Serial Communication

Input/Output Systems

Processor needs to communicate with other devices:

- Receive signals from sensors
- Send commands to actuators
- Or both (e.g., disks, audio, video devices, other processors)

An Example: SICK Laser Range Finder

- Laser is scanned horizontally
- Using phase information, can infer the distance to the nearest obstacle
- Resolution: ~.5 degrees, 1 cm
- Can handle full 180 degrees at 20 Hz

Serial Communication

- Communicate a set of bytes using a single signal line
- We do this by sending one bit at a time:
  - The value of the first bit determines the state of a signal line for a specified period of time
  - Then, the value of the 2^{nd} bit is used
  - Etc.

Serial Experiment…

Serial Communication

The sender and receiver must have some way of agreeing on when a specific bit is being sent

- Some cases: the sender will also send a clock signal (on a separate line)
- Other cases: each side has a clock to tell it when to write/read a bit
  - The sender/receiver must first synchronize their clocks before transfer begins

Asynchronous Serial Communication

- The sender and receiver have their own clocks, which they do not share
- This reduces the number of signal lines

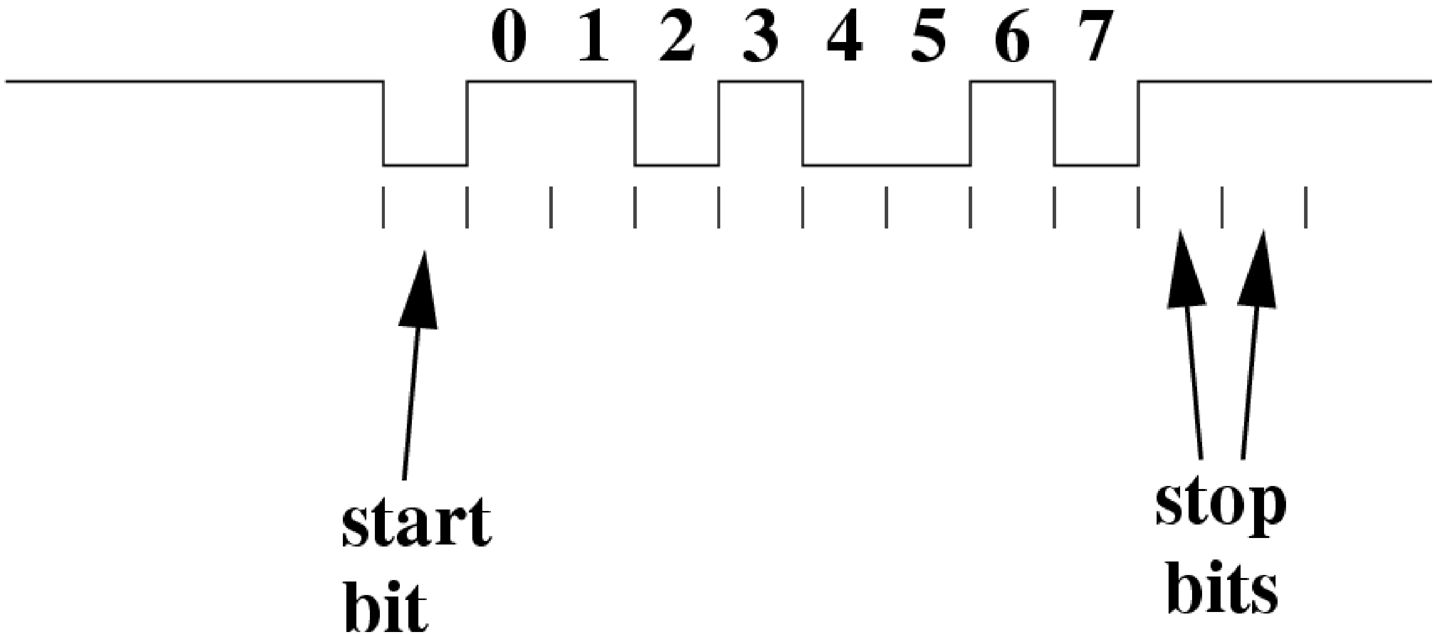
But: we still need some way to agree that data is valid. How?

Asynchronous Serial Communication

How can the two sides agree that the data is valid?

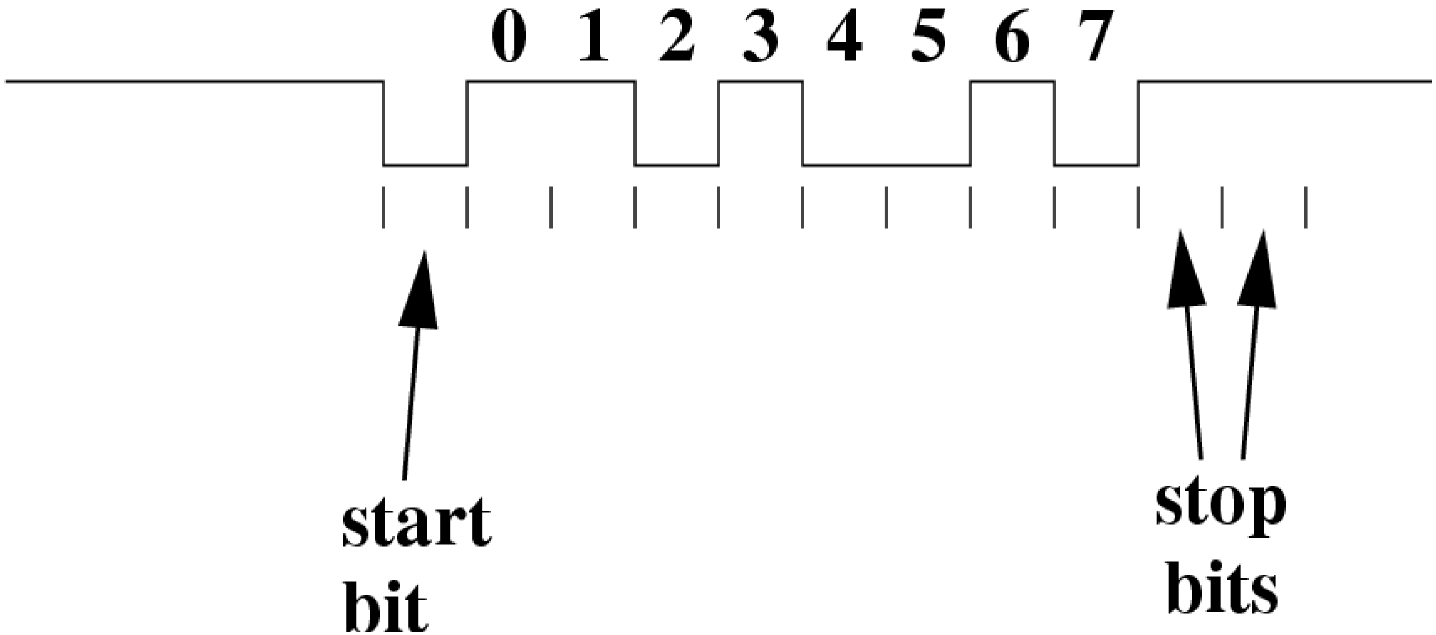
- Must both be operating at essentially the same transmit/receive frequency
- A data byte is prefaced with a bit of information that tells the receiver that bits are coming
- The receiver uses the arrival time of this **start bit** to synchronize its clock

A Typical Data Frame



The start bit indicates that a byte is coming

A Typical Data Frame



The stop bits allow the receiver to immediately check whether this is a valid frame

- If not, the byte is thrown away

Data Frame Handling

Most of the time, we do not deal with the data frame level. Instead, we rely on:

- Hardware solutions: Universal Asynchronous Receiver Transmitter (UART)
  - Very common in computing devices
- Software solutions in libraries

One (Old) Standard: RS232-C

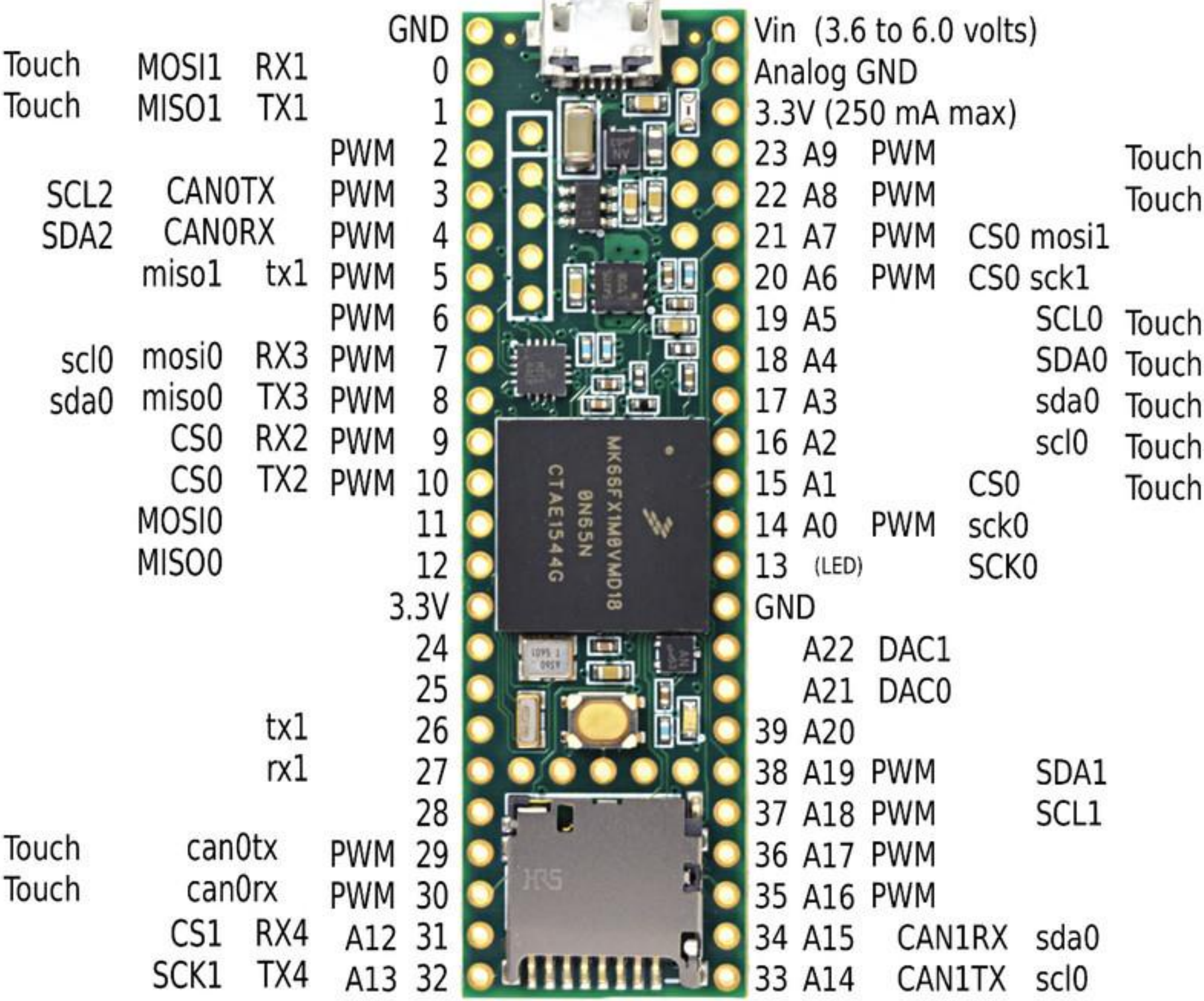
Defines a logic encoding standard:

- “High” is encoded with a voltage of -5 to -15 (-12 to -13V is typical)
- “Low” is encoded with a voltage of 5 to 15 (12 to 13V is typical)

RS232 on the Teensy 3.5

Our Teensy has 7 Universal, Asynchronous serial Receiver/Transmitters (UARTs):

- #0: USB; #1 … 6: RX/TX pins
- Each handles all of the bit-level manipulation
  - Software only worries about the byte level
- 1 … 6 use 0V and 3.3V to encode “lows” and “highs”
  - Must convert if talking to a true RS232C device (+/- 13V)


GND
Vin (3.6 to 6.0 volts)
Touch MOSI1 RX1 0
Analog GND
Touch MISO1 TX1 1
3.3V (250 mA max)
PWM 2
23 A9 PWM Touch
SCL2 CAN0TX PWM 3
22 A8 PWM Touch
SDA2 CAN0RX PWM 4
21 A7 PWM CS0 mosi1
miso1 tx1 PWM 5
20 A6 PWM CS0 sck1
PWM 6
19 A5 SCL0 Touch
scl0 mosi0 RX3 PWM 7
18 A4 SDA0 Touch
sda0 miso0 TX3 PWM 8
17 A3 sda0 Touch
CS0 RX2 PWM 9
16 A2 scl0 Touch
CS0 TX2 PWM 10
15 A1 CS0 Touch
MOSI0 11
14 A0 PWM sck0
MISO0 12
13 (LED) SCK0
3.3V
GND
24
A22 DAC1
25
A21 DAC0
tx1 26
39 A20
rx1 27
38 A19 PWM SDA1
28
37 A18 PWM SCL1
Touch can0tx PWM 29
36 A17 PWM
Touch can0rx PWM 30
35 A16 PWM
CS1 RX4 A12 31
34 A15 CAN1RX sda0
SCK1 TX4 A13 32
33 A14 CAN1TX scl0

Serial Initialization

Options include:

- Serial.begin(9600);
- SerialX.begin(9600);
  – Where X = 1 … 6

Generating Serial Output

```
int val = 42;
float f = 6.282;

Serial.println("foo:");
Serial.println(val);
Serial.printf("foo: %d (%f)\n", val, f);
```

Reading Serial Input

- Serial.read() will return the next character in the buffer
- If the buffer is empty, then this function will ***block*** until a character is available to be read
- This can be very dangerous in a real-time domain

Checking for Characters

What we would like to do is to ask ahead of time as to whether a character is ready to be read …

Checking for Characters

What we would like to do is to ask ahead of time as to whether a character is ready to be read …

```
loop(){
     if(Serial.available()) {
          char c = Serial.read();
          <do something with the read char>
     }
     <do something else while waiting>
}
```

Character Representation

- A “char” is just an 8-bit number
- This allows us to perform meaningful mathematical operations on the characters

Character Representation: ASCII

Binary	Dec	Hex	Glyph	Binary	Dec	Hex	Glyph	Binary	Dec	Hex	Glyph
010 0000	32	20	SP	100 0000	64	40	@	110 0000	96	60	`
010 0001	33	21	!	100 0001	65	41	A	110 0001	97	61	a
010 0010	34	22	"	100 0010	66	42	B	110 0010	98	62	b
010 0011	35	23	#	100 0011	67	43	C	110 0011	99	63	c
010 0100	36	24	$	100 0100	68	44	D	110 0100	100	64	d
010 0101	37	25	%	100 0101	69	45	E	110 0101	101	65	e
010 0110	38	26	&	100 0110	70	46	F	110 0110	102	66	f
010 0111	39	27	'	100 0111	71	47	G	110 0111	103	67	g
010 1000	40	28	(	100 1000	72	48	H	110 1000	104	68	h
010 1001	41	29	)	100 1001	73	49	I	110 1001	105	69	i
010 1010	42	2A	*	100 1010	74	4A	J	110 1010	106	6A	j
010 1011	43	2B	+	100 1011	75	4B	K	110 1011	107	6B	k
010 1100	44	2C	,	100 1100	76	4C	L	110 1100	108	6C	l
010 1101	45	2D	-	100 1101	77	4D	M	110 1101	109	6D	m
010 1110	46	2E	.	100 1110	78	4E	N	110 1110	110	6E	n
010 1111	47	2F	/	100 1111	79	4F	O	110 1111	111	6F	o
011 0000	48	30	0	101 0000	80	50	P	111 0000	112	70	p
011 0001	49	31	1	101 0001	81	51	Q	111 0001	113	71	q
011 0010	50	32	2	101 0010	82	52	R	111 0010	114	72	r
011 0011	51	33	3	101 0011	83	53	S	111 0011	115	73	s
011 0100	52	34	4	101 0100	84	54	T	111 0100	116	74	t
011 0101	53	35	5	101 0101	85	55	U	111 0101	117	75	u
011 0110	54	36	6	101 0110	86	56	V	111 0110	118	76	v
011 0111	55	37	7	101 0111	87	57	W	111 0111	119	77	w
011 1000	56	38	8	101 1000	88	58	X	111 1000	120	78	x
011 1001	57	39	9	101 1001	89	59	Y	111 1001	121	79	y
011 1010	58	3A	:	101 1010	90	5A	Z	111 1010	122	7A	z
011 1011	59	3B	;	101 1011	91	5B	[	111 1011	123	7B	{
011 1100	60	3C	<	101 1100	92	5C	\	111 1100	124	7C	|
011 1101	61	3D	=	101 1101	93	5D	]	111 1101	125	7D	}
011 1110	62	3E	>	101 1110	94	5E	^	111 1110	126	7E	~
011 1111	63	3F	?	101 1111	95	5F	_				

Serial Challenge

- Suppose that we know that we will be receiving a sequence of 3 decimal digits from the serial port
- How do we translate these digits into an integer representation?

Serial Challenge II

- Suppose that we know that we will be receiving a sequence of k decimal digits from the serial port
- How do we translate these digits into an integer representation?

- Can assume that the digits will fit within a uint16_t

Character Representation: ASCII

Binary	Dec	Hex	Glyph	Binary	Dec	Hex	Glyph	Binary	Dec	Hex	Glyph
010 0000	32	20	SP	100 0000	64	40	@	110 0000	96	60	`
010 0001	33	21	!	100 0001	65	41	A	110 0001	97	61	a
010 0010	34	22	"	100 0010	66	42	B	110 0010	98	62	b
010 0011	35	23	#	100 0011	67	43	C	110 0011	99	63	c
010 0100	36	24	$	100 0100	68	44	D	110 0100	100	64	d
010 0101	37	25	%	100 0101	69	45	E	110 0101	101	65	e
010 0110	38	26	&	100 0110	70	46	F	110 0110	102	66	f
010 0111	39	27	'	100 0111	71	47	G	110 0111	103	67	g
010 1000	40	28	(	100 1000	72	48	H	110 1000	104	68	h
010 1001	41	29	)	100 1001	73	49	I	110 1001	105	69	i
010 1010	42	2A	*	100 1010	74	4A	J	110 1010	106	6A	j
010 1011	43	2B	+	100 1011	75	4B	K	110 1011	107	6B	k
010 1100	44	2C	,	100 1100	76	4C	L	110 1100	108	6C	l
010 1101	45	2D	-	100 1101	77	4D	M	110 1101	109	6D	m
010 1110	46	2E	.	100 1110	78	4E	N	110 1110	110	6E	n
010 1111	47	2F	/	100 1111	79	4F	O	110 1111	111	6F	o
011 0000	48	30	0	101 0000	80	50	P	111 0000	112	70	p
011 0001	49	31	1	101 0001	81	51	Q	111 0001	113	71	q
011 0010	50	32	2	101 0010	82	52	R	111 0010	114	72	r
011 0011	51	33	3	101 0011	83	53	S	111 0011	115	73	s
011 0100	52	34	4	101 0100	84	54	T	111 0100	116	74	t
011 0101	53	35	5	101 0101	85	55	U	111 0101	117	75	u
011 0110	54	36	6	101 0110	86	56	V	111 0110	118	76	v
011 0111	55	37	7	101 0111	87	57	W	111 0111	119	77	w
011 1000	56	38	8	101 1000	88	58	X	111 1000	120	78	x
011 1001	57	39	9	101 1001	89	59	Y	111 1001	121	79	y
011 1010	58	3A	:	101 1010	90	5A	Z	111 1010	122	7A	z
011 1011	59	3B	;	101 1011	91	5B	[	111 1011	123	7B	{
011 1100	60	3C	<	101 1100	92	5C	\	111 1100	124	7C	|
011 1101	61	3D	=	101 1101	93	5D	]	111 1101	125	7D	}
011 1110	62	3E	>	101 1110	94	5E	^	111 1110	126	7E	~
011 1111	63	3F	?	101 1111	95	5F	_				

Synchronous Serial Communication

Synchronous Serial Communication

- A clock signal is also provided
- This allows for very fast communication
- Client/server model of communication: one side (the client) is in control of when/what are communicated
  - Client initiates any data transfer and provides the clock signal
  - Also referred to as a “master/slave” model

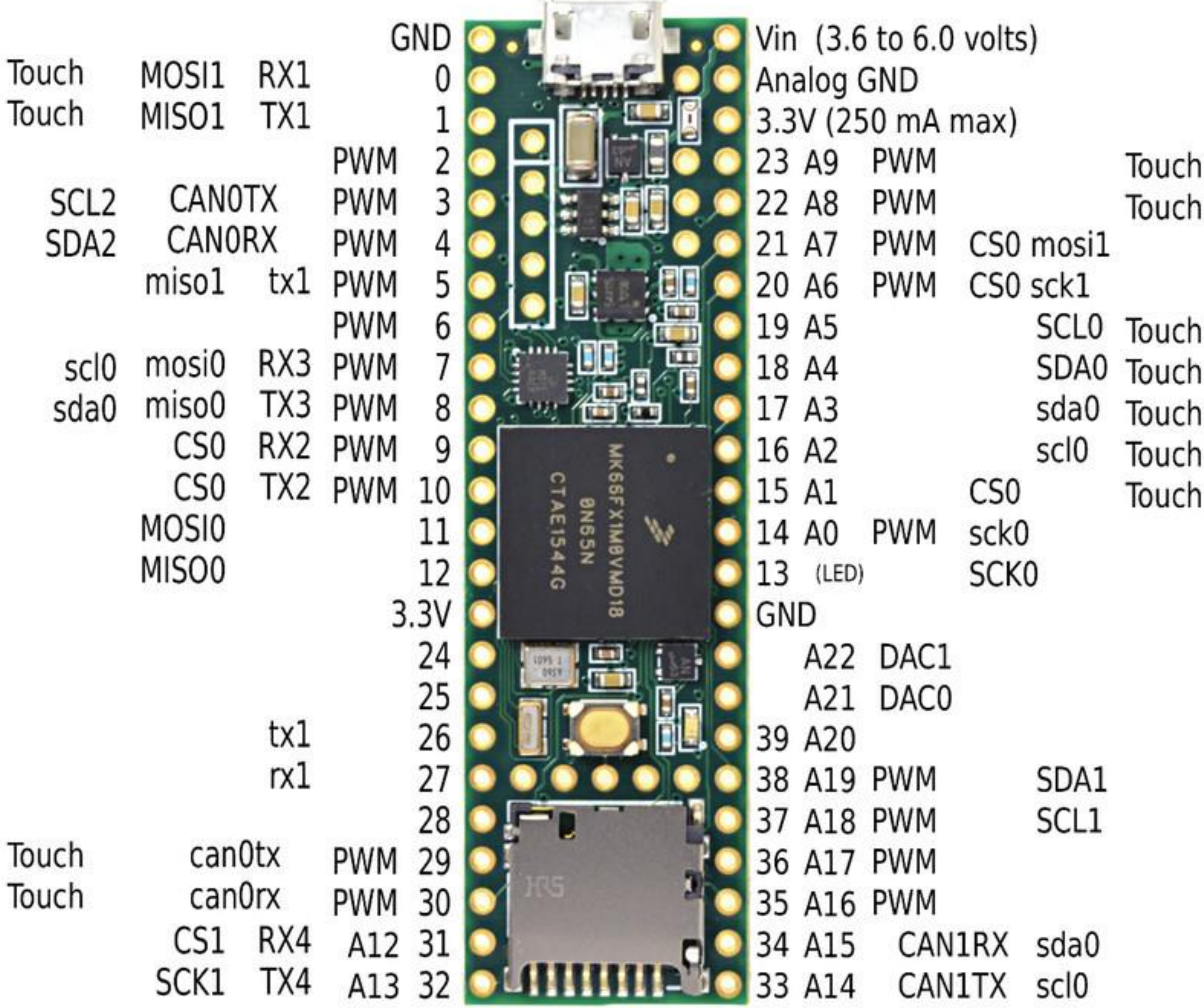
Serial Peripheral Interface (SPI)

Signal lines:

- SCK: serial clock
- MOSI: master-out-slave-in: communication of data from client to server
- MISO: master-in-slave-out: server to client
- CS: chip select: client brings this line low before data are exchanged

Serial Peripheral Interface (SPI)

- Servers can only transmit/receive data when CS is low
- Data exchange happens simultaneously
- Only one client in the circuit
- Servers can be daisy-chained into a single circuit
- Teensy has hardware support for the


GND
Vin (3.6 to 6.0 volts)
Touch MOSI1 RX1 0
Analog GND
Touch MISO1 TX1 1
3.3V (250 mA max)
PWM 2
23 A9 PWM Touch
SCL2 CAN0TX PWM 3
22 A8 PWM Touch
SDA2 CAN0RX PWM 4
21 A7 PWM CS0 mosi1
miso1 tx1 PWM 5
20 A6 PWM CS0 sck1
PWM 6
19 A5 SCL0 Touch
scl0 mosi0 RX3 PWM 7
18 A4 SDA0 Touch
sda0 miso0 TX3 PWM 8
17 A3 sda0 Touch
CS0 RX2 PWM 9
16 A2 scl0 Touch
CS0 TX2 PWM 10
15 A1 CS0 Touch
MOSI0 11
14 A0 PWM sck0
MISO0 12
13 (LED) SCK0
3.3V
GND
24
A22 DAC1
25
A21 DAC0
tx1 26
39 A20
rx1 27
38 A19 PWM SDA1
28
37 A18 PWM SCL1
Touch can0tx PWM 29
36 A17 PWM
Touch can0rx PWM 30
35 A16 PWM
CS1 RX4 A12 31
34 A15 CAN1RX sda0
SCK1 TX4 A13 32
33 A14 CAN1TX scl0

Inter Integrated Circuit (I2C)

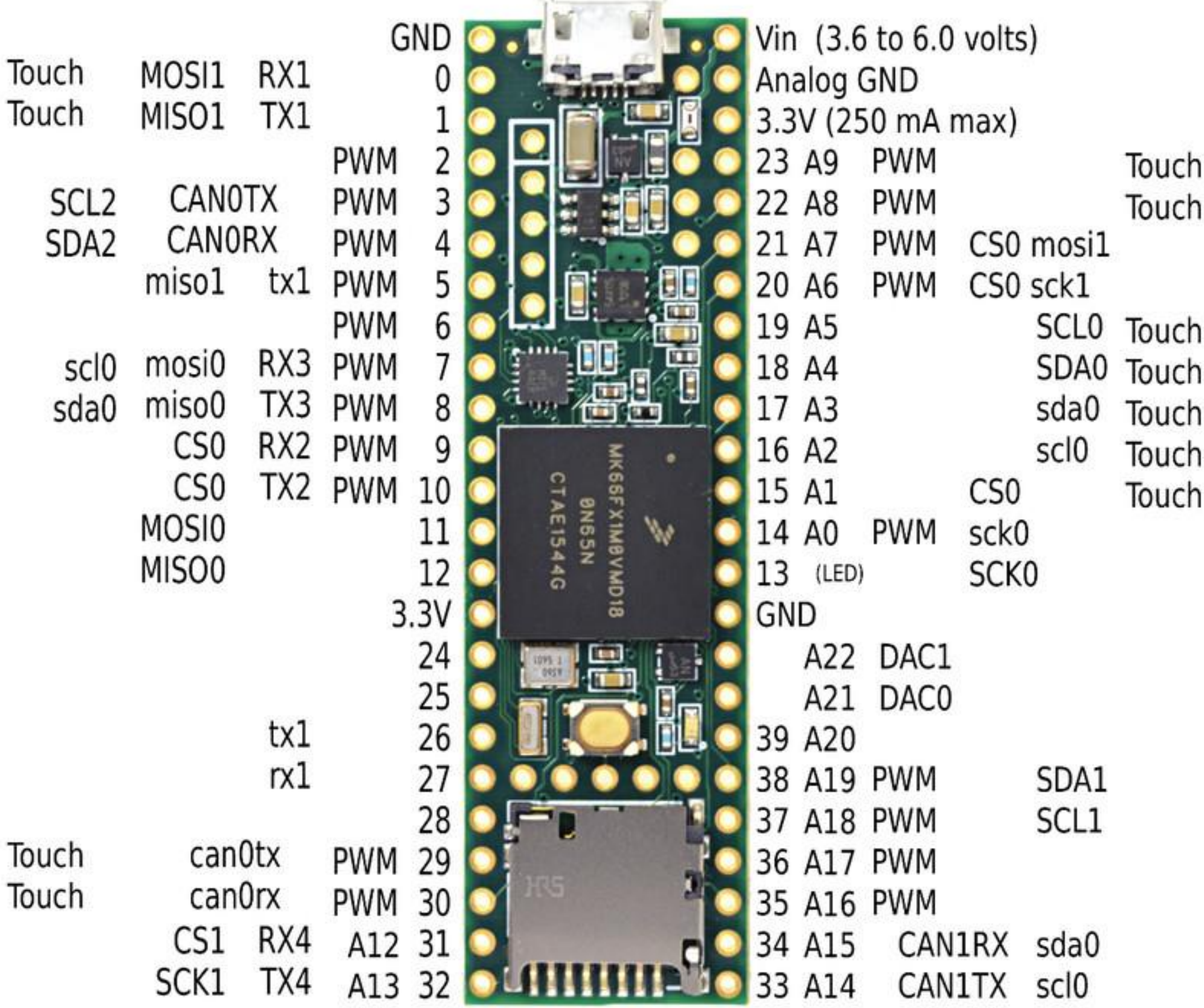
Signals:

- SCL: clock signal
- SDA: data signal

Servers have unique addresses (ID numbers) that are used by the client to initiate the conversation

Inter Integrated Circuit (I2C)

- Both the client and the server write to the data bus:
  - First the client writes data
  - Followed by the server writing data
- Multiple clients can exist
- The client always provides the clock signal
- Support for I2C in hardware on the Teensy


GND
Vin (3.6 to 6.0 volts)
Touch MOSI1 RX1 0
Analog GND
Touch MISO1 TX1 1
3.3V (250 mA max)
PWM 2
23 A9 PWM Touch
SCL2 CAN0TX PWM 3
22 A8 PWM Touch
SDA2 CAN0RX PWM 4
21 A7 PWM CS0 mosi1
miso1 tx1 PWM 5
20 A6 PWM CS0 sck1
PWM 6
19 A5 SCL0 Touch
scl0 mosi0 RX3 PWM 7
18 A4 SDA0 Touch
sda0 miso0 TX3 PWM 8
17 A3 sda0 Touch
CS0 RX2 PWM 9
16 A2 scl0 Touch
CS0 TX2 PWM 10
15 A1 CS0 Touch
MOSI0 11
14 A0 PWM sck0
MISO0 12
13 (LED) SCK0
3.3V
GND
24
A22 DAC1
25
A21 DAC0
tx1 26
39 A20
rx1 27
38 A19 PWM SDA1
28
37 A18 PWM SCL1
Touch can0tx PWM 29
36 A17 PWM
Touch can0rx PWM 30
35 A16 PWM
CS1 RX4 A12 31
34 A15 CAN1RX sda0
SCK1 TX4 A13 32
33 A14 CAN1TX scl0

Controller Area Network

- Communication across devices that are separated by some distance (10s of meters)
- Can function in electrically noisy environments
- Slow communication speeds (compared to I2C and SPI)
- Client/server model, but servers are not explicitly addressed. Instead, message types are addressed