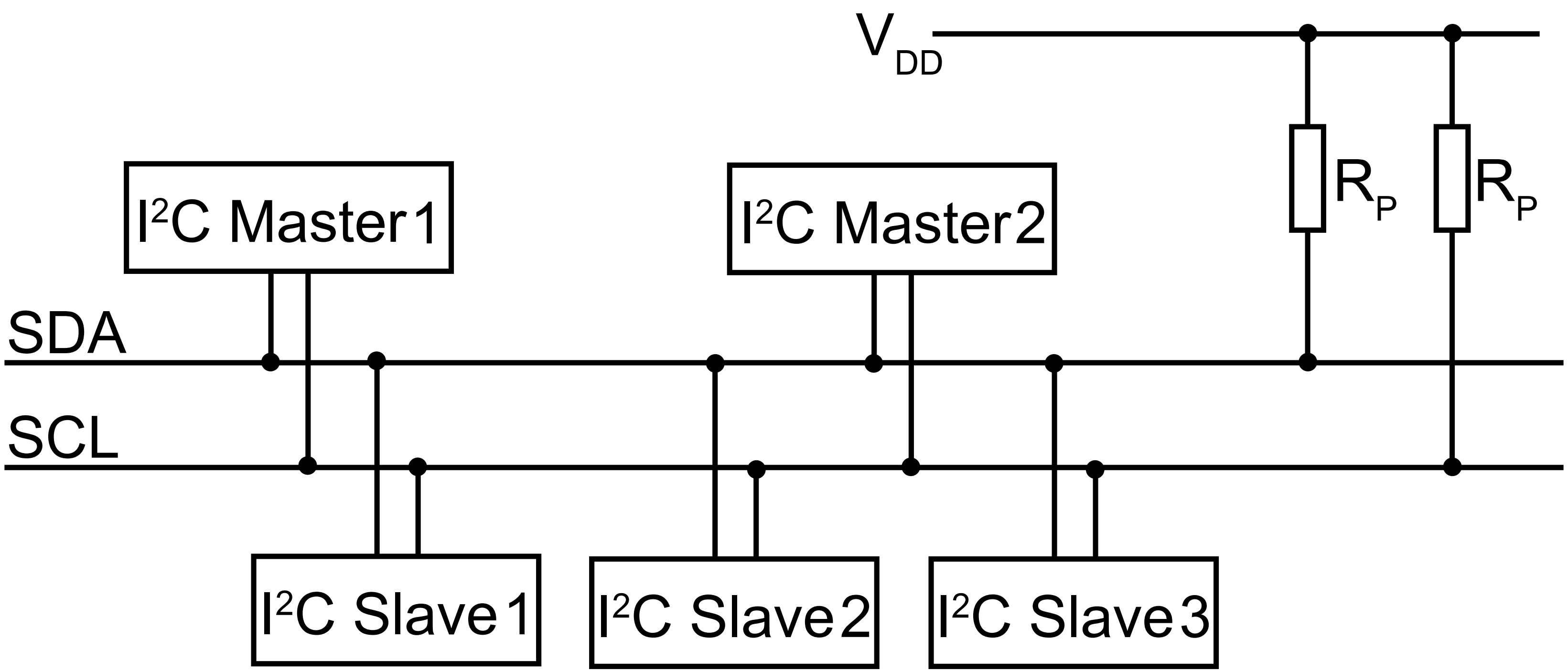
- needed a way to communicate with diverse, fast peripherals
- simplicity and speed were very important
- point-to-point protocol developed: Serial Peripheral Interface (SPI)
- very good for streaming data to/from external devices

I^{2}C topology

two wires: data (SDA) and clock (SCL)

bus-based: devices send ‘messages’ to each other

- message begins with destination device address
- specifies whether the message is a read or write operation
- master controls clock
- master and slave can both transmit to exhange a byte followed by an acknowledgement bit



I²C messages

127 devices can be connected

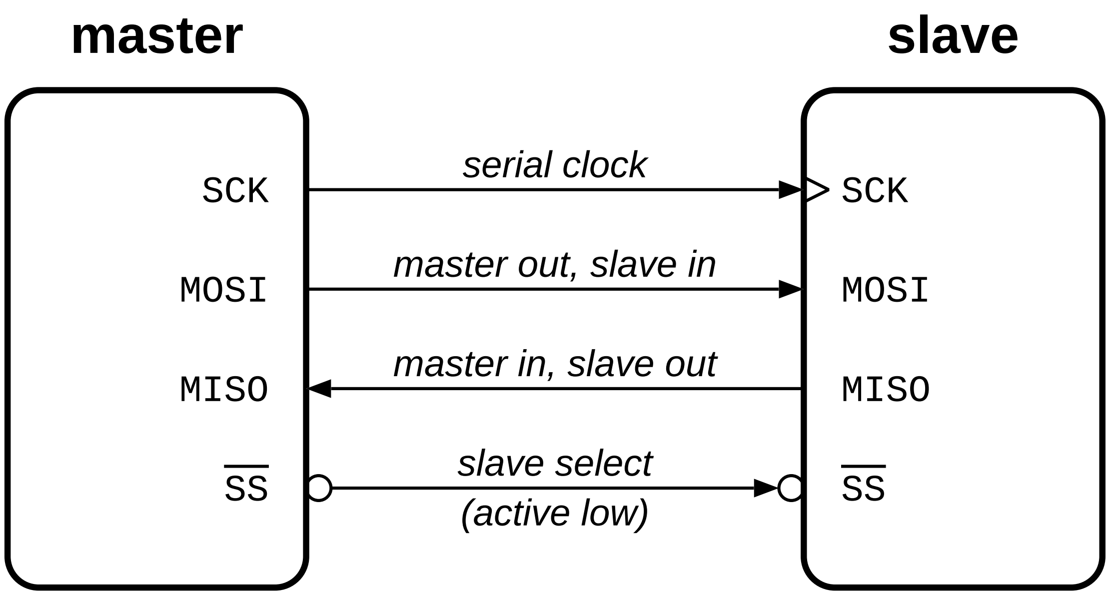
- each device has an address
- messages are sent to a specific device
- messages are byte oriented, and either read or write (not both)
- the protocol is half-duplex

SCL SDA			(bus claim)
seven-bit slave address	R/$\overline{\text{W}}$	ack	$1\times$ slave address and direction byte
eight data bits		ack	$N\times$ data bytes
SCL SDA			(bus release)

SPI topology

simple case: one master, one slave

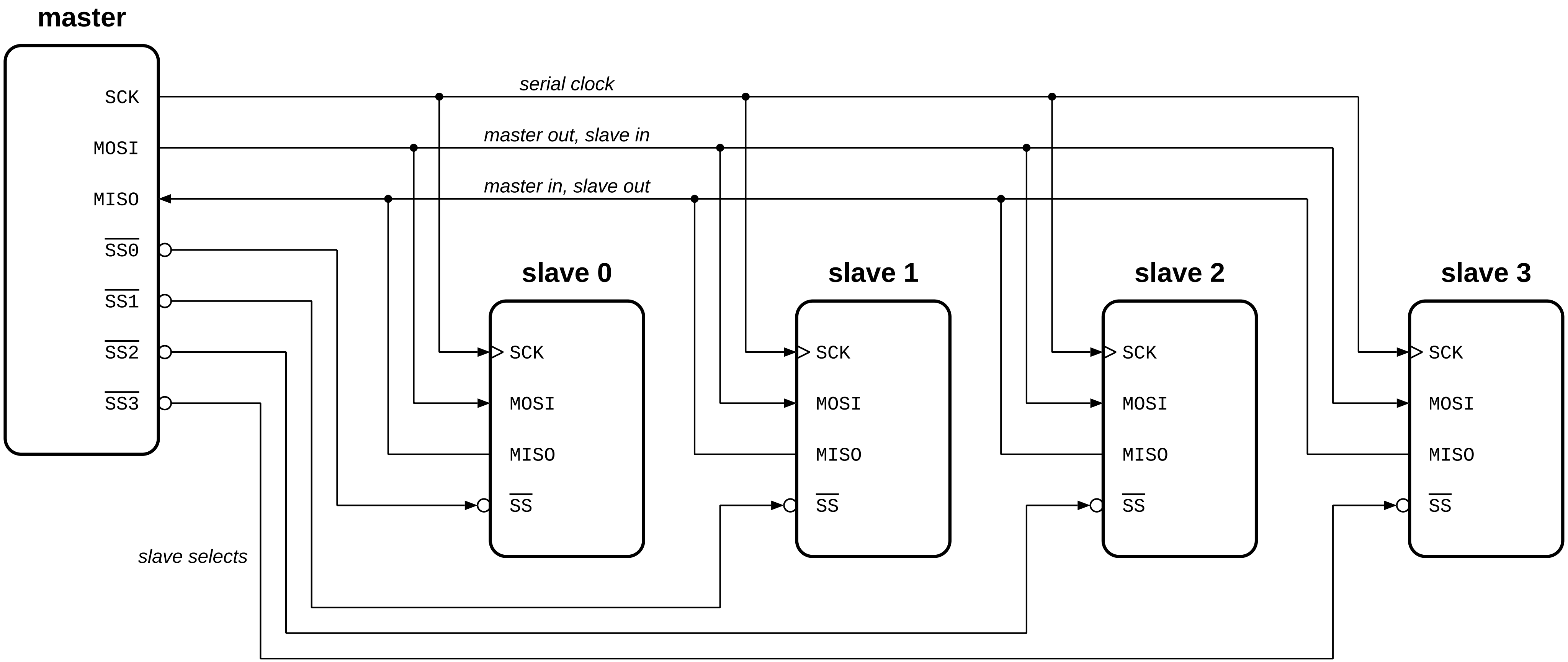
- master controls slave select and clock
- two data lines: MOSI (master→slave) and MISO (slave→master)
- data clocked on *both* lines every clock cycle
- stream of bits (no byte orientation), and protocol is full-duplex



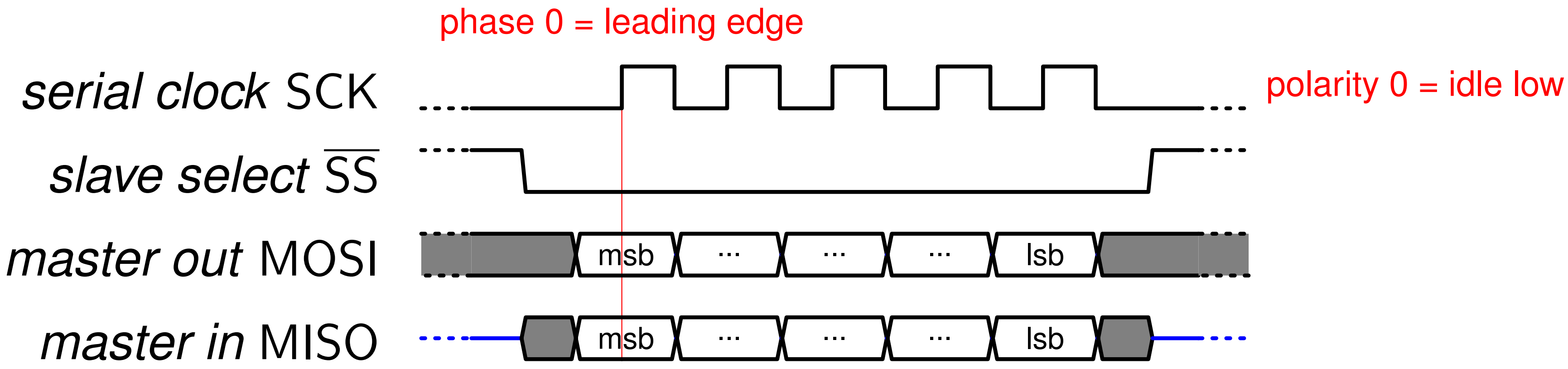
SPI with multiple slaves

common case: one master, a few slaves

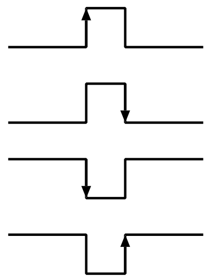
- MISO is high-impedance (disconnected) unless slave selected
- only one slave select can be active at any given time
- needs additional slave select wire for each additional slave
  - (can be avoided with shift registers, binary decoders, etc.)



SPI data exchange

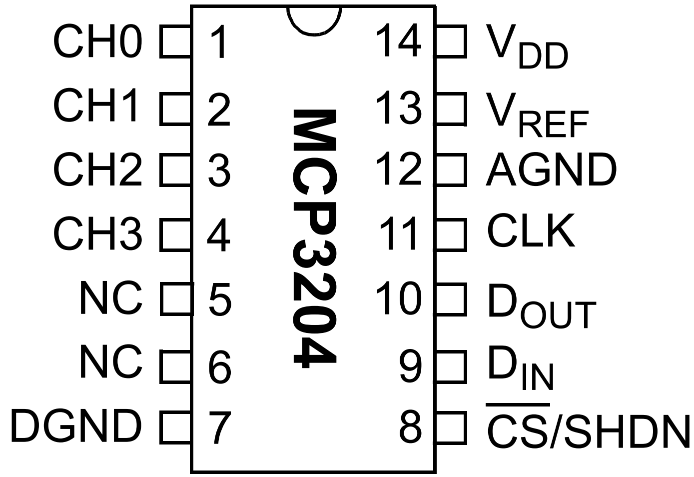


SPI clock mode (polarity, phase)	clock idles	active edge
0 (0, 0)	low	leading (rising)
1 (0, 1)	low	trailing (falling)
2 (1, 0)	high	leading (falling)
3 (1, 1)	high	trailing (rising)



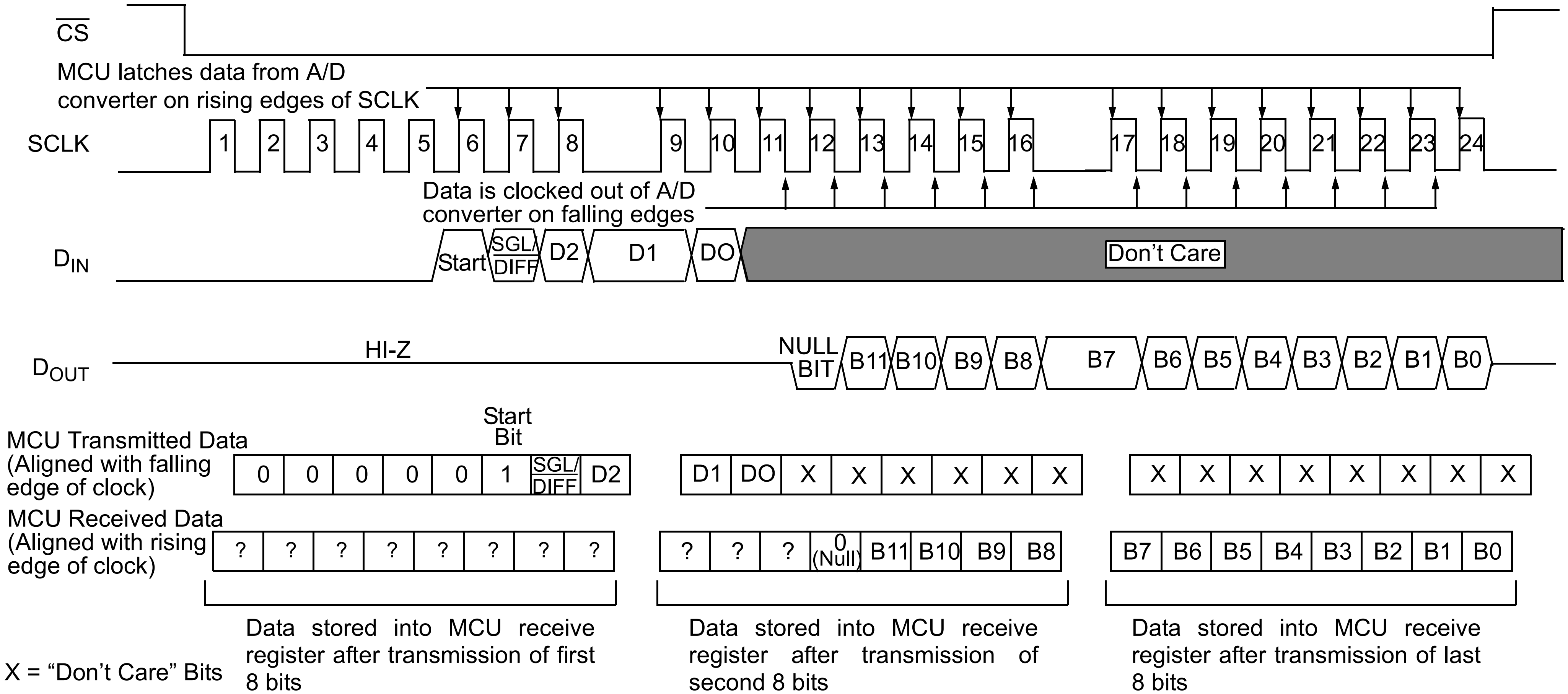
SPI example: MCP3204

4-channel, 12-bit A/D converter

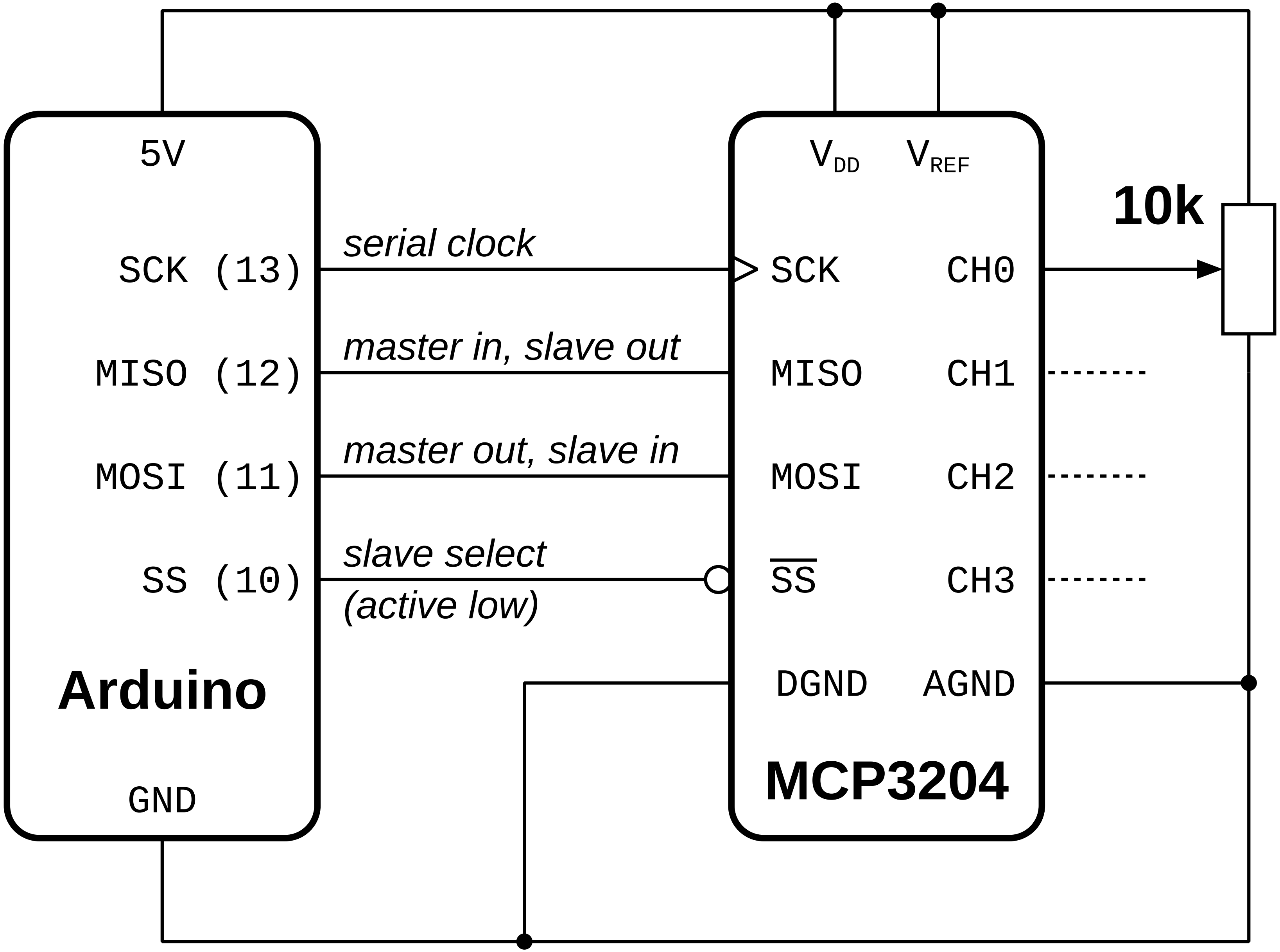


V_{DD}	5 V power supply
DGND	0 V digital ground
AGND	0 V analogue ground
V_{REF}	reference voltage, sets the upper limit of input voltage (corresponding to the maximum digital A/D output value)
CH0–CH4	the four analogue input channels
CLK	SPI serial clock input
D_{IN}	SPI serial data input (equivalent to MOSI)
D_{OUT}	SPI serial data output (equivalent to MISO)
$\overline{CS}$	SPI active-low chip select (equivalent to $\overline{SS}$)

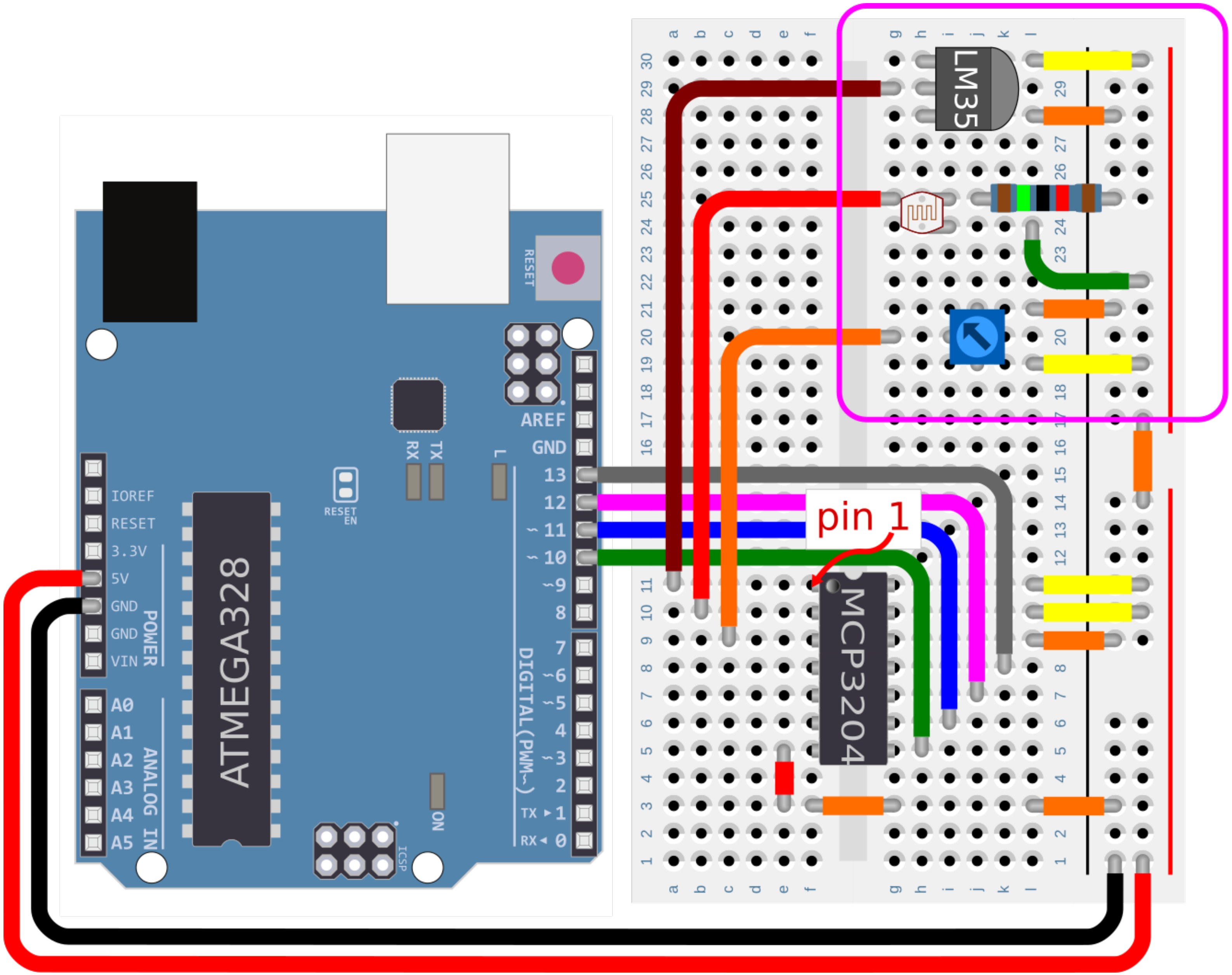
SPI example timing



SPI example circuit



SPI example layout



Put any analogue sources that you want here. These are only some suggestions.

the SPI library

Arduino has hardware support for SPI, and a library for accessing it

- include the SPI library

```
#include <SPI.h>
```

- configure the SPI library

```
void setup()
{
  SPI.begin();
  SPI.setClockDivider(SPI_CLOCK_DIV16); // 1 MHz
  SPI.setDataMode(SPI_MODE0); // idle low, leading edge
  SPI.setBitOrder(MSBFIRST);
}
```

- use the SPI library to transfer 8 bits at a time

```
byte misoValue = SPI.transfer(mosiValue);
```

  `misoValue` is read in at the same time that `mosiValue` is written out

- note: you are responsible for managing any ‘chip select’ signals!

‘bit banging’

when no hardware support for SPI (or any other protocol)

- assign some digital I/O pins to the needed signals
- implement the protocol manually, by writing/reading the pins
- this is known as ‘bit banging’

SPI signals

```
#define SSN  10 // slave select pin
#define MOSI 11 // master out (slave in) pin
#define MISO 12 // master in (slave out) pin
#define SCK  13 // serial clock pin
```

SPI configuration

```
void setup() {
  pinMode(SSN,  OUTPUT);   digitalWrite(SSN, HIGH); // slave inactive
  pinMode(SCK,  OUTPUT);   digitalWrite(SCK, LOW);  // clock idle
  pinMode(MOSI, OUTPUT);
  pinMode(MISO, INPUT);
}
```

‘bit banging’

write a single bit to SPI device

```
void sendBit(unsigned char bit)
{
  digitalWrite(MOSI, bit & 1);   // write value to device
  digitalWrite(SCK, HIGH);       // clock data into device
  digitalWrite(SCK, LOW);        // clock idle
}
```

read a single bit from SPI device

```
int recvBit(void)
{
  digitalWrite(SCK, HIGH);       // clock data out of the device
  int bit = digitalRead(MISO);   // read value from device
  digitalWrite(SCK, LOW);        // clock idle
  return bit;
}
```

‘bit banging’

example transaction: perform Analogue to Digital Conversion

```
int readADC(int channel) {
  digitalWrite(SSN, LOW);         // slave select active

  sendBit(1);                     // start bit
  sendBit(1);                     // single-ended mode

  sendBit(channel >> 2);          // ms bit
  sendBit(channel >> 1);
  sendBit(channel);               // ls bit

  sendBit(0);                     // discard empty result bit
  sendBit(0);                     // discard null result bit

  int advalue = 0;
  for (int i= 0;  i < 12;  ++i)
    advalue = (advalue << 1) + recvBit();

  digitalWrite(SSN, HIGH);        // slave select inactive

  return advalue;
}
```