UCDAVIS
UNIVERSITY OF CALIFORNIA

Internet of Things

Senior Design Project Course

Digital Communication protocols

Part 2

Lecturer: Avesta Sasan

University of California Davis

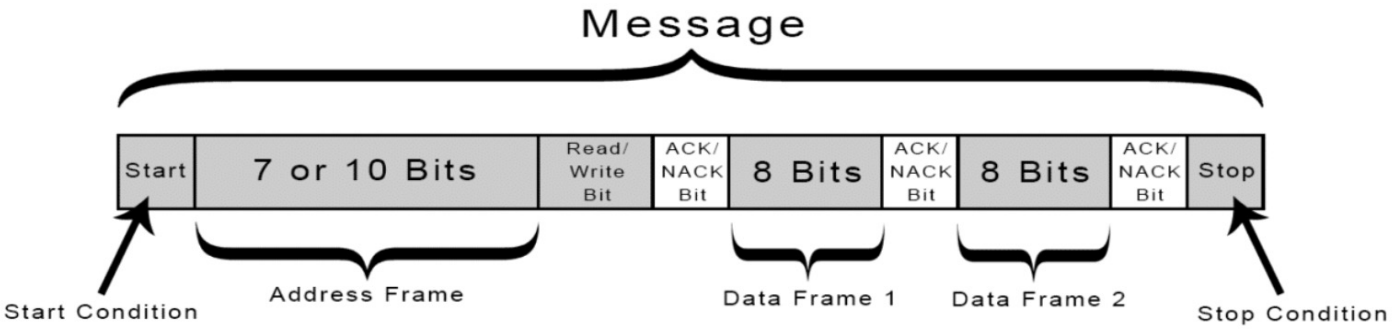
Digital Sensors (Review)

- To read digital sensors you need to use the sensor's **interface protocols**

- Some common interface protocols are:
  - Universal Asynchronous Receiver/Transmitor (**UART**)
    - Asynchronous
    - https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter 
  - Serial Peripheral Interface **(SPI)**
    - Synchronous
    - Single Master Multiple Slaves. https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus 
  - Inter-Integrated Circuit **(I2C)**
    - Synchronous
    - Multi Master Multiple Slaves. https://en.wikipedia.org/wiki/I2C 

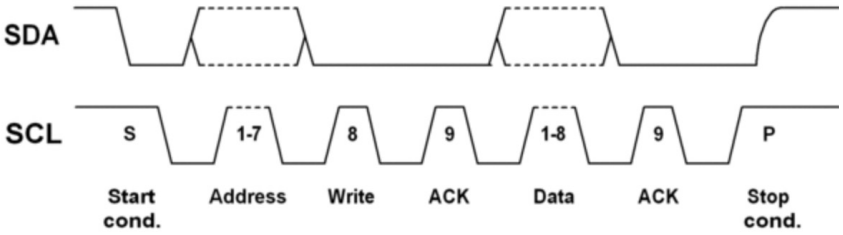
I2C Device Addressing

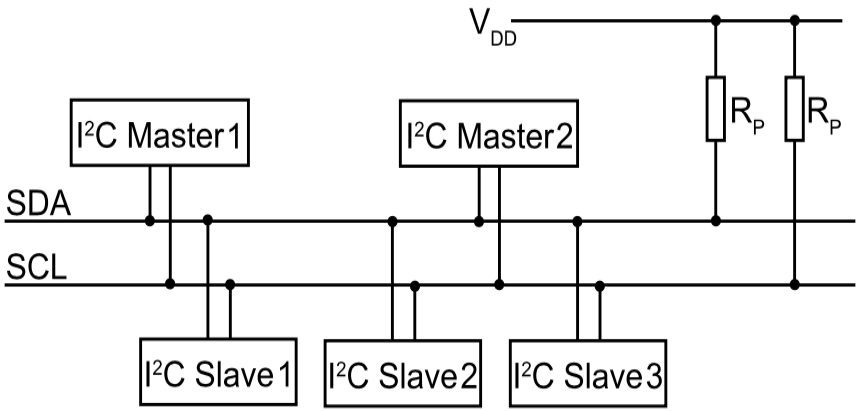
- I2C transfer data in form of messages.
- Messages are broken down into segments shown in the image below
  - **Address Frame:** A 7 or 10 bit sequence unique to each slave that identifies the slave when the master wants to talk to it.
  - **Start Condition:** The SDA line switches from a high voltage level to a low voltage level *before* the SCL line switches from high to low.
  - **Stop Condition:** The SDA line switches from a low voltage level to a high voltage level *after* the SCL line switches from low to high.
  - **Read/Write Bit:** A single bit specifying whether the master is sending data to the slave (low voltage level) or requesting data from it (high voltage level).
  - **ACK/NACK Bit:** Each frame in a message is followed by an acknowledge/no-acknowledge bit. If an address frame or data frame was successfully received, an ACK bit is returned to the sender from the receiving device.



I2C Device Addressing

- The **first byte** of an I2C transfer always contains the **slave address** and the **data direction.**
- Usually the address is transferred with the **MSB first.**



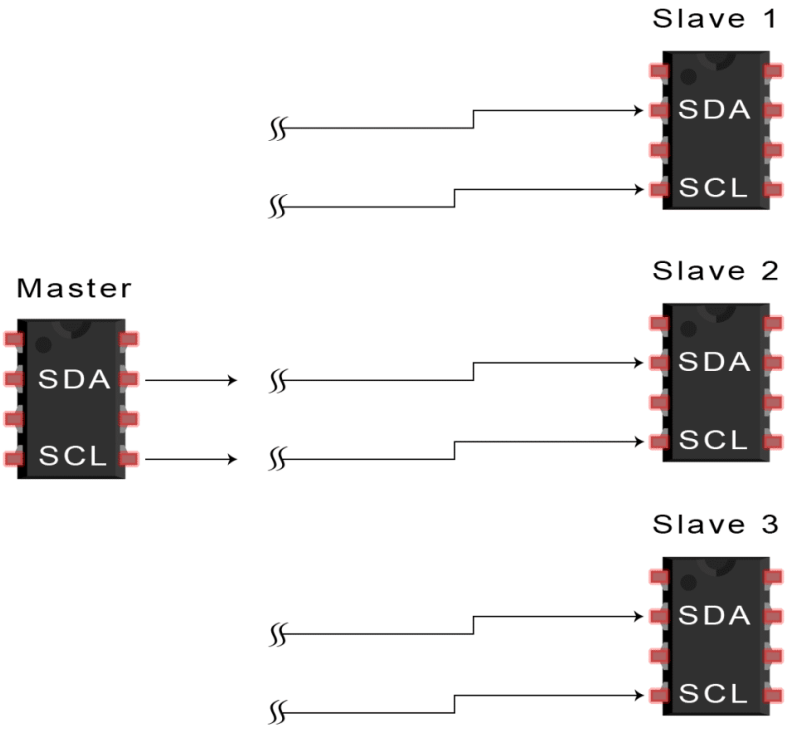


- Following web page is a good source to learn how to use I2C:
  - https://www.i2c-bus.org/addressing/

I2C Device Addressing

- The master sends the start condition to every connected slave:
- **How:** Switching the SDA line from a high voltage level to a low voltage level *before* switching the SCL line from high to low:

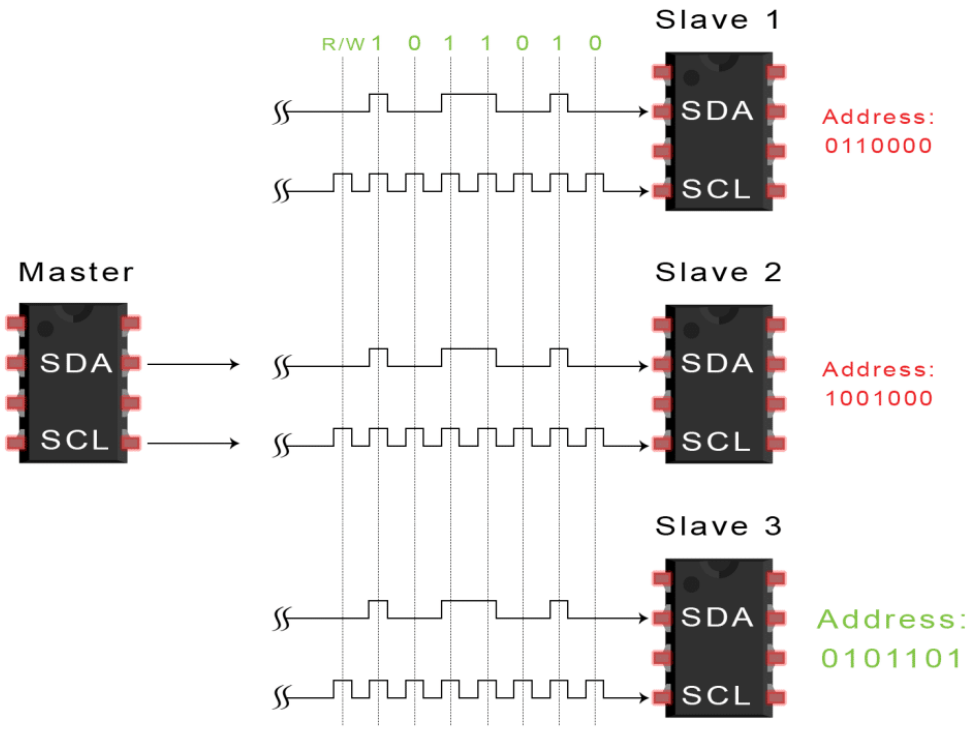
1



I2C Device Addressing

- The master sends each slave the 7 or 10 bit address of the slave it wants to communicate with, along with the read/write bit:

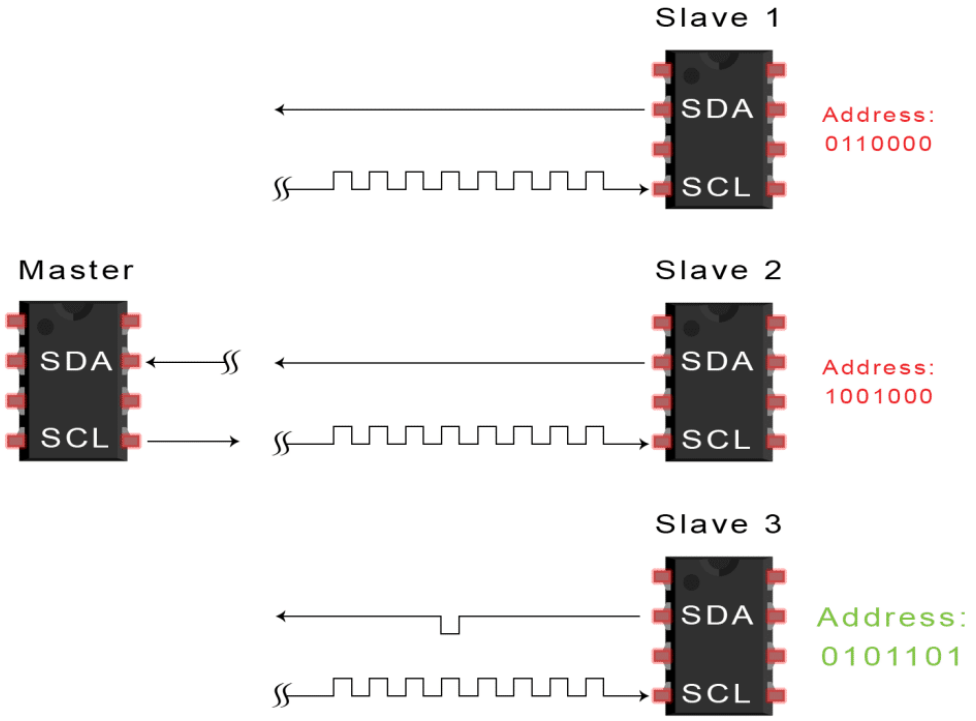
2



I2C Device Addressing

- Each slave compares the address sent from the master to its own address.
  - If matches, the slave returns an ACK bit by pulling the SDA line low for one bit.
  - If does not match the slave's own address, the slave leaves the SDA line high.

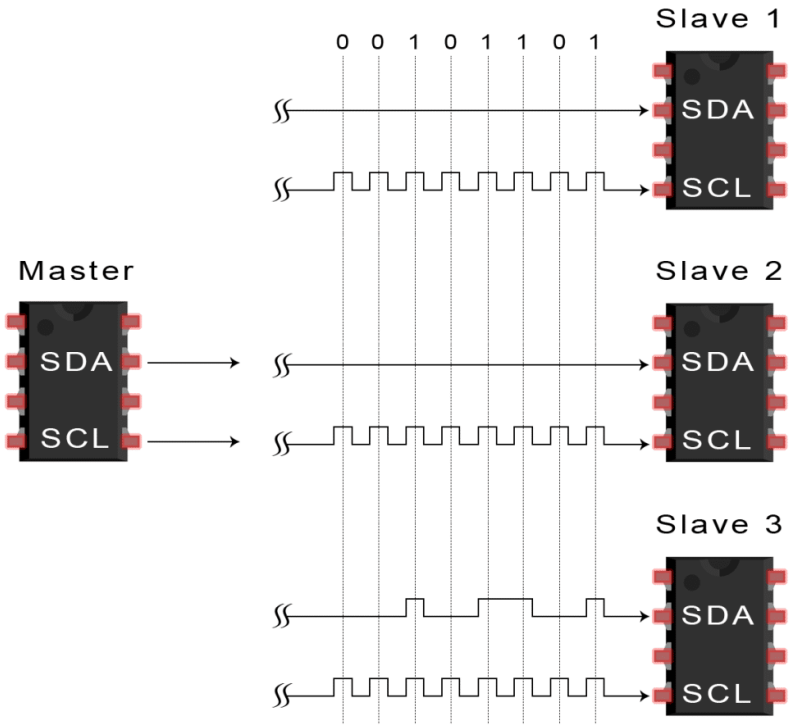
3



I2C Device Addressing

- The master sends or receives the data frame:
- Can you tell if the data is being sent from the master to slave or from slave to the master in this example?

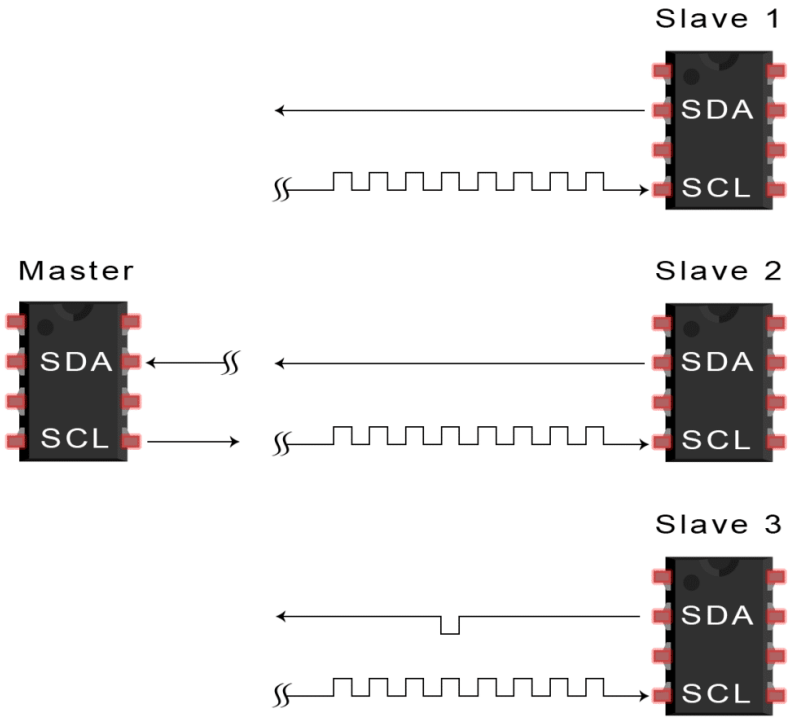
4



I2C Device Addressing

- After each data frame has been transferred, the receiving device returns another ACK bit to the sender to acknowledge successful receipt of the frame:

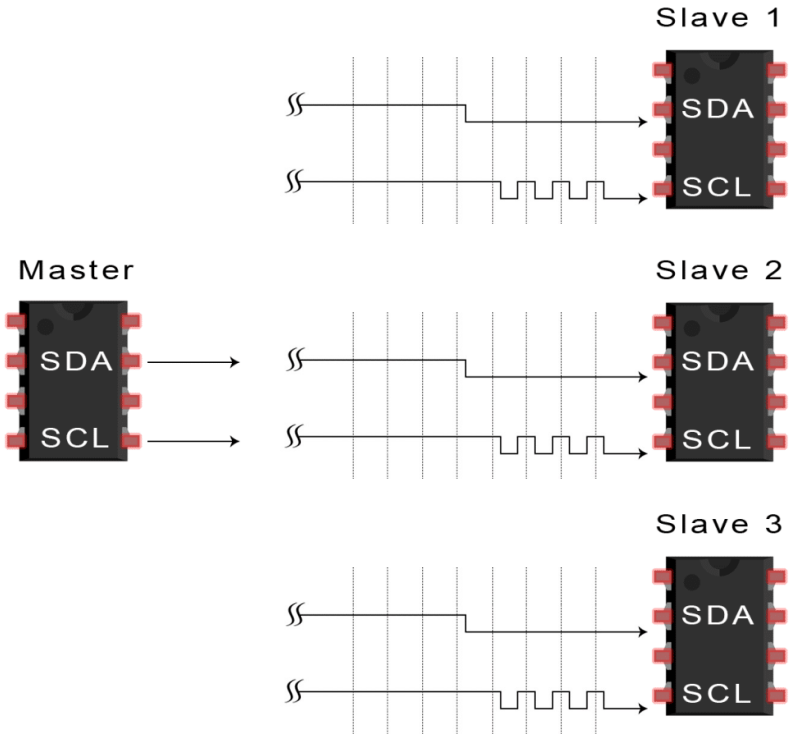
5



I2C Device Addressing

- To stop the data transmission, the master sends a stop condition to the slave by switching SCL high before switching SDA high:

6

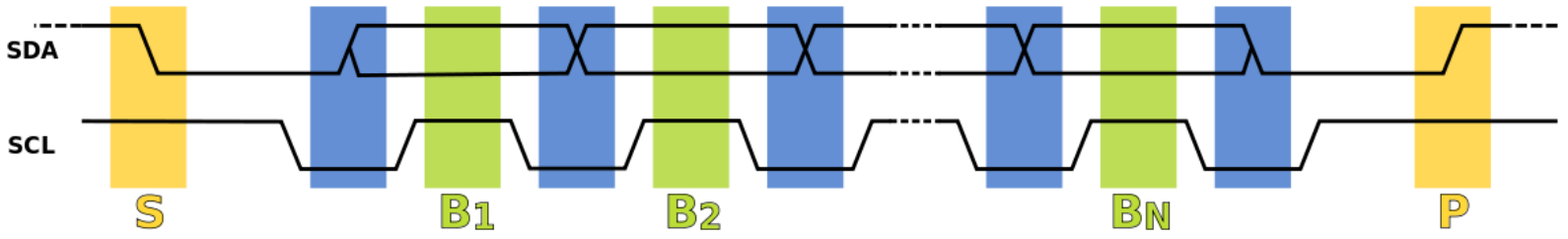


When I2C Slaves Look for Their Device Address?

- During the data transfer SDA changes while SCL is LOW
  - After Falling edge of SCL
  - Before Rising edge of SCL

- Data transfer is initiated with **SDA being pulled low while SCL stays high.**
  - This condition does not happen again during the data transfer!
  - After this condition, devices look for the next 7 bits for address, if if address doesn't match they ignore the data transfer and look for stop bit!

- A *stop* bit (P) is signaled when **SDA is pulled high while SCL is high.**
  - This condition doesn't happen during the data transfer.



Power and Energy Comparison

- Comparing UART, SPI and I2C Energy and Power consumption:
- The communication protocol is implemented in both HW and SW
- The data rate is set at 15.8 Kbps.

Parameters	UART		SPI				I^2C			
			RX selected		RX unselected		RX selected		RX unselected	
	1 byte	9 bytes	1 byte	9 bytes	1 byte	9 bytes	1 byte	9 bytes	1 byte	9 bytes
Interface implementation in hardware										
Power consumption with inactive interface, mW	6.06	6.06	4.66	4.66	4.66	4.66	6.13	6.13	6.13	6.13
Communication:										
Required time, ms	0.68	5.78	0.52	4.6	0.52	4.6	1.26	5.91	1.26	5.91
Required energy, μJ	7.31	53.31	2.5	23.53	2.42	22.52	8.11	31.71	8.01	30.23
Normalized energy[a], μJ	7.27	53.01	2.48	23.33	2.4	22.32	8.19	31.99	8.08	30.5
Interface implementation in software										
Power consumption with inactive interface, mW	11.04	11.04	11.17	11.17	11.17	11.17	11.23	11.23	11.23	11.23
Communication:										
Required time, ms	0.64	5.72	0.51	4.62	0.51	4.61	1.23	5.85	1.23	5.86
Required energy, μJ	9.00	80.06	7.26	65.25	7.26	65.25	17.45	82.84	17.44	83.6
Normalized energy[a], μJ	9.01	80.16	7.28	65.42	7.28	65.42	17.45	82.84	17.44	83.6

Source: http://ieeexplore.ieee.org/document/6208716/

Implement Protocol in HW or SW?

Parameters	UART		SPI				I^2C			
			RX selected		RX unselected		RX selected		RX unselected	
	1 byte	9 bytes	1 byte	9 bytes	1 byte	9 bytes	1 byte	9 bytes	1 byte	9 bytes
Interface implementation in hardware										
Power consumption with inactive interface, mW	6.06	6.06	4.66	4.66	4.66	4.66	6.13	6.13	6.13	6.13
Communication:										
Required time, ms	0.68	5.78	0.52	4.6	0.52	4.6	1.26	5.91	1.26	5.91
Required energy, μJ	7.31	53.31	2.5	23.53	2.42	22.52	8.11	31.71	8.01	30.23
Normalized energy[a], μJ	7.27	53.01	2.48	23.33	2.4	22.32	8.19	31.99	8.08	30.5
Interface implementation in software										
Power consumption with inactive interface, mW	11.04	11.04	11.17	11.17	11.17	11.17	11.23	11.23	11.23	11.23
Communication:										
Required time, ms	0.64	5.72	0.51	4.62	0.51	4.61	1.23	5.85	1.23	5.86
Required energy, μJ	9.00	80.06	7.26	65.25	7.26	65.25	17.45	82.84	17.44	83.6
Normalized energy[a], μJ	9.01	80.16	7.28	65.42	7.28	65.42	17.45	82.84	17.44	83.6

- Software implementation consumes more power than hardware implementation!
  - SW implementation, should be executed with general purpose processing logic, hence more operations are needed!
  - Customized HW is always more efficient!
  - HW could use tricks, such as going to sleep mode, using interrupts, …
- **Conclusion:** If low power solutions is desired, look for Microprocessor that has the HW implementation for the desired communication protocol.

Source: http://ieeexplore.ieee.org/document/6208716/

Power and Energy Comparison

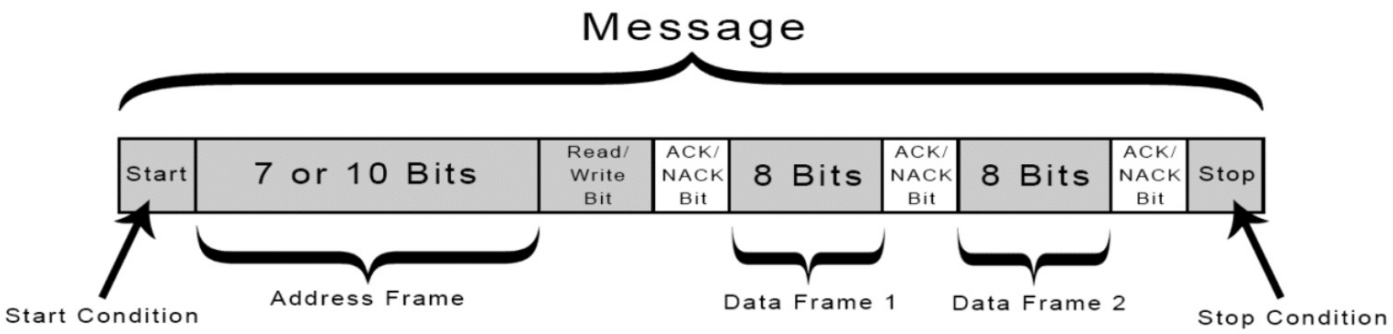
Parameters	UART		SPI				I^2C			
			RX selected		RX unselected		RX selected		RX unselected	
	1 byte	9 bytes	1 byte	9 bytes	1 byte	9 bytes	1 byte	9 bytes	1 byte	9 bytes
Interface implementation in hardware										
Power consumption with inactive interface, mW	6.06	6.06	4.66	4.66	4.66	4.66	6.13	6.13	6.13	6.13
Communication:										
Required time, ms	0.68	5.78	0.52	4.6	0.52	4.6	1.26	5.91	1.26	5.91
Required energy, μJ	7.31	53.31	2.5	23.53	2.42	22.52	8.11	31.71	8.01	30.23
Normalized energy[a], μJ	7.27	53.01	2.48	23.33	2.4	22.32	8.19	31.99	8.08	30.5
Interface implementation in software										
Power consumption with inactive interface, mW	11.04	11.04	11.17	11.17	11.17	11.17	11.23	11.23	11.23	11.23
Communication:										
Required time, ms	0.64	5.72	0.51	4.62	0.51	4.61	1.23	5.85	1.23	5.86
Required energy, μJ	9.00	80.06	7.26	65.25	7.26	65.25	17.45	82.84	17.44	83.6
Normalized energy[a], μJ	9.01	80.16	7.28	65.42	7.28	65.42	17.45	82.84	17.44	83.6

- Compared to UART, I2C is less efficient for smaller number of bytes, but become more efficient as number of bytes increases.
- **Reason**: I2C need to send an extra byte for device identification.
  - For 1-Byte packet overhead is 100% when a single byte is transferred
  - Overhead is a much smaller % when number of transfer bytes is large!
  - For 10 bytes and larger I2C is more efficient than UART
- **Conclusion:** For power efficiency know your sensor behavior (data packet size) to choose the best protocol.

Source: http://ieeexplore.ieee.org/document/6208716/

Question:

- Sensor will be sending 28 bytes of data to the microcontroller. The UART uses a data size of 8bit. How many clock cycles are needed for UART communication? How many clock cycles are needed for I2C communication? Which is more efficient? Repeat the same question if sensor need to send 7 bytes of data?



Power and Energy Comparison

Parameters	UART		SPI				I^2C			
			RX selected		RX unselected		RX selected		RX unselected	
	1 byte	9 bytes	1 byte	9 bytes	1 byte	9 bytes	1 byte	9 bytes	1 byte	9 bytes
Interface implementation in hardware										
Power consumption with inactive interface, mW	6.06	6.06	4.66	4.66	4.66	4.66	6.13	6.13	6.13	6.13
Communication:										
Required time, ms	0.68	5.78	0.52	4.6	0.52	4.6	1.26	5.91	1.26	5.91
Required energy, μJ	7.31	53.31	2.5	23.53	2.42	22.52	8.11	31.71	8.01	30.23
Normalized energy[a], μJ	7.27	53.01	2.48	23.33	2.4	22.32	8.19	31.99	8.08	30.5
Interface implementation in software										
Power consumption with inactive interface, mW	11.04	11.04	11.17	11.17	11.17	11.17	11.23	11.23	11.23	11.23
Communication:										
Required time, ms	0.64	5.72	0.51	4.62	0.51	4.61	1.23	5.85	1.23	5.86
Required energy, μJ	9.00	80.06	7.26	65.25	7.26	65.25	17.45	82.84	17.44	83.6
Normalized energy[a], μJ	9.01	80.16	7.28	65.42	7.28	65.42	17.45	82.84	17.44	83.6

- SPI is more energy efficient than I2C, but it come at a cost:
  - SPI doesn't support automatic device authentication (**Scalability**)
    - I2C identify the device by the first byte is transmits
    - SPI require wiring up the device correctly and SS (or alternatively using CS)
  - SPI doesn't support multiple Masters! (**functionality**)
  - SPI has larger wiring overhead.
- **Conclusion:** Consider the number of sensors and masters you want to connect to a device, need for adding additional sensors in the future, and the power consumption constraints to decide between I2C and SPI

Source: http://ieeexplore.ieee.org/document/6208716/