TI-RSLK MAX

Texas Instruments Robotics System Learning Kit

Texas Instruments

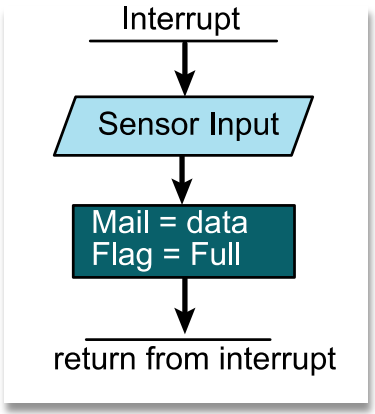
Module 21

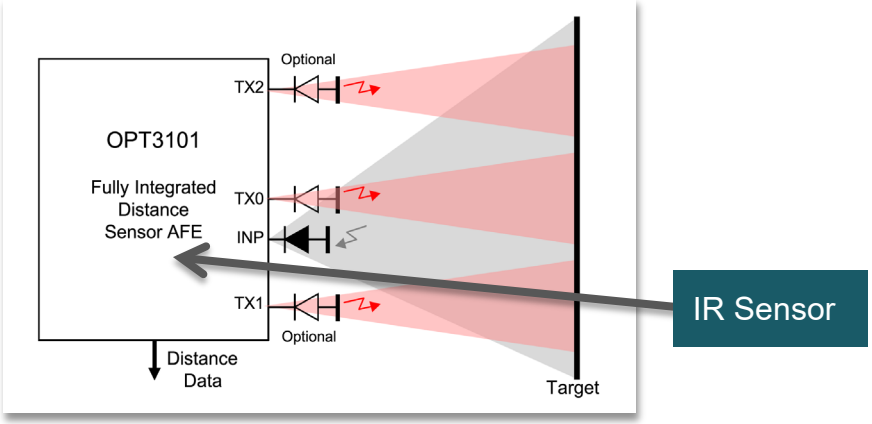
Lecture: Sensor Integration - Theory

Sensor Integration

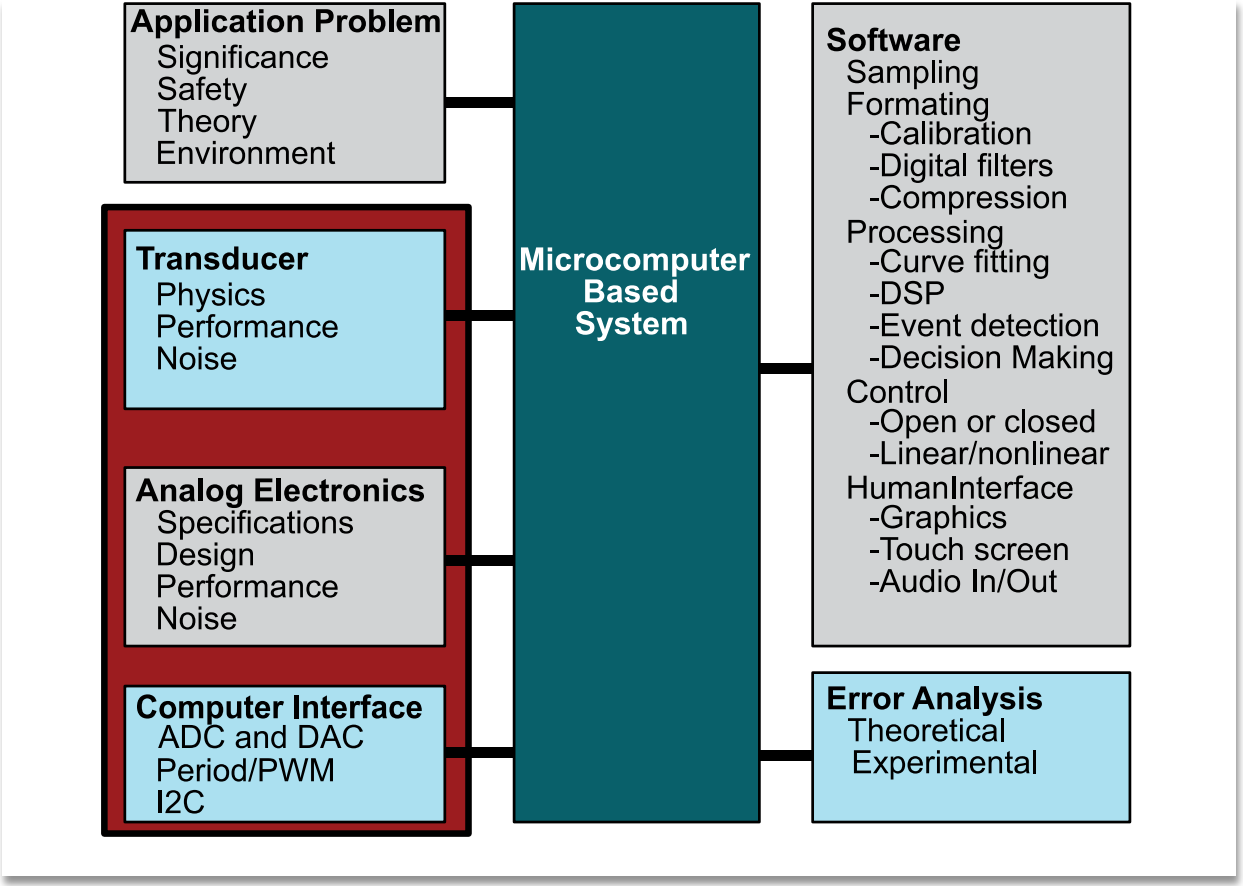
You will learn in this module

- Integrated Sensors
  - Physical to electrical conversion
  - Analog to digital conversion
  - Range, resolution, precision
- Sensor Interfacing
  - I2C protocol
  - Network of sensors
  - MSP432 software driver
  - Periodic sampling, Nyquist Theorem
- Digital Signal Processing
  - Digital filters, Central Limit Theorem
  - Spectrum analyzer
  - Sensor integration

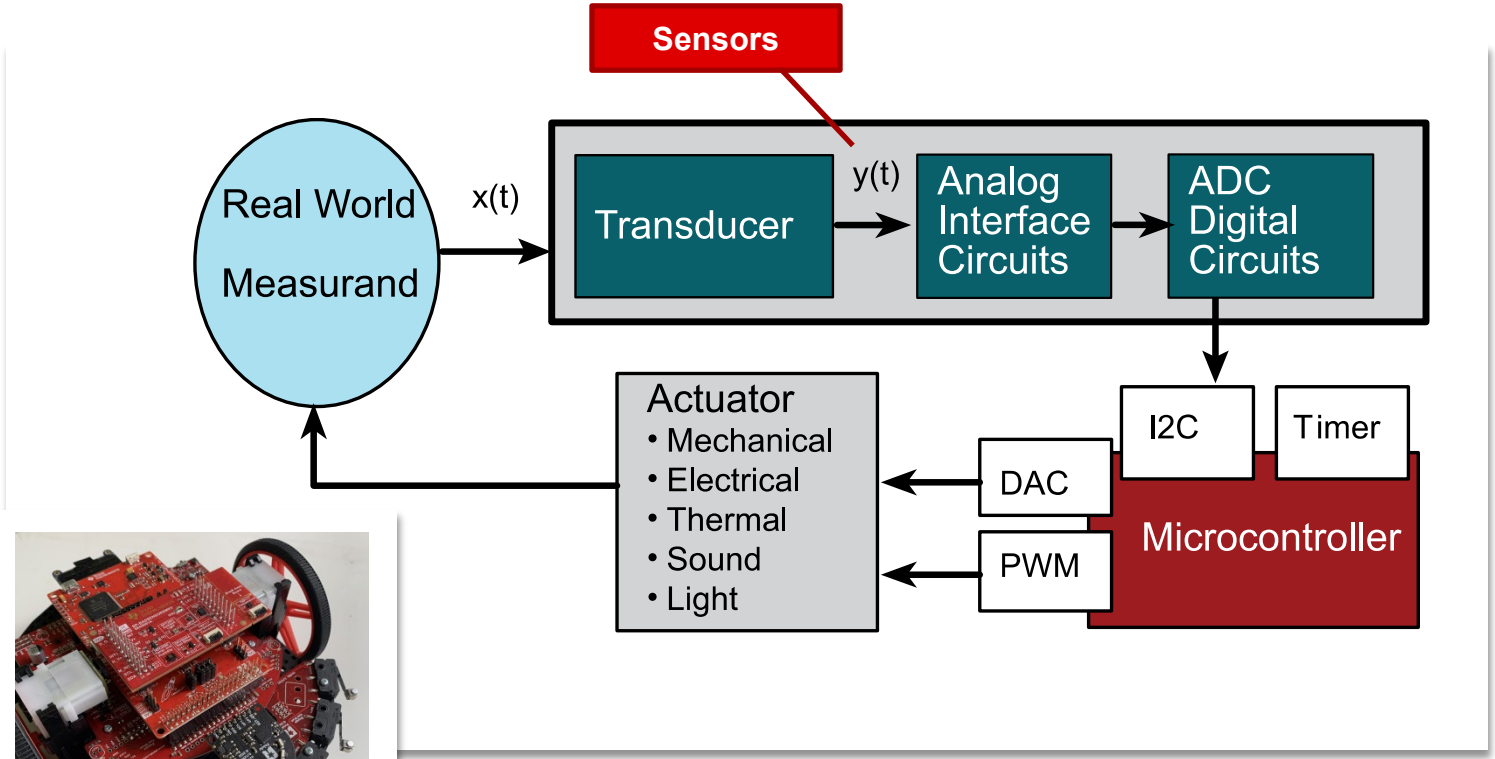




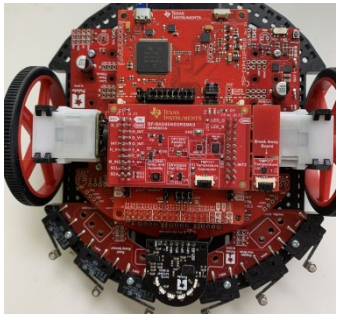
Data Acquisition Systems

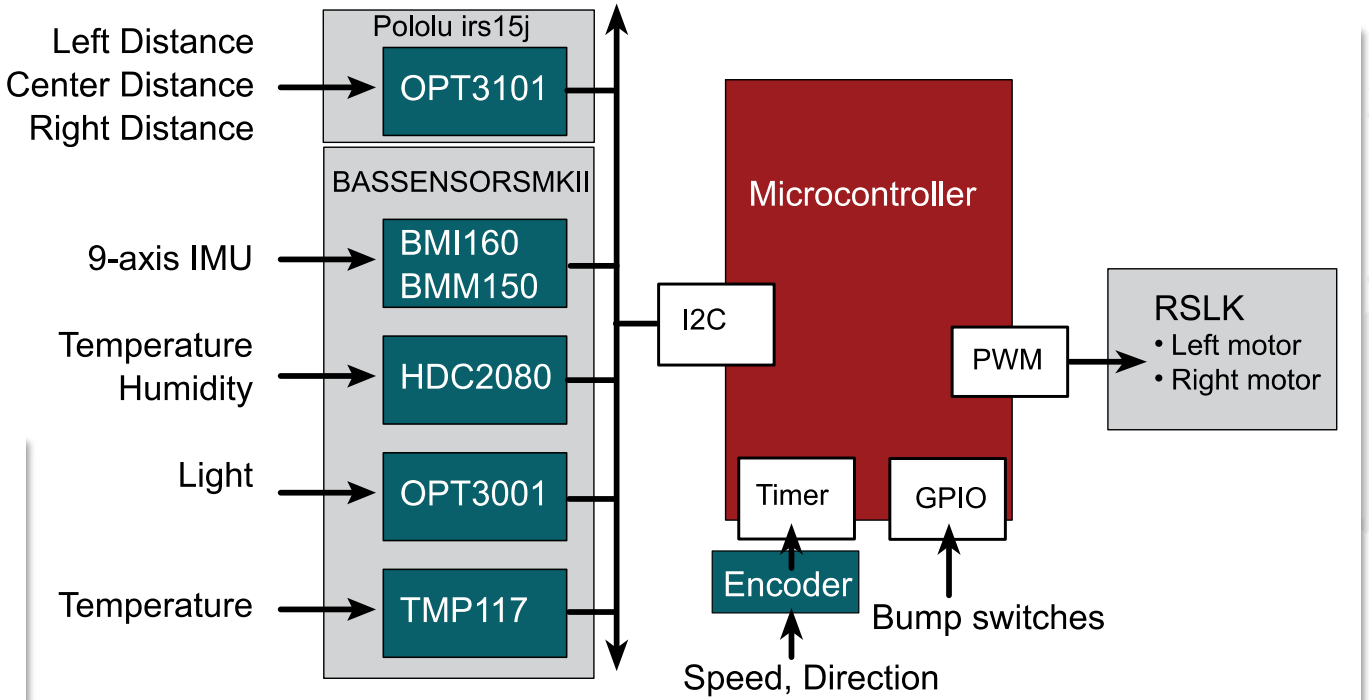


Sensors in an Automated Control System



TI–RSLK Max Control System





Sampling: conversion from physical to analog to digital

Sensor: physical to analog

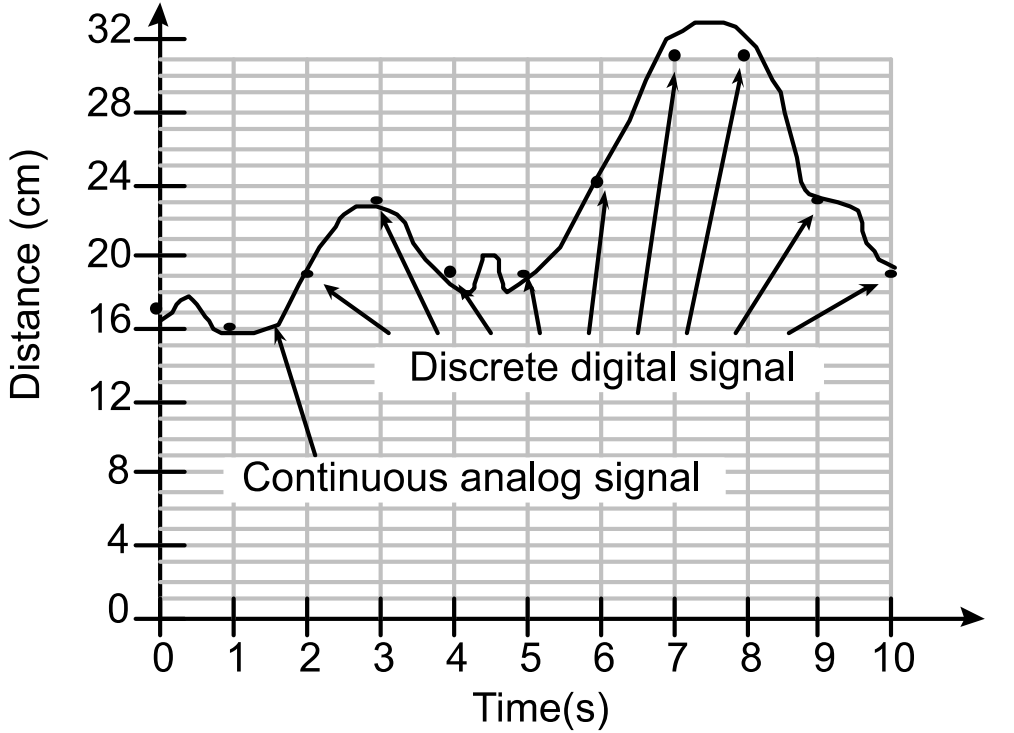
- Analog signal is voltage
- Analog signal is time

Amplitude

- Range
- Resolution
- Precision

Time domain

- Sampling rate, f_s
  - 0 to ½ f_s
- Number of samples
  - Buffer size N
- Frequency resolution
  - f_s/N



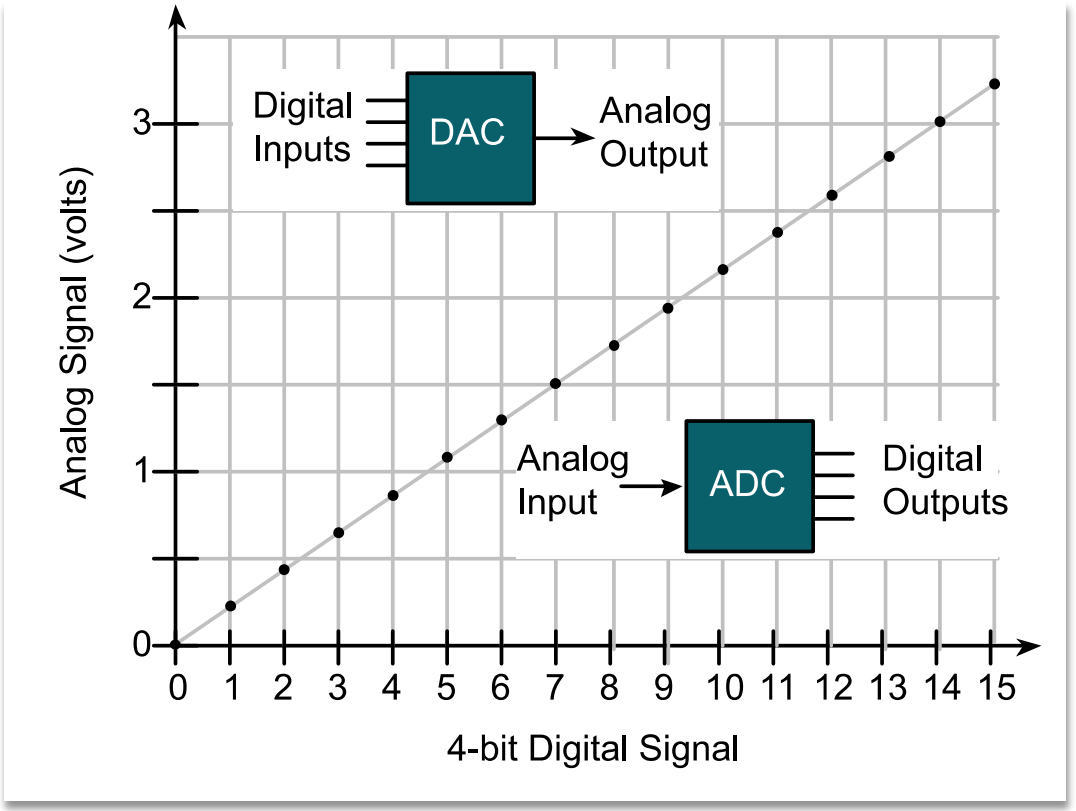
DAC versus ADC

DAC

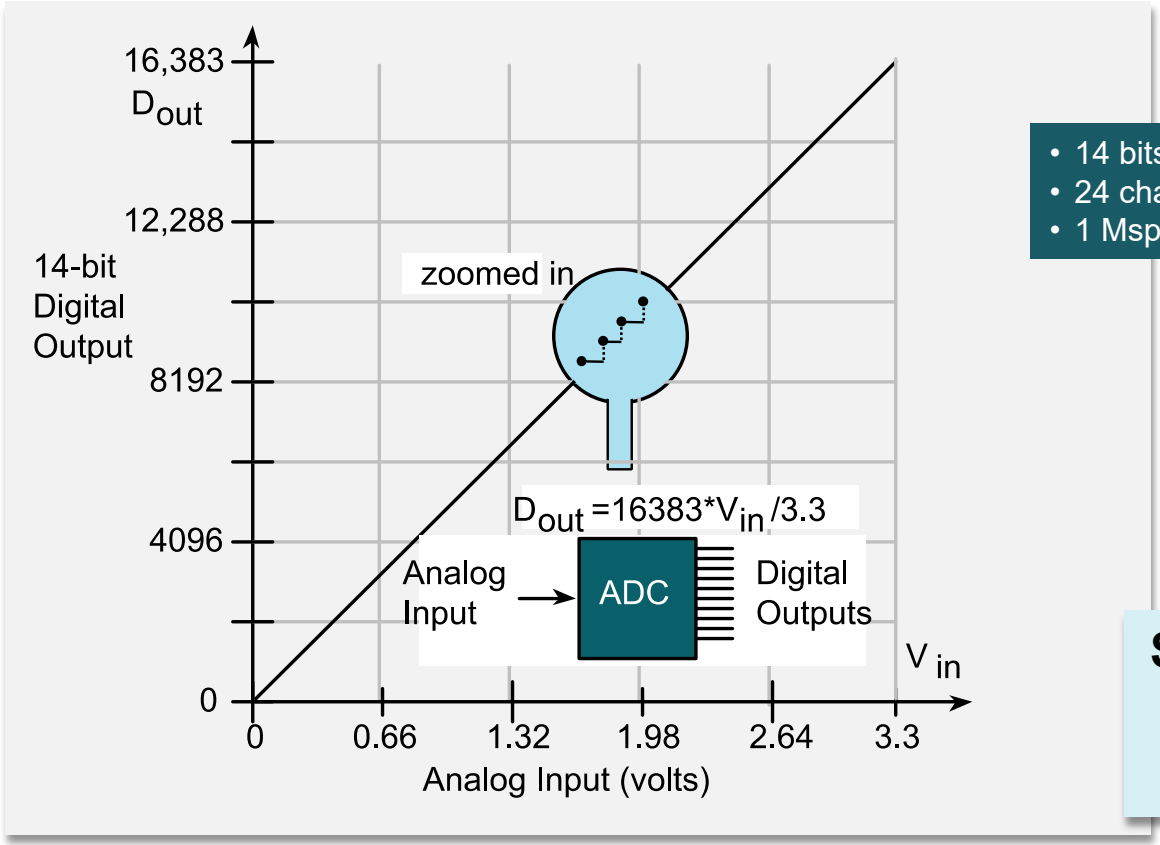
- Digital to Analog
- uC output
- Signal generation

ADC

- Analog to Digital
- uC input
- Measurements



MSP432 ADC14

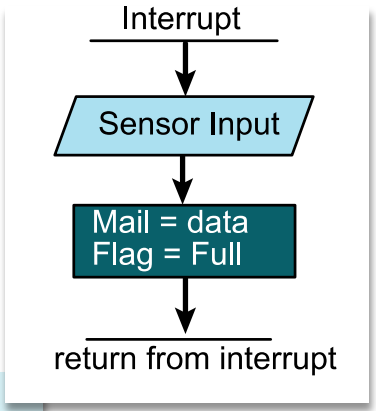


- 14 bits
- 24 channels
- 1 Msps

See Chapter 15 for more information

Periodic Interrupt and Mailbox

1. Read new measurement
2. Save in global
3. Set semaphore




```
// OPT3101 distance sensor started periodically every 33ms
// Interrupt triggered when measurement complete
// *PTxChan set to 0,1,2 when measurement done
void PORT6_IRQHandler(void){
  *PTxChan = OPT3101_GetMeasurement(Pdistances,Pamplitudes);
  P6->IFG = 0x00;  // acknowledge interrupt, clear all flags
}
```

390us

- 3 channels (Left, Center, Right)
- 10 Hz effective sampling rate
- Software overhead 390us/33ms = 1.1%

Summary

Digital Sampling

- Range
- Resolution
- Precision
- Accuracy
- Sampling rate, f_s
  - 0 to ½ f_s
- Number of samples
  - Buffer size N
- Frequency resolution
  - f_s/N

$$\frac{100}{n}\sum_{i=0}^{n}\frac{\left|x_{ti}-x_{mi}\right|}{x_{t\max}}$$

Software

- Initialization
- Sampling occurs in ISR
- Mailbox

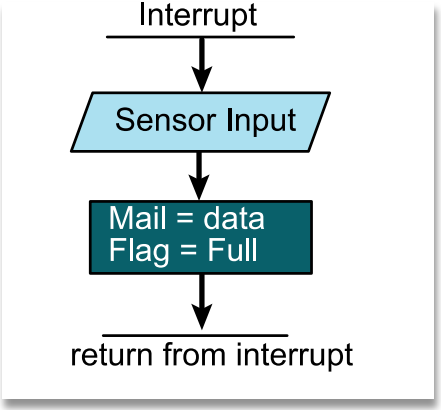
Module 21

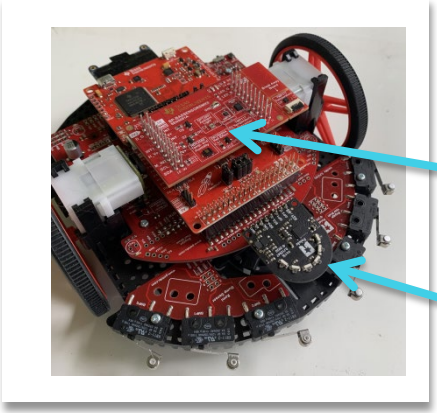
Lecture: Sensor Integration – Performance Measurements

Sensor Integration

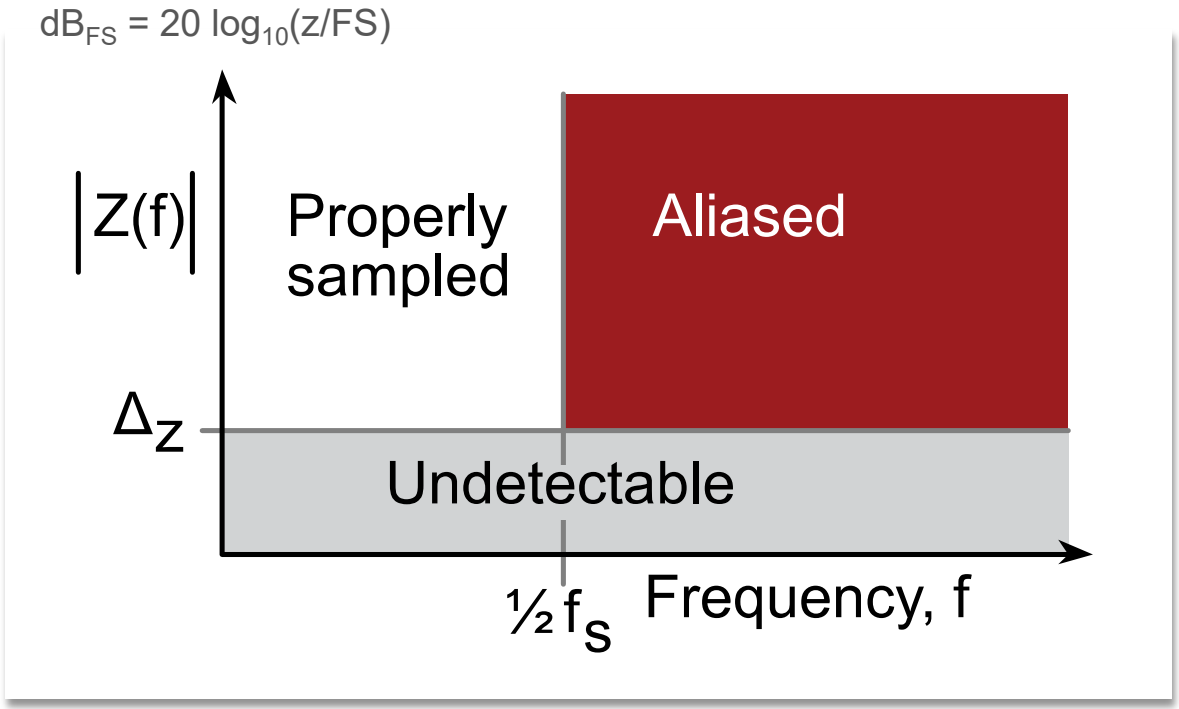
You will learn in this module

- Analog to Digital Converter
  - Sampling, Nyquist Theorem
  - Digital filtering
- Noise and statistics
  - Probability Mass Function
  - Spectrum Analyzer
  - Central Limit Theorem
- Data Acquisition Systems
  - Range, resolution, precision
  - Calibration
  - Accuracy





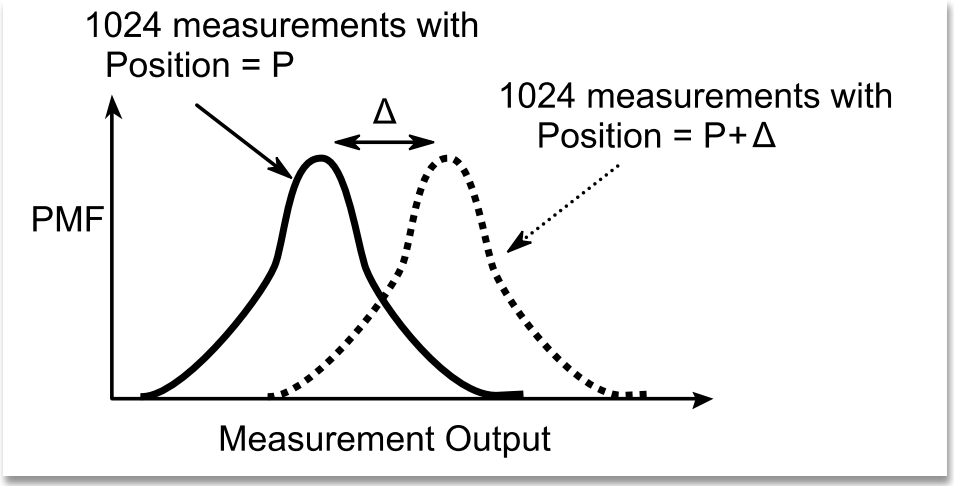
Nyquist Theorem



The **Nyquist Theorem** states that if the signal is sampled with a frequency of f_s, then the digital samples only contain frequency components from 0 to ½ f_s.

Statistics to Analyze Noise

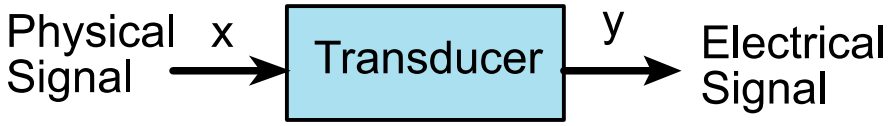
- Probability Mass Function (PMF)
- Average (μ = mean)
- Standard deviation (σ = sigma)
- Range (max-min)
- Coefficient of variation, CV = σ/μ
- ENOB=Precision $\log_2(\mu/\sigma)$ in bits
- Resolution, $\Delta \approx \sigma$
- Signal to noise ratio, SNR = μ/σ

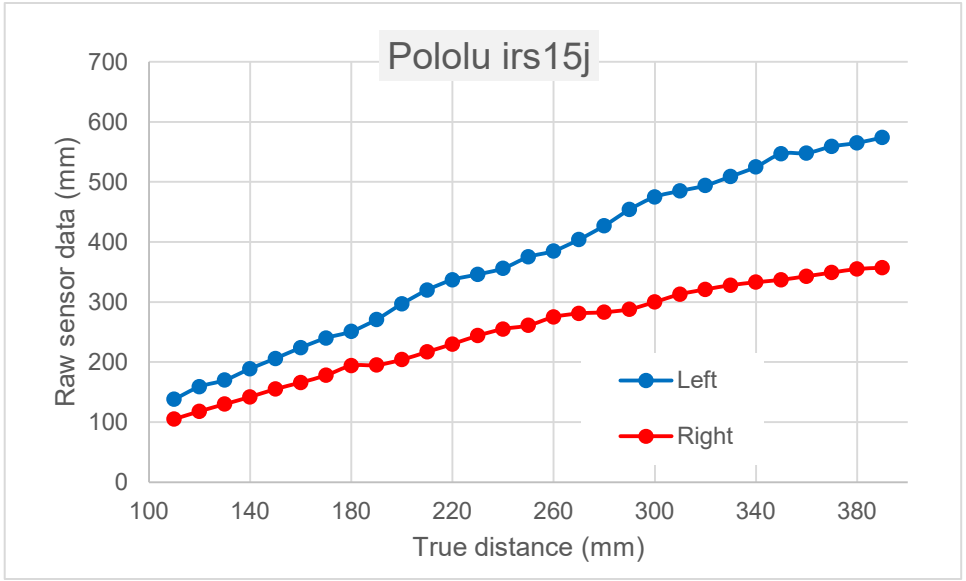


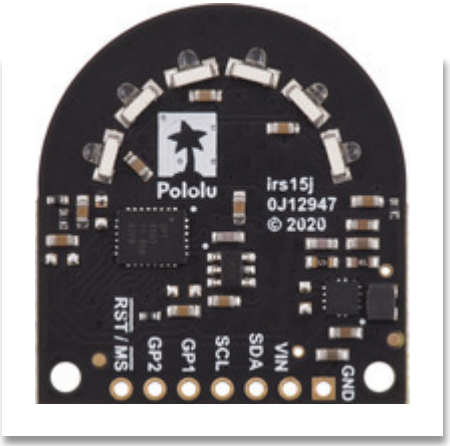
$$\mu \Rightarrow \overline{X} = \frac{1}{N}\sum_i X_i$$

$$\sigma^2 \Rightarrow S^2 = \frac{1}{N-1}\sum_i (X_i - \overline{X})^2$$

OPT3101 Transducer





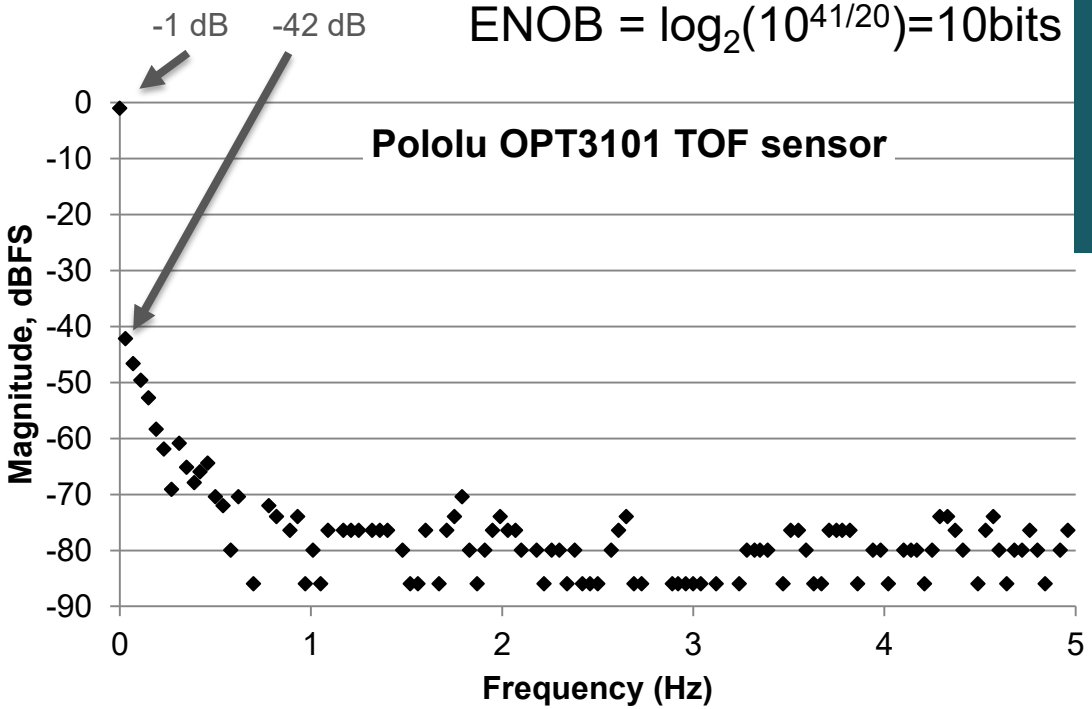


- Left is linear
- Right is quadratic
- Needs calibration

OPT3101 distance sensors are noisy

SNR = 41 dB

ENOB = $\log_2(10^{41/20})$=10bits



- Generation/recombination
- Periodic
- White
- Flicker, 1/f
- EM field pickup

Digital LPF

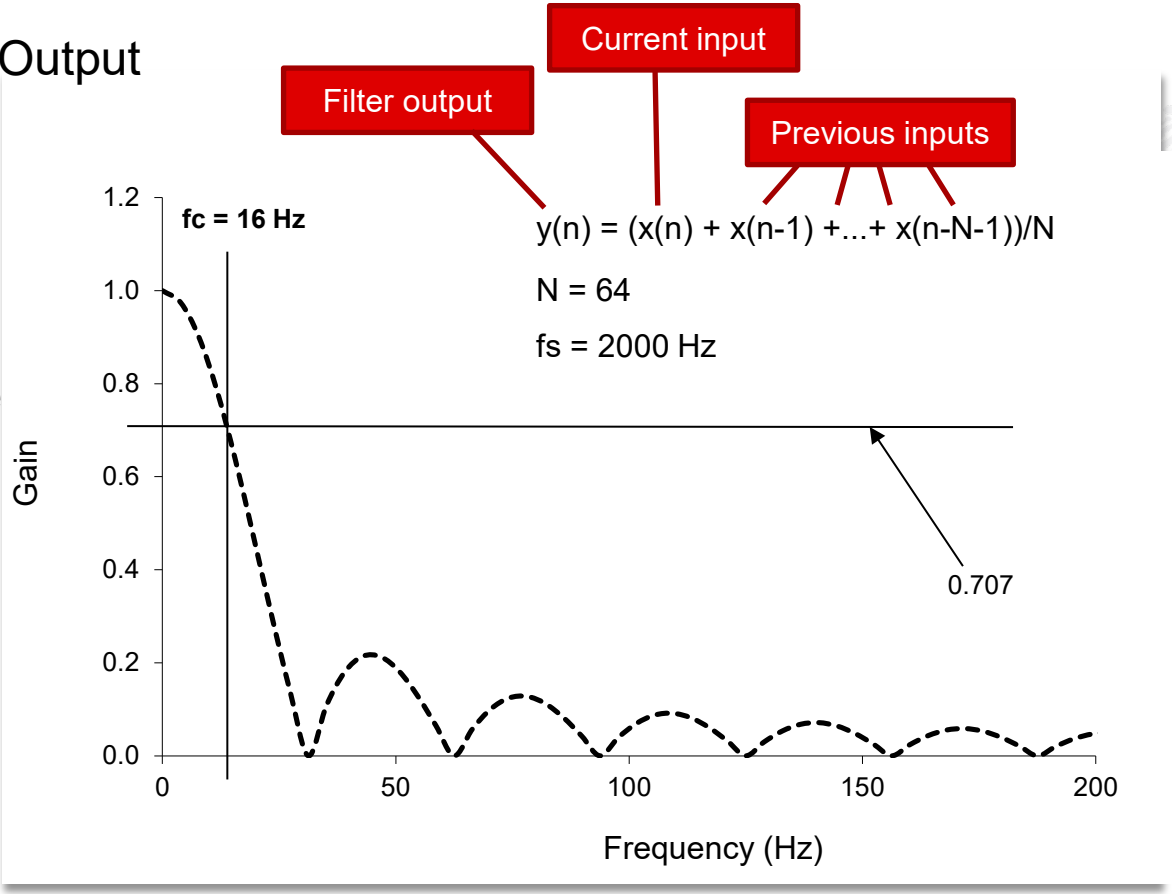
$dB_{FS} = 20 \log_{10}(d/200mm)$

Averaging Digital Low Pass Filter: Frequency Response

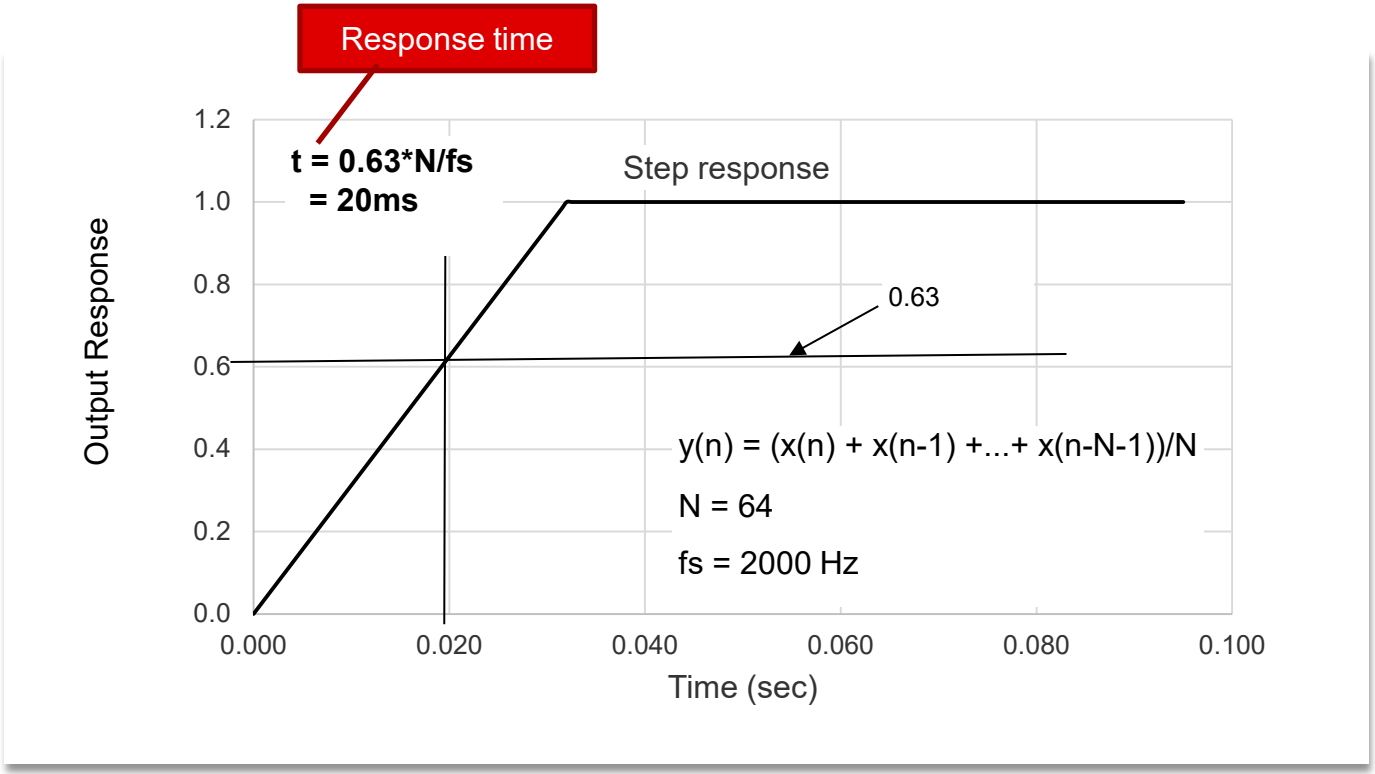
Input ➡ Filter ➡ Output

Linear Filter

- fs is sampling rate
- N is a constant
- Finite Impulse Response
- Low pass
- Group delay

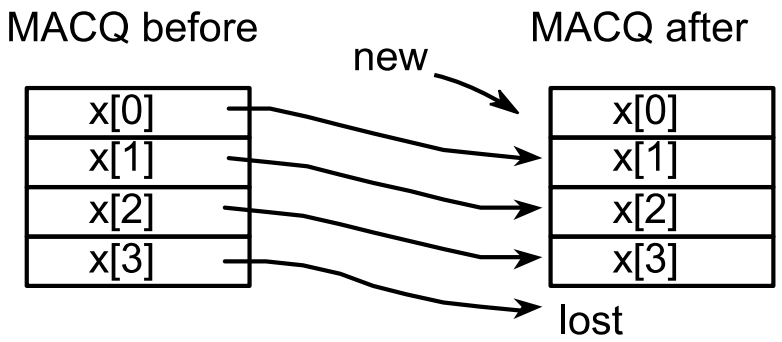


Averaging Digital Low Pass Filter: Group Delay



Digital Filtering (Implementation)

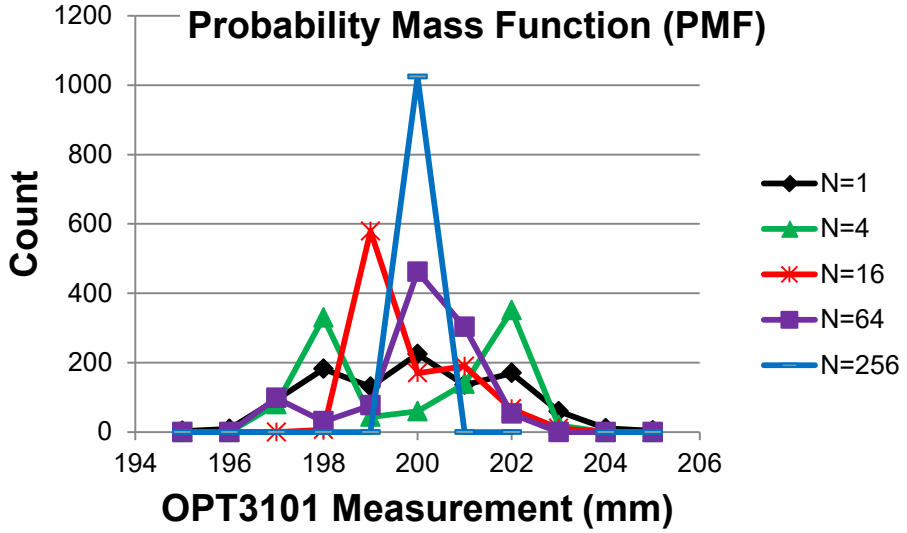
$$y(n) = (x(n)+x(n-1)+x(n-2)+x(n-3)/4$$

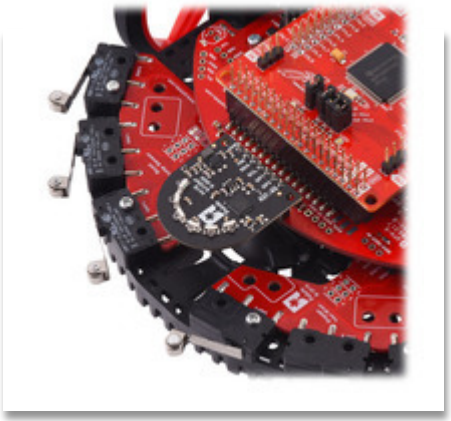



```
x[3] = x[2];
x[2] = x[1];
x[1] = x[0];
x[0] = Measurement();
y = (x[0]+x[1]+x[2]+x[3])/4;
```

See LPF.c

Probability Mass Function (PMF)

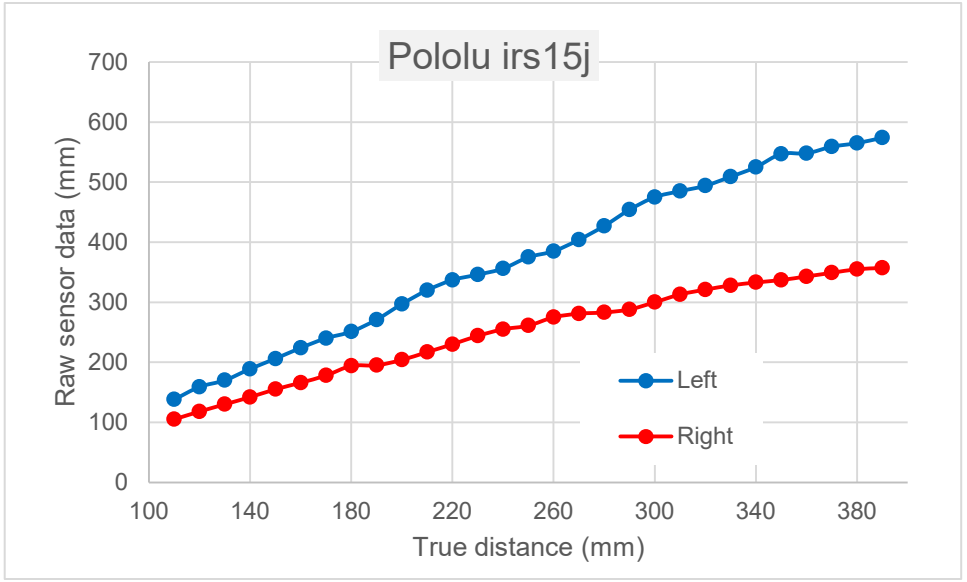




CLT states that as independent random variables are added, their sum tends toward a Normal distribution.

- Constant input distance 200 mm
- Average of last n samples
- f_s = 10 Hz

Distance to wall



Calibration

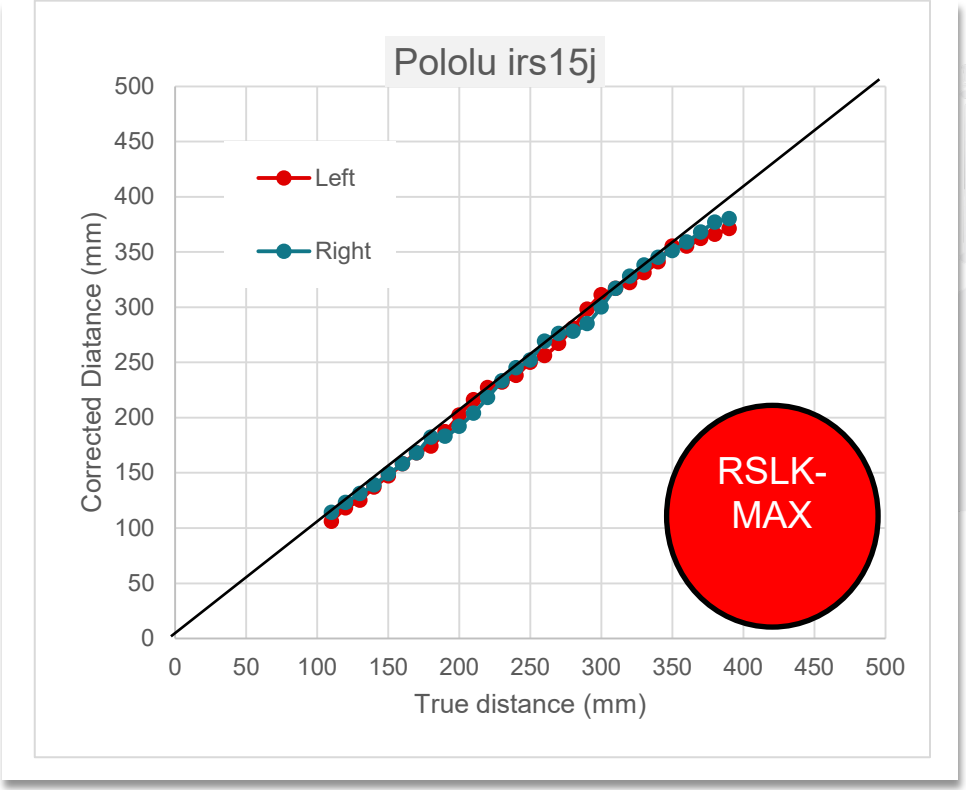
- Center of robot to wall
- Left is linear,
  - **Left** = A_{l*} **n_l** + B_l
- Right is quadratic
  - **Right** = A_r*$\mathbf{n_r^2}$ + B_r*$\mathbf{n_r}$ + C_r
- Ideal track width 500 to 700 mm
- Accuracy full scale in percent

Truth

Measured

$$\frac{100}{n}\sum_{i=0}^{n}\frac{\left|x_{ti}-x_{mi}\right|}{x_{t\max}}$$

Maximum



Summary

Analog to Digital Conversion

- Noise

Sampling

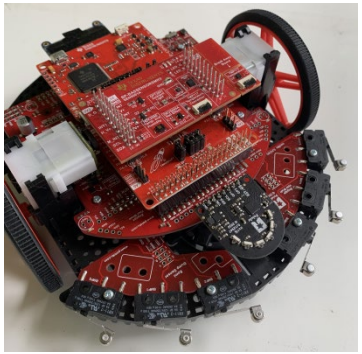
- Nyquist Theorem, Aliasing
- Central Limit Theorem

Filters

- Analog LPF
- Digital LPF

Data Acquisition Systems

- Calibration
- Accuracy



$$\frac{100}{n}\sum_{i=0}^{n}\frac{\left|x_{ti}-x_{mi}\right|}{x_{t\max}}$$

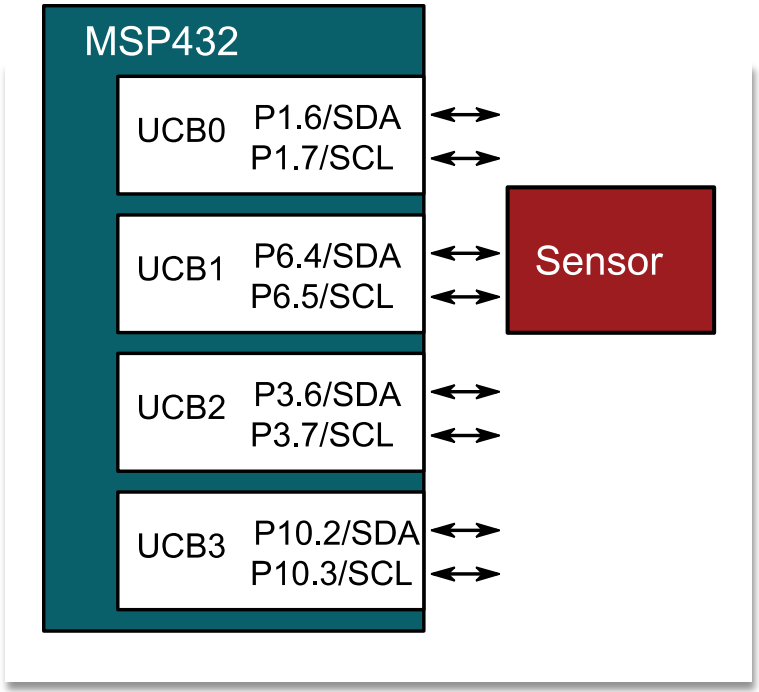
Module 21

Lecture: Sensor Integration - I^2C Communication

Serial Communication

You will learn in this module

- Communication
  - Encode
  - Transmit
  - Decode
- Inter-Integrated Circuit (I^2C) Interface
  - Serial transmission at a bit rate
  - Master/slave
  - Addressing
  - Clock/data protocol
- Performance measures
  - Bandwidth
  - Response time

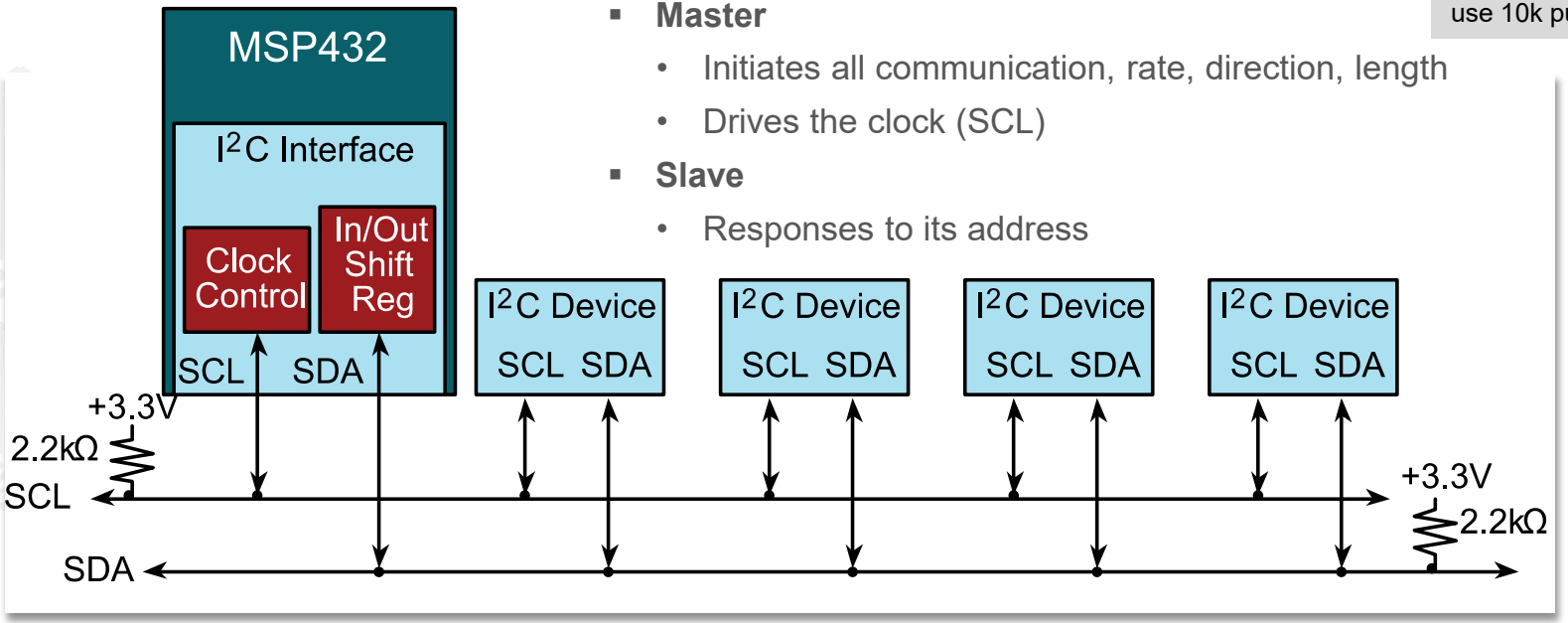


Inter-Integrated Circuit Interface (I2C)

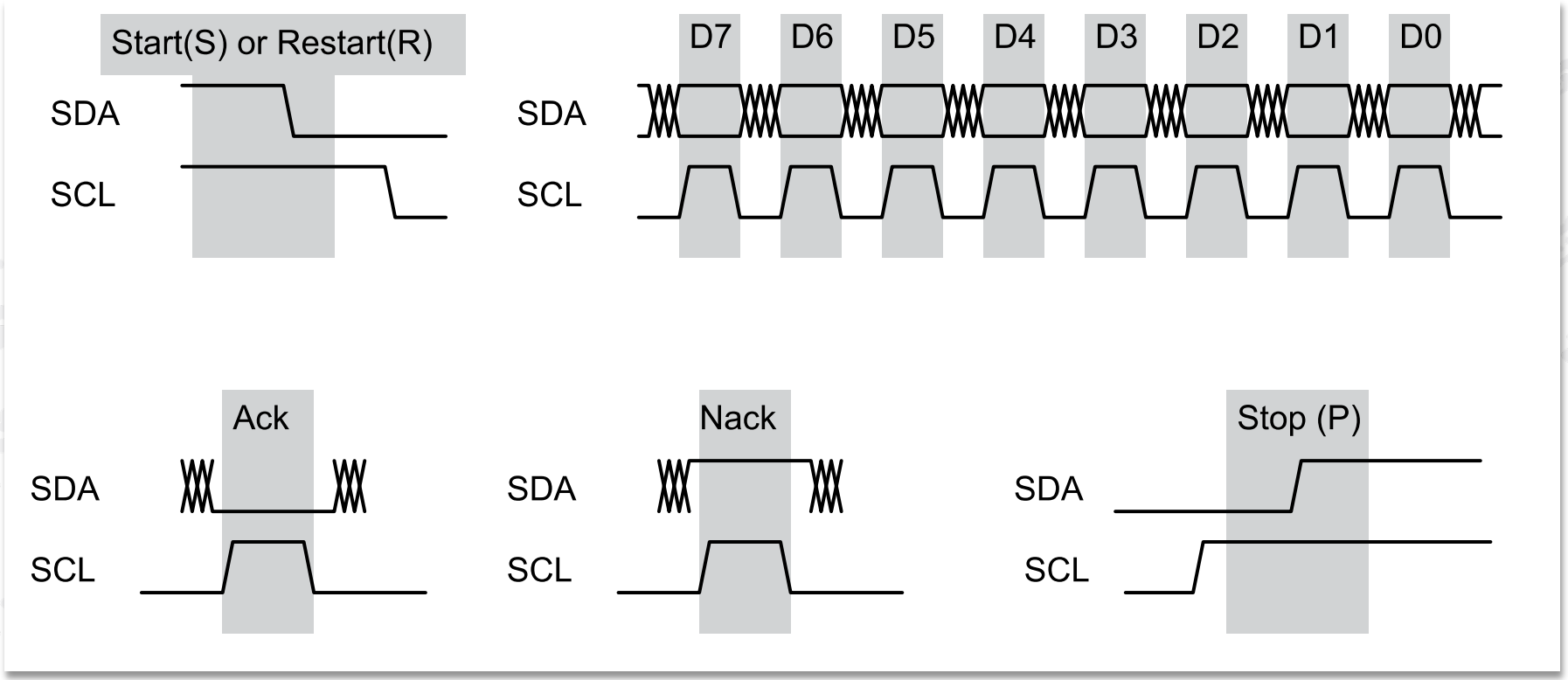
Encode
Transmit
Decode

- **Physical channel**
  - SCL is clock (bit rate is 400 kHz), synchronous serial
  - SDA is data (transmit or receive)
  - Driven low, passive pull to high
- **Master**
  - Initiates all communication, rate, direction, length
  - Drives the clock (SCL)
- **Slave**
  - Responses to its address

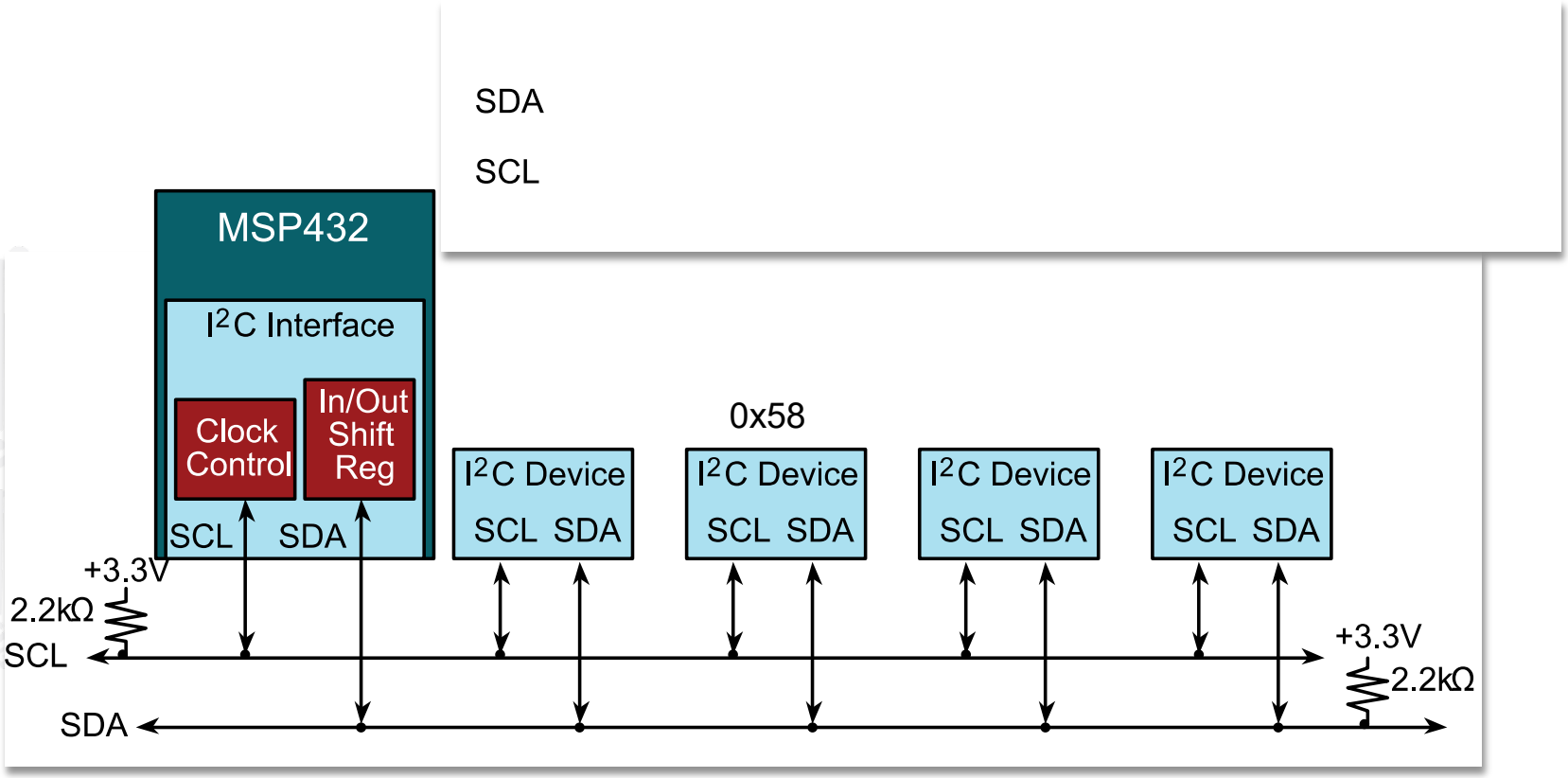
Pololu irs15j and
TI BP-
BASSENSORSMKII
use 10k pullups



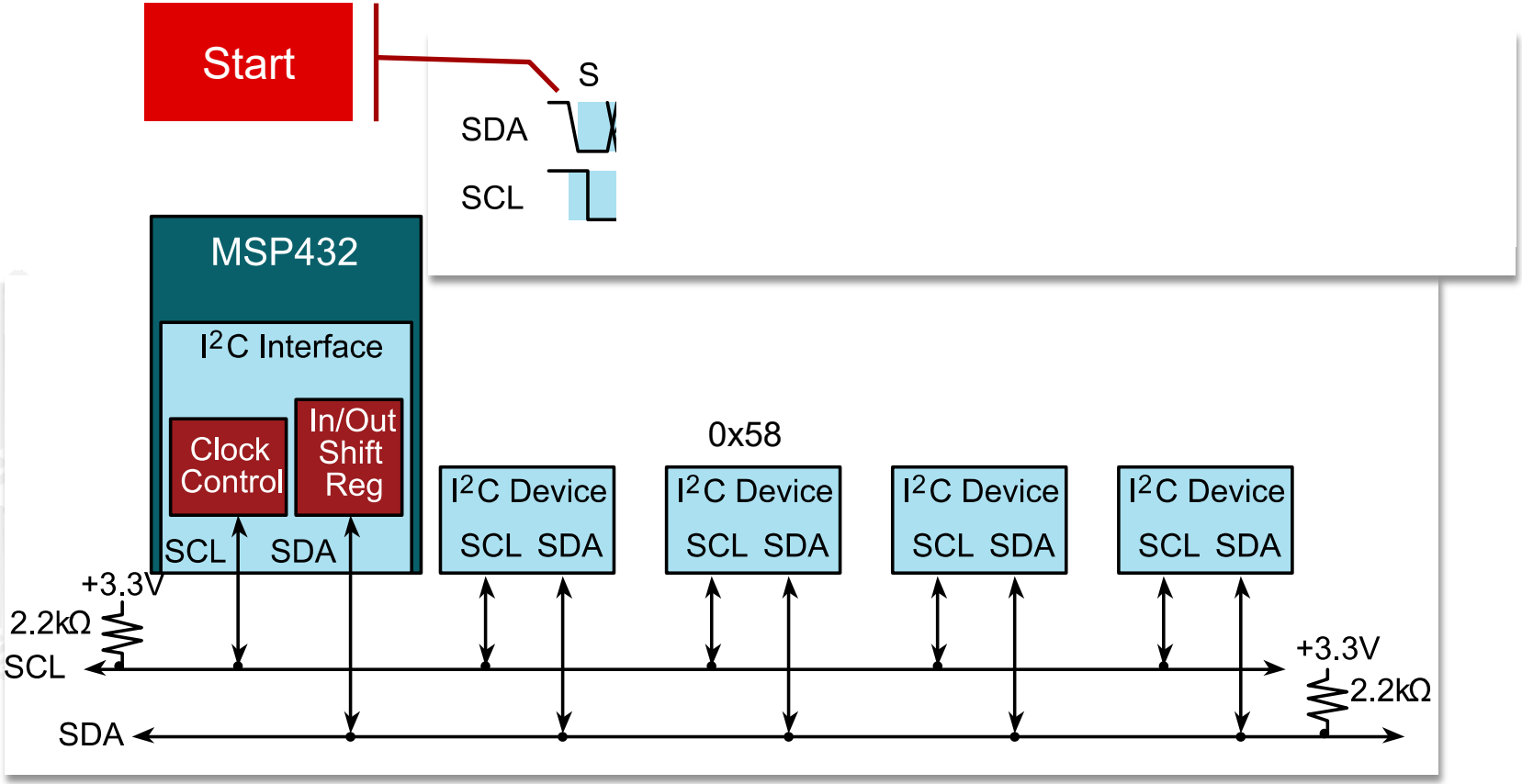
Inter-Integrated Circuit Interface (I2C)



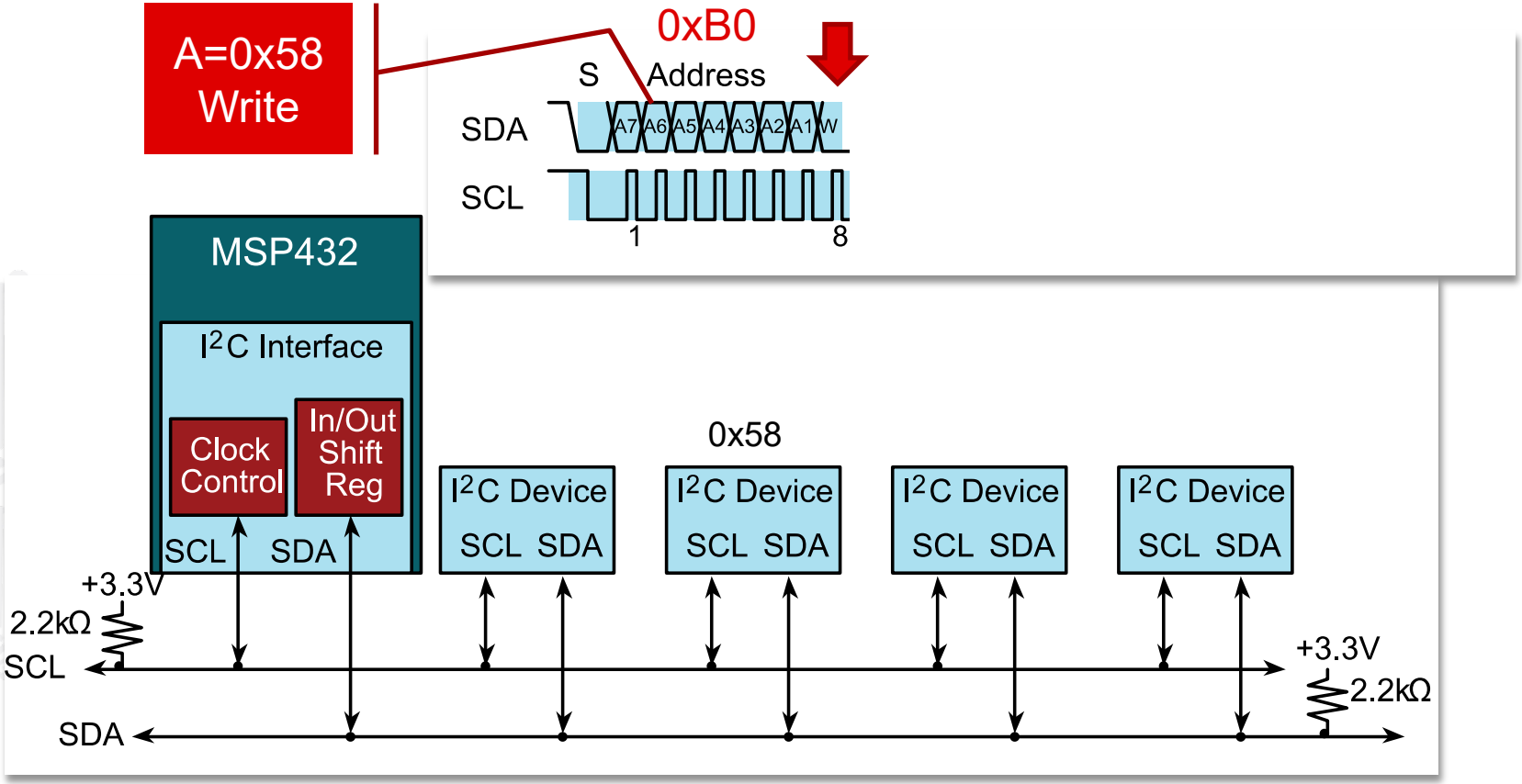
I2C Example: Master Sends Two Bytes to Slave at Address=0x58



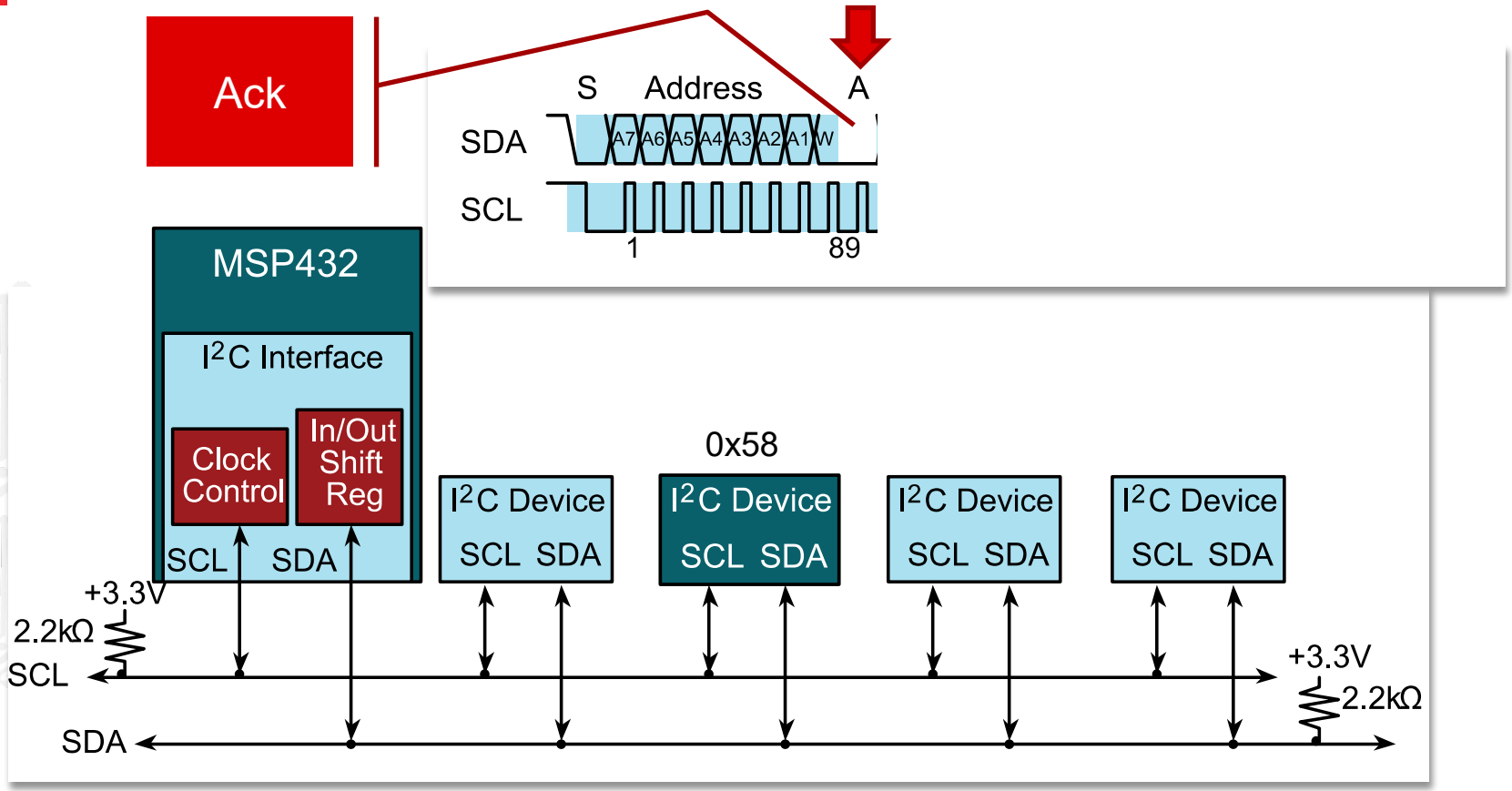
I2C Example: Master Sends Two Bytes to Slave at Address=0x58



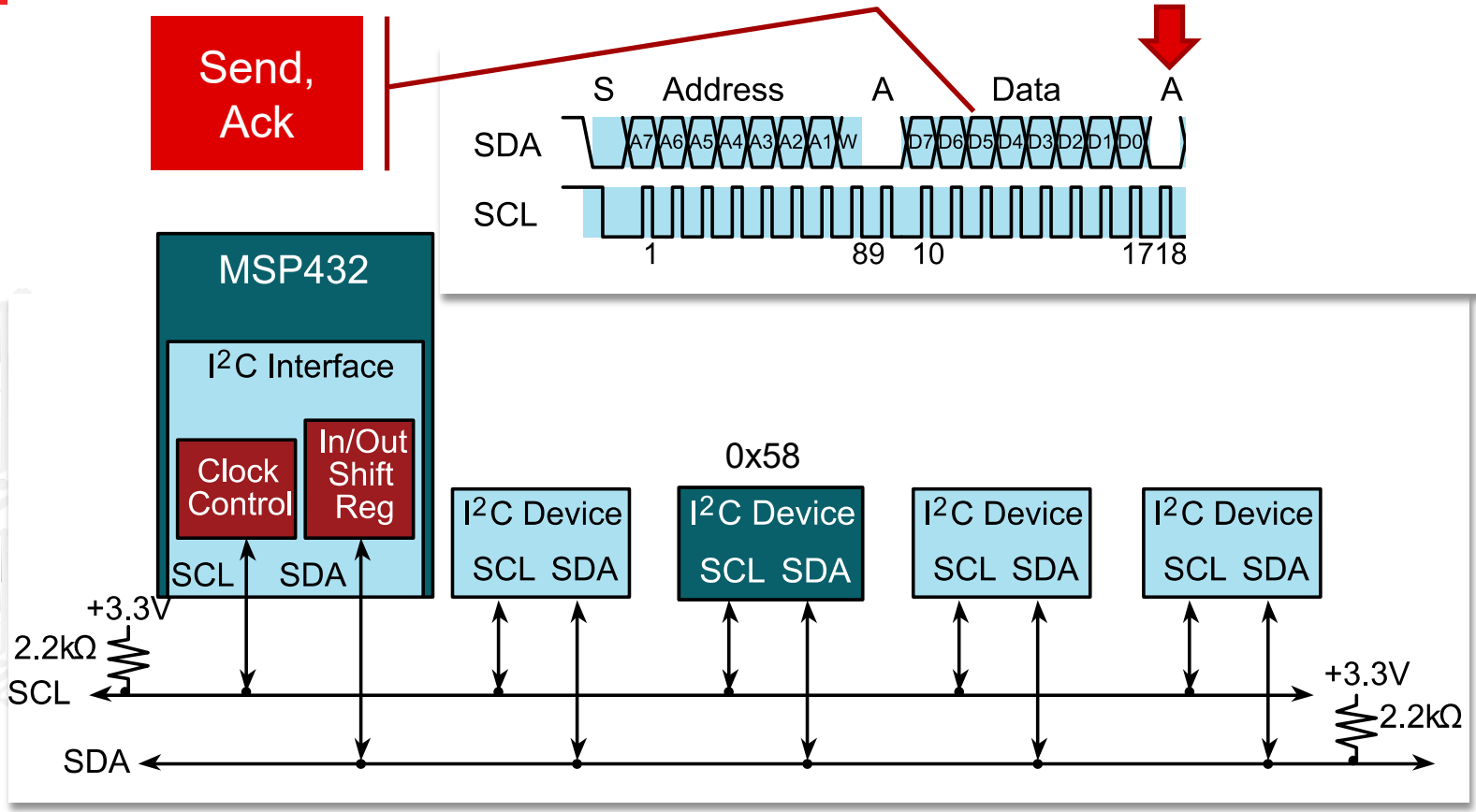
I2C Example: Master Sends Two Bytes to Slave at Address=0x58



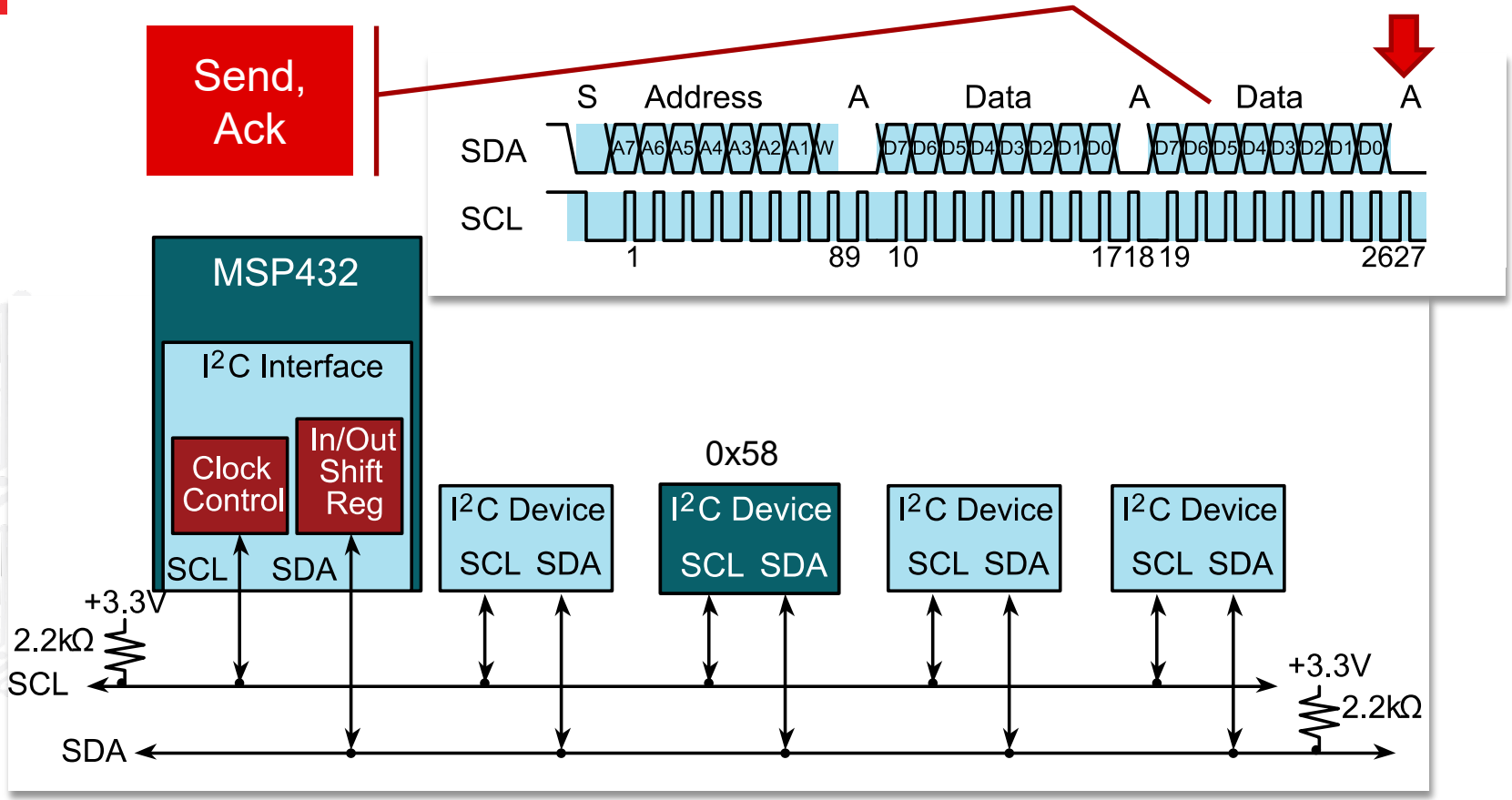
I2C Example: Master Sends Two Bytes to Slave at Address=0x58



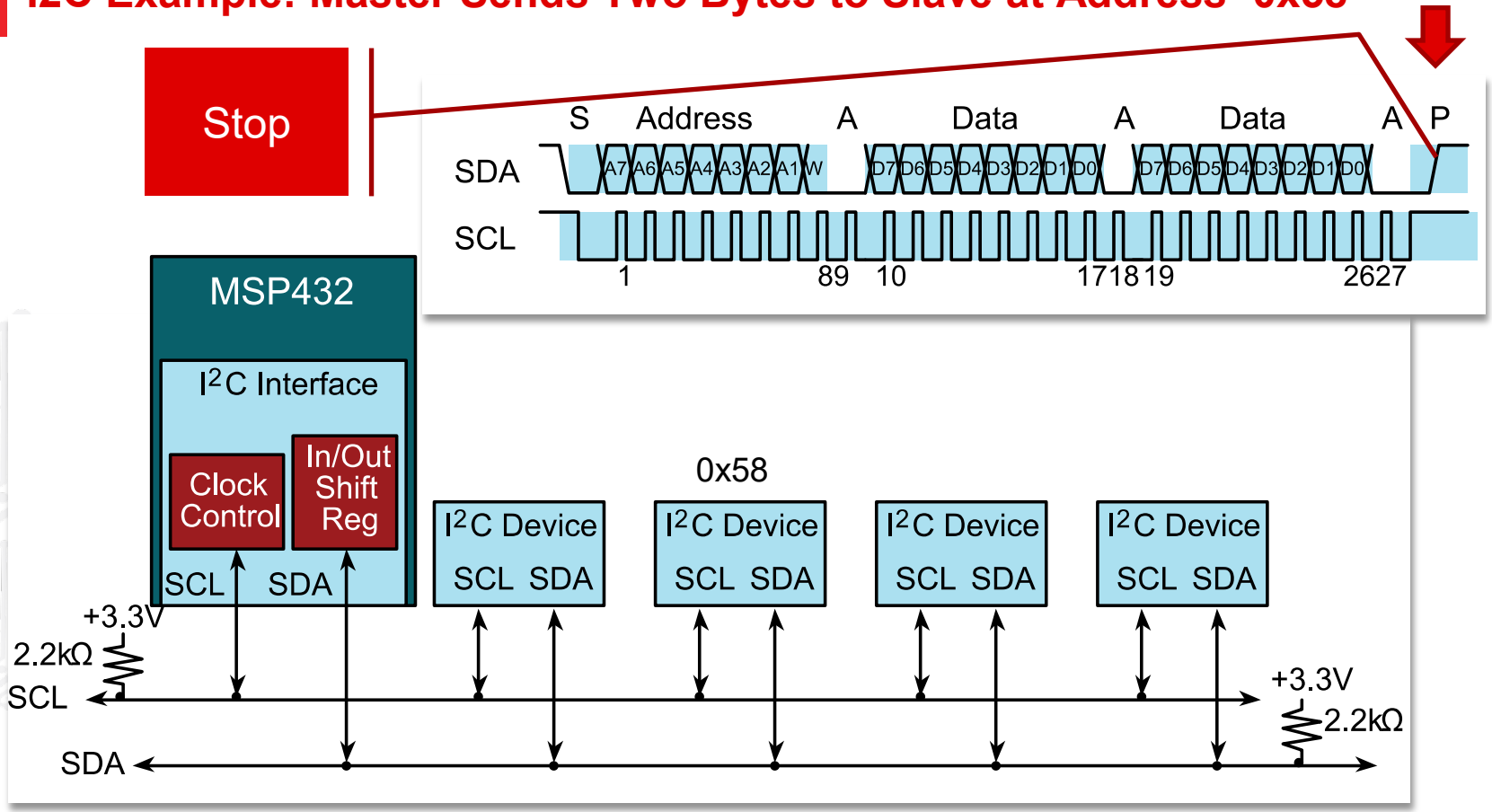
I2C Example: Master Sends Two Bytes to Slave at Address=0x58



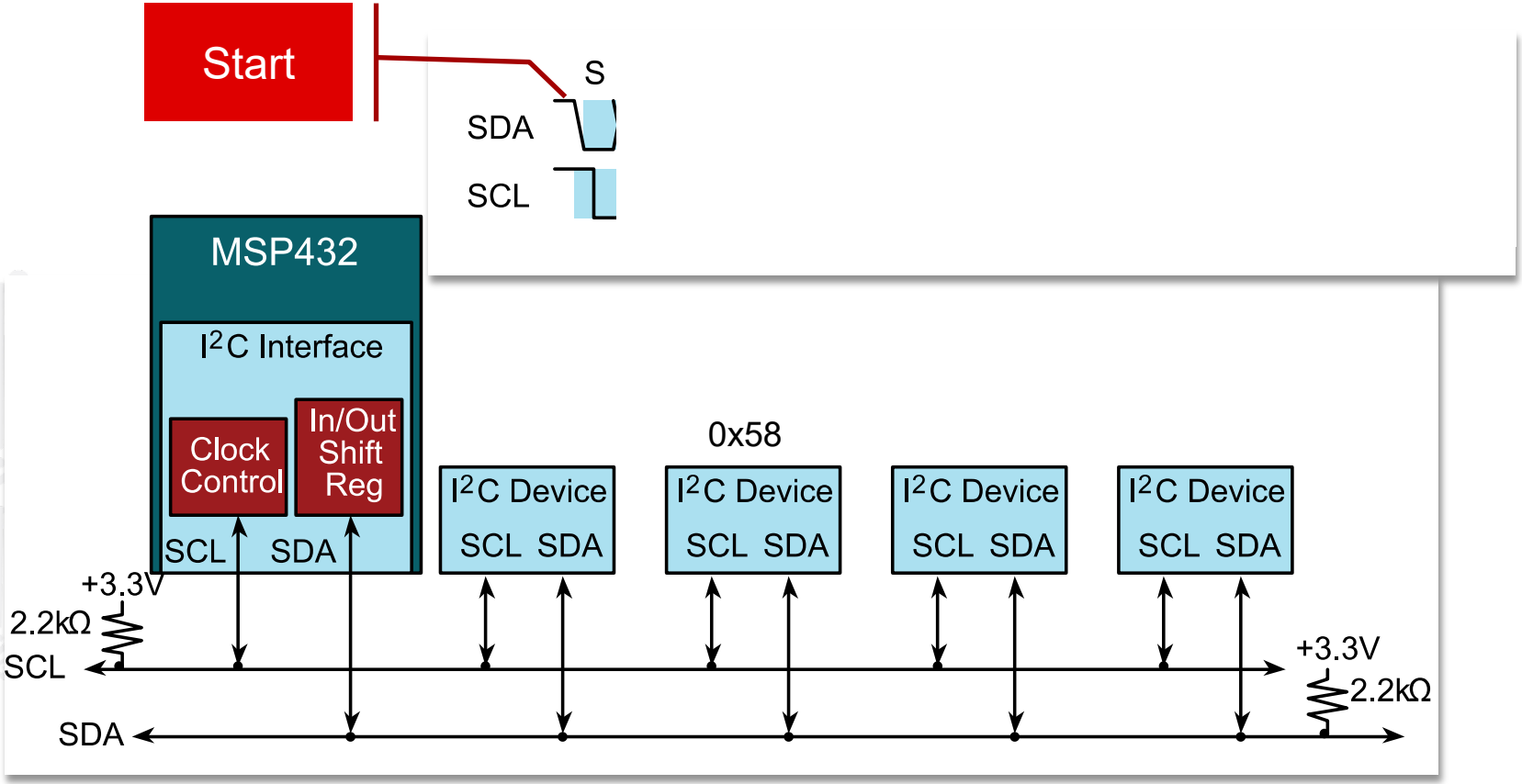
I2C Example: Master Sends Two Bytes to Slave at Address=0x58



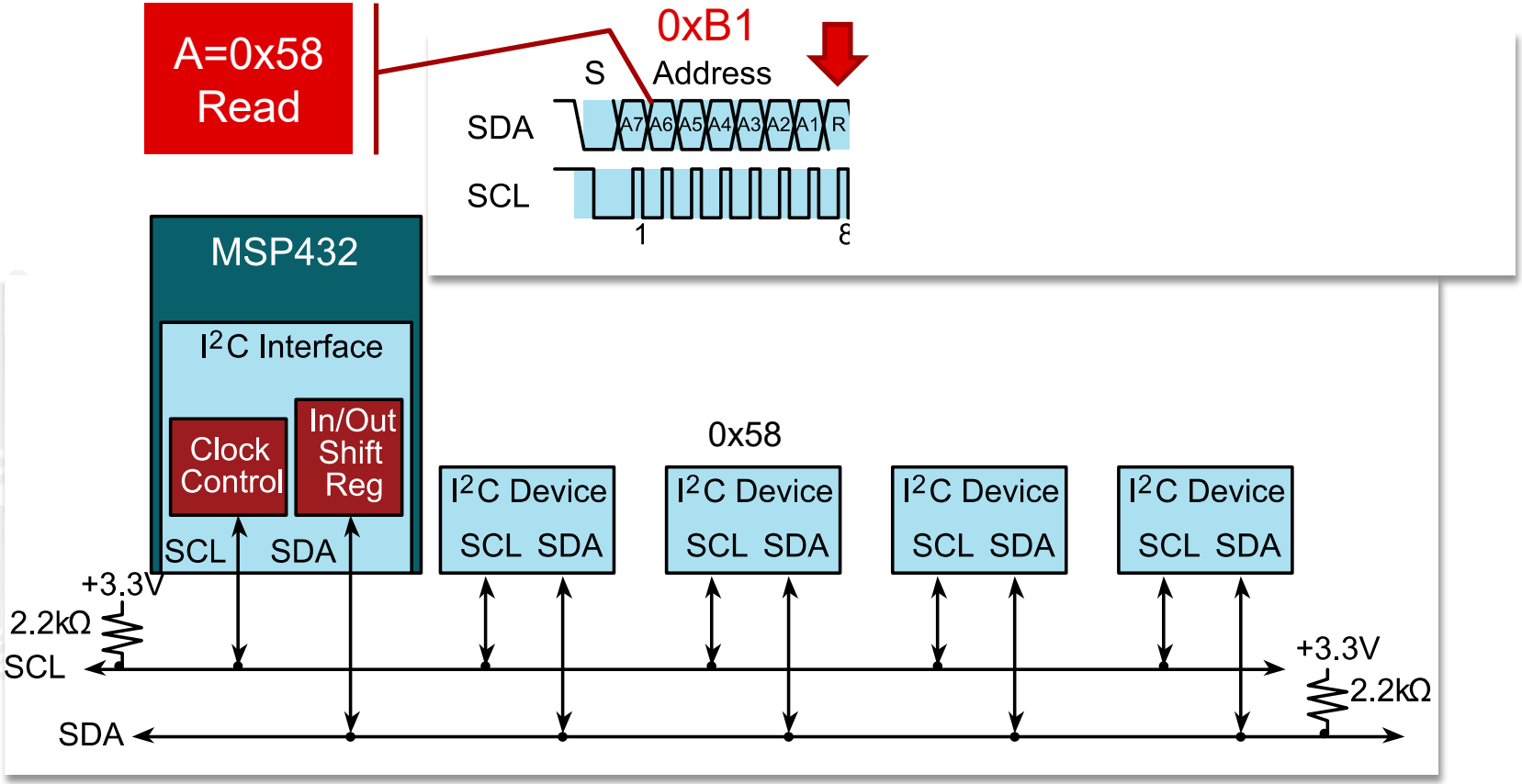
I2C Example: Master Sends Two Bytes to Slave at Address=0x58



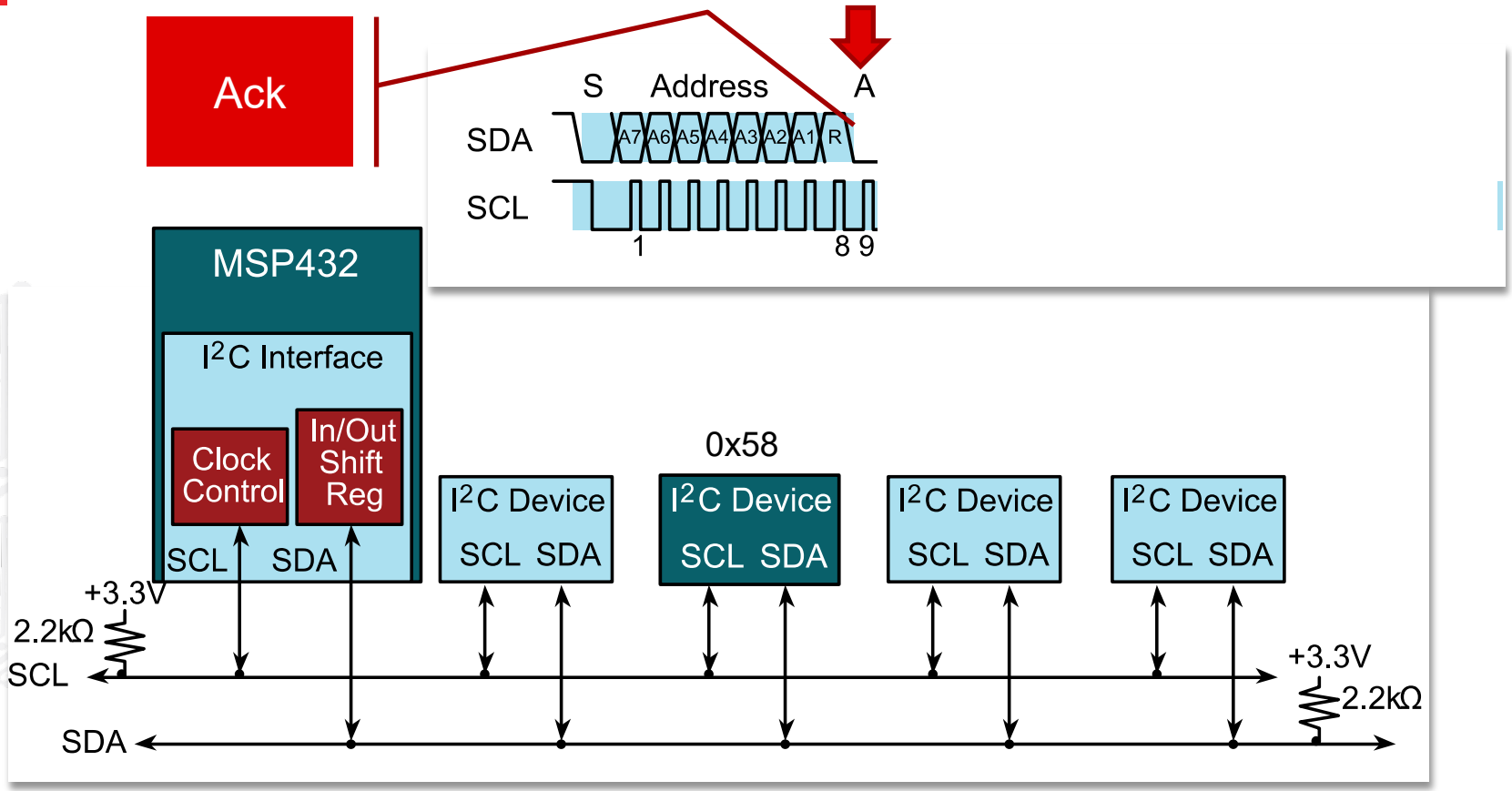
I2C Example: Master Receives Two Bytes from Slave at Address=0x58



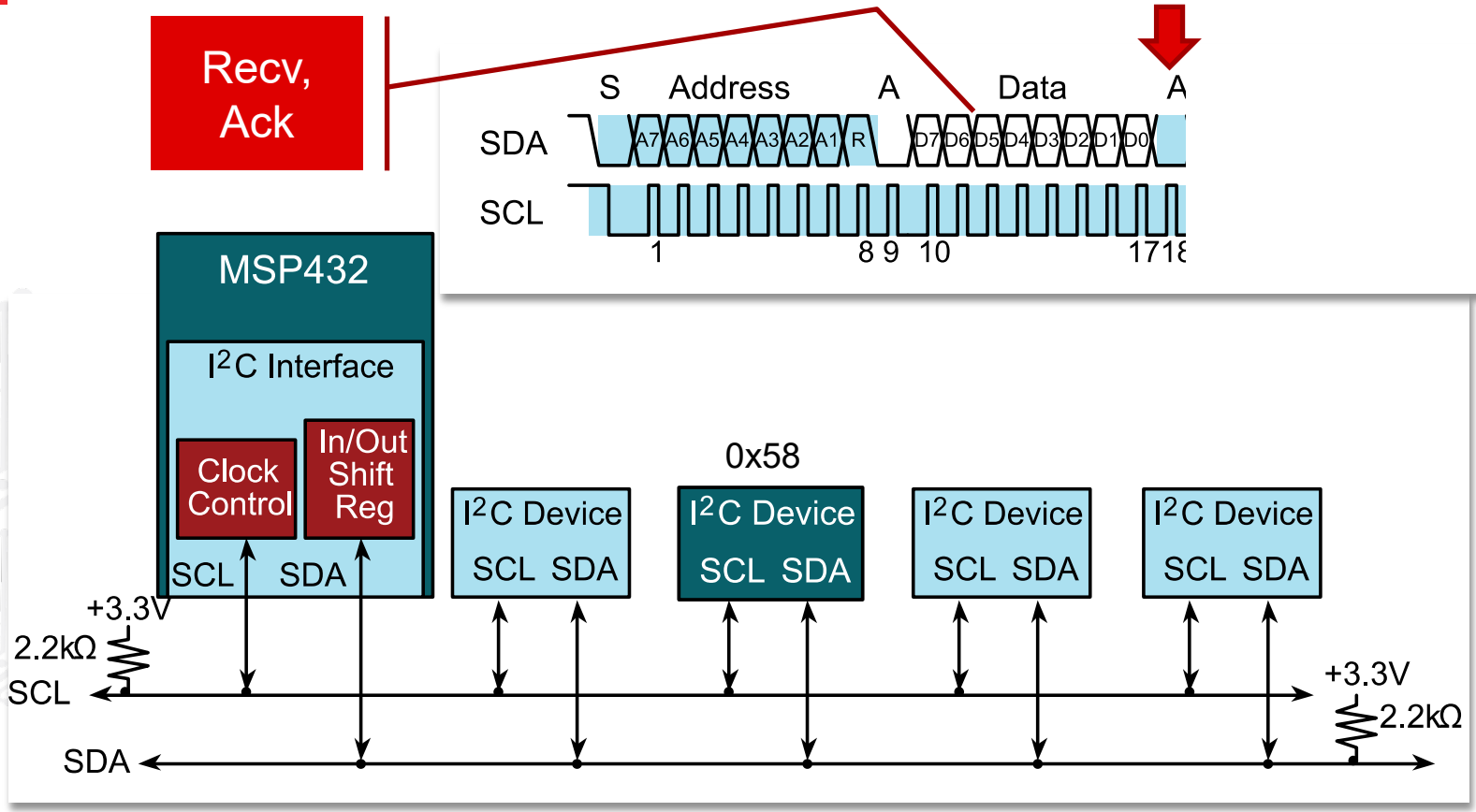
I2C Example: Master Receives Two Bytes from Slave at Address=0x58



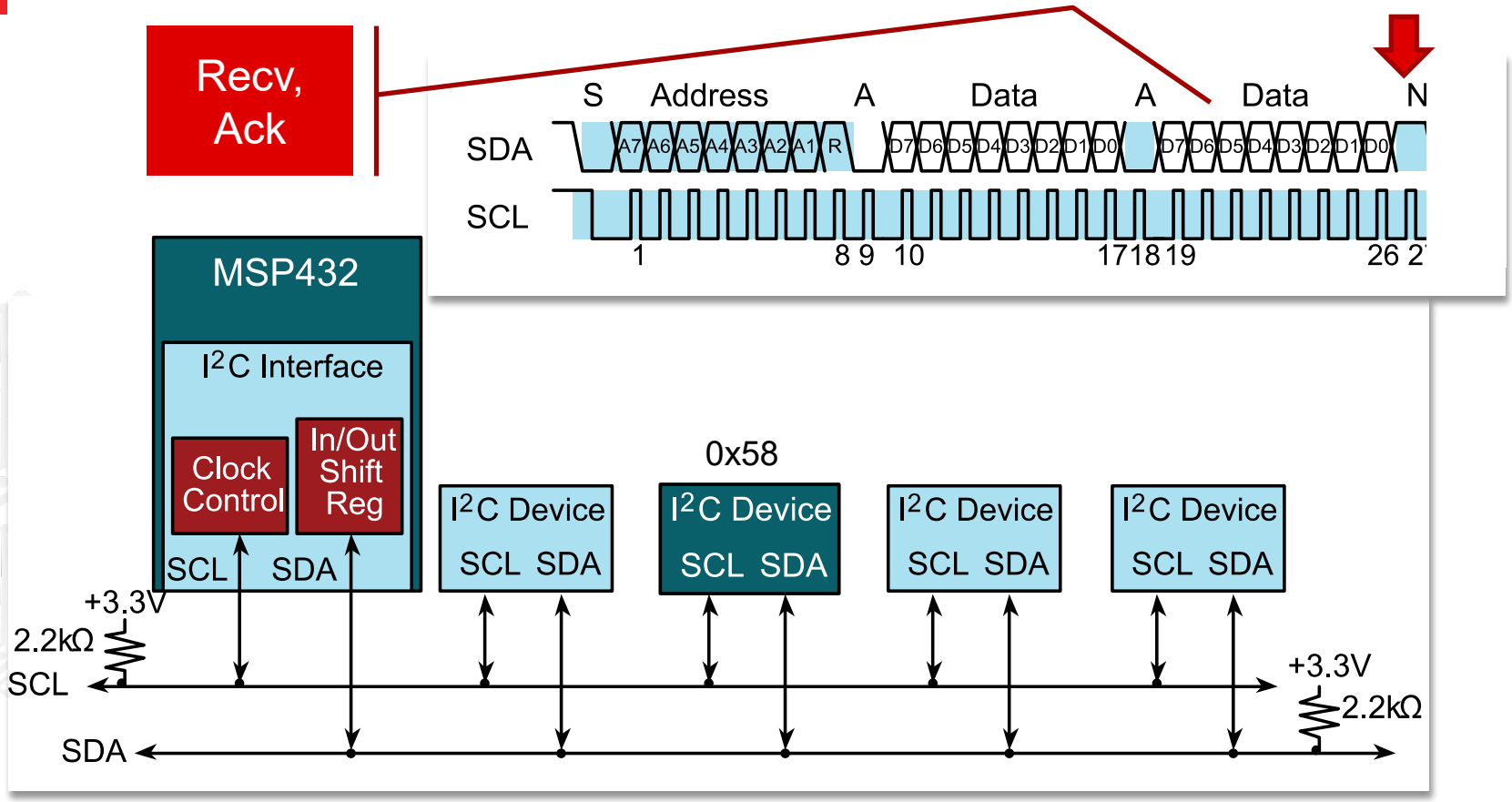
I2C Example: Master Receives Two Bytes from Slave at Address=0x58



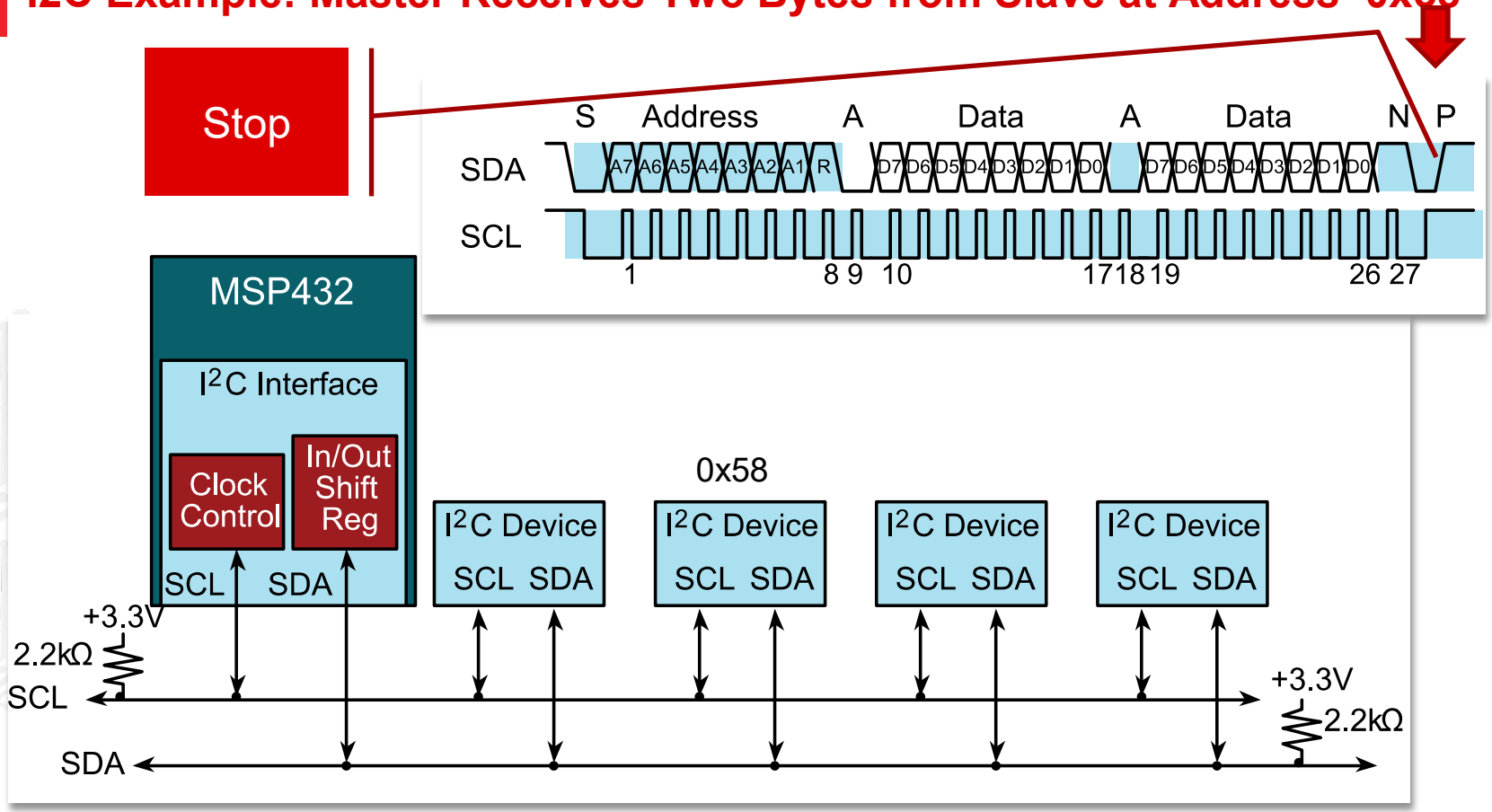
I2C Example: Master Receives Two Bytes from Slave at Address=0x58



I2C Example: Master Receives Two Bytes from Slave at Address=0x58



I2C Example: Master Receives Two Bytes from Slave at Address=0x58



I2C Example: Master Receives Six Data Bytes from OPT3101

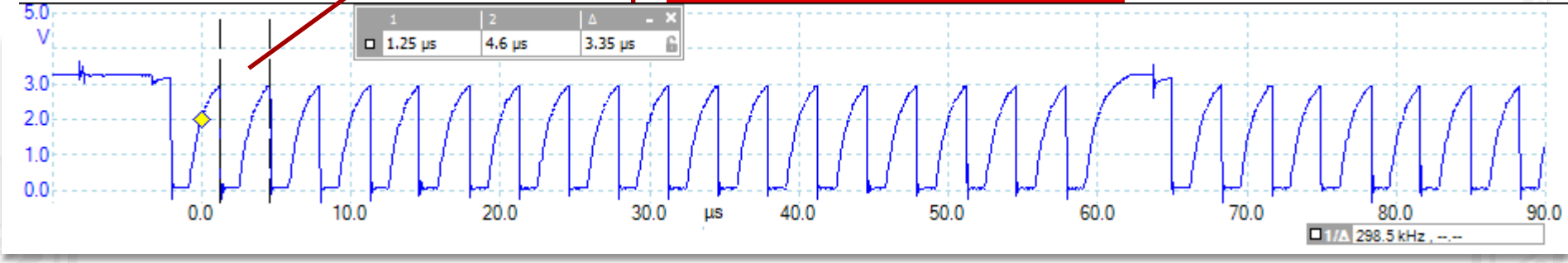
Address=0x58

Bandwidth = 48 bits/400µs = 120 kbits/sec

https://www.saleae.com/

Scope Trace of I2C SCL

Set rate to 400 kHz
Actual rate is 300 kHz



<u>Master sequence</u>

1. Drive its SCL clock low
2. Set the SDA line
3. Wait for a fixed amount of time
4. Let its SCL clock float
5. Wait for the SCL clock to be high
6. Wait for a fixed amount of time
7. Stop waiting if the clock goes low

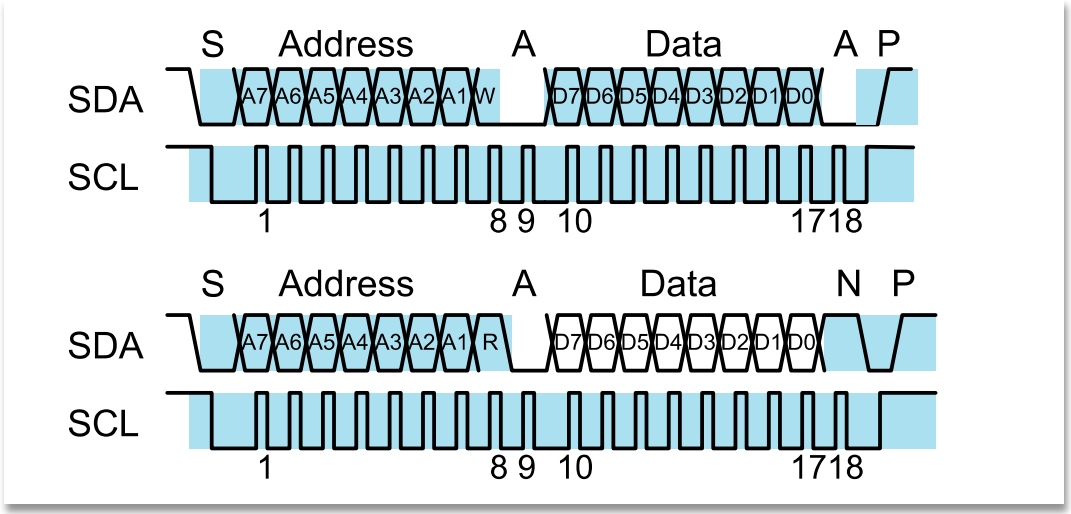
<u>Slave sequence (clock stretching)</u>

1. Wait for the SCL clock to be low
2. Drive SCL clock low
3. Wait until it's ready to capture
4. Let its SCL float
5. Wait for the SCL clock to be high
6. Capture the SDA data

Summary

I2C Communication

- One bit at a time
- Active low, passive high
- Master-slave
- Synchronous
- Busy-wait software
- Bit rate
- Bandwidth



Bit rate = 400 kbits/sec
Bandwidth = 8 bits*400kHz/20 = 160 kbits/sec

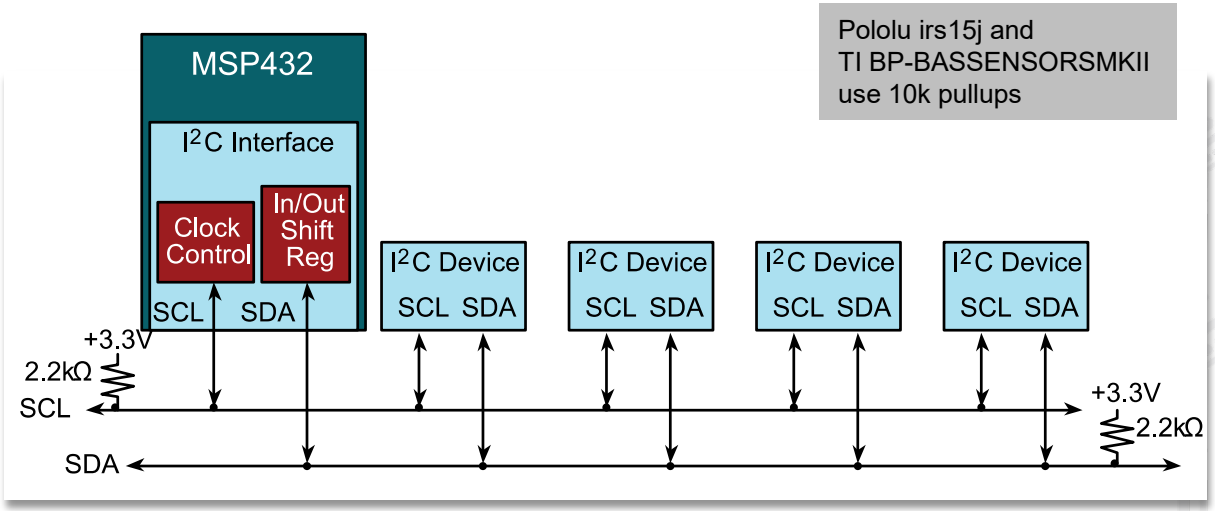
Module 21

Lecture: Sensor Integration - MSP432 I^2C Software

MSP432 I²C Software

You will learn in this module

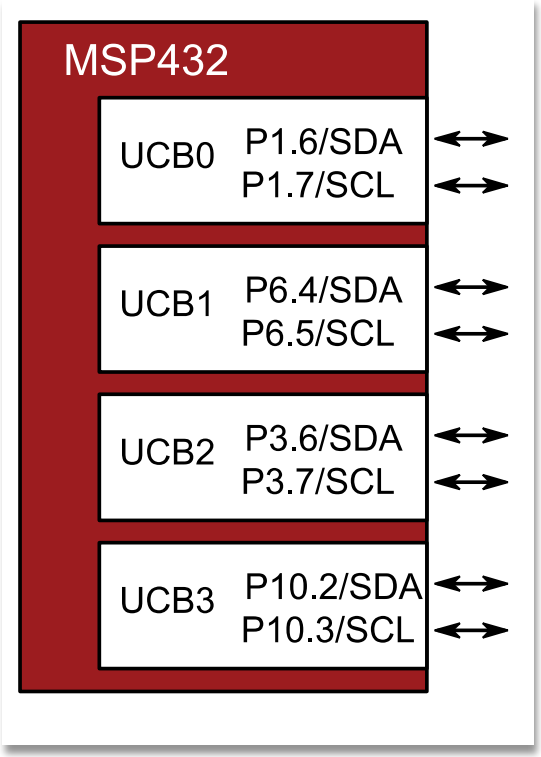
- MSP432 Initialization
  - Master
  - Bit rate
- MSP432 sends data
  1. Master sends start
  2. Master sends address
     slave acknowledge
  3. Master sends data
     slave acknowledge
  4. Master sends stop
- MSP432 receives data
  1. Master sends start
  2. Master sends address
     Slave acknowledge
  3. Slave sends data
     Master acknowledge
  4. Master sends stop



Pololu irs15j and
TI BP-BASSENSORSMKII
use 10k pullups

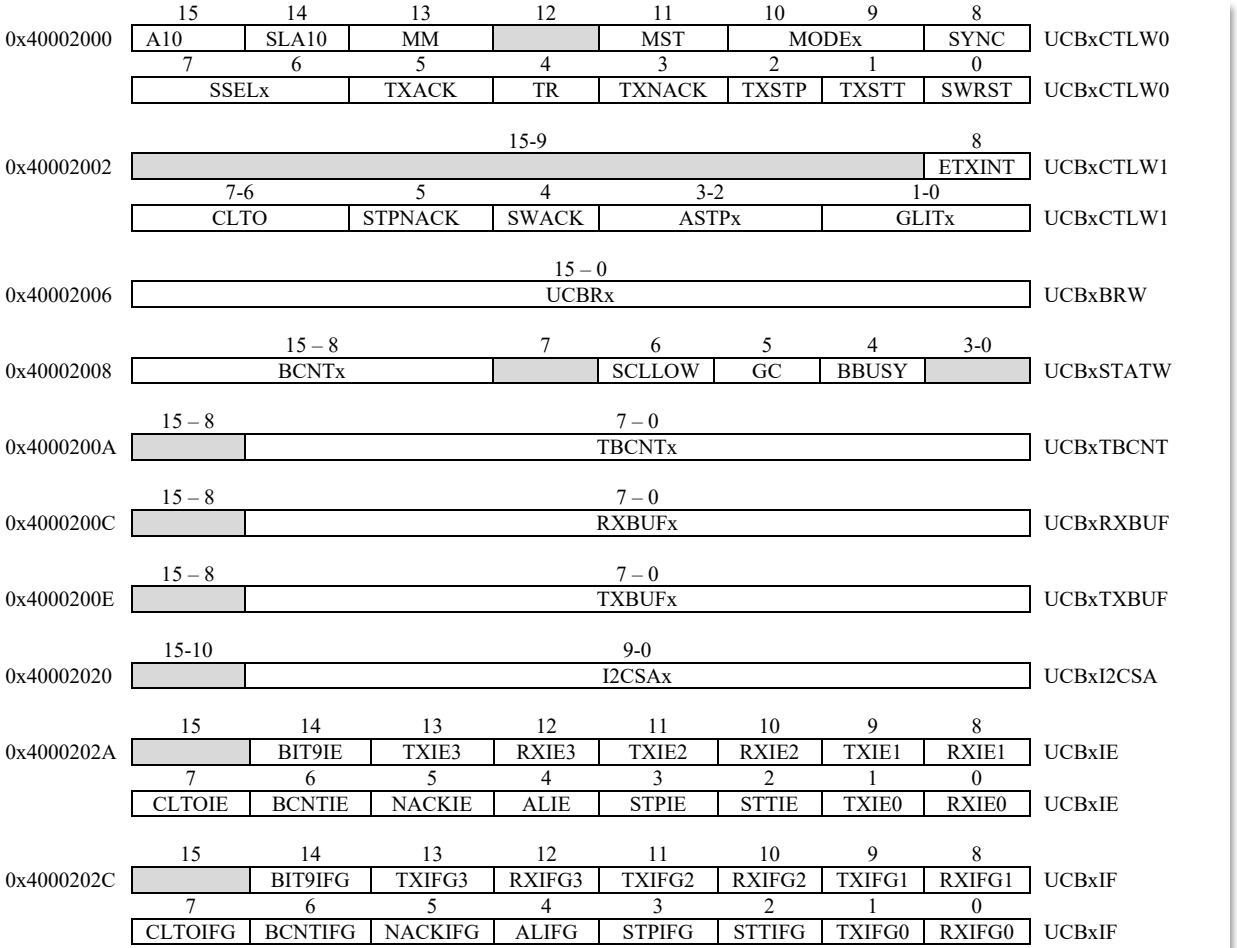
- OPT3101 ToF Distance sensor
- BMI160/BMM150 9-axis IMU
- HDC2080 Temperature/Humidity
- OPT3001 Light intensity
- TMP117 Temperature

I2C Port Selection



Pin	PxSEL1=0, PxSEL0=1
P1.6	UCB0SDA
P1.7	UCB0SCL
P6.4	UCB1SDA
P6.5	UCB1SCL
P3.6	UCB2SDA
P3.7	UCB0SCL
P10.2	UCB2SDA
P10.3	UCB3SCL

I2C Registers



	15	14	13	12	11	10	9	8	
0x40002000	A10	SLA10	MM		MST	MODEx		SYNC	UCBxCTLW0
	7	6	5	4	3	2	1	0	
	SSELx		TXACK	TR	TXNACK	TXSTP	TXSTT	SWRST	UCBxCTLW0

	15-9							8	
0x40002002								ETXINT	UCBxCTLW1
	7-6		5	4	3-2		1-0		
	CLTO		STPNACK	SWACK	ASTPx		GLITx		UCBxCTLW1

	15 – 0	
0x40002006	UCBRx	UCBxBRW

	15 – 8	7	6	5	4	3-0	
0x40002008	BCNTx		SCLLOW	GC	BBUSY		UCBxSTATW

	15 – 8	7 – 0	
0x4000200A		TBCNTx	UCBxTBCNT

	15 – 8	7 – 0	
0x4000200C		RXBUFx	UCBxRXBUF

	15 – 8	7 – 0	
0x4000200E		TXBUFx	UCBxTXBUF

	15-10	9-0	
0x40002020		I2CSAx	UCBxI2CSA

	15	14	13	12	11	10	9	8	
0x4000202A		BIT9IE	TXIE3	RXIE3	TXIE2	RXIE2	TXIE1	RXIE1	UCBxIE
	7	6	5	4	3	2	1	0	
	CLTOIE	BCNTIE	NACKIE	ALIE	STPIE	STTIE	TXIE0	RXIE0	UCBxIE

	15	14	13	12	11	10	9	8	
0x4000202C		BIT9IFG	TXIFG3	RXIFG3	TXIFG2	RXIFG2	TXIFG1	RXIFG1	UCBxIF
	7	6	5	4	3	2	1	0	
	CLTOIFG	BCNTIFG	NACKIFG	ALIFG	STPIFG	STTIFG	TXIFG0	RXIFG0	UCBxIF

MSP432 I²C Initialization (1 of 4)

```
// prescale sets bit rate = 12MHz/prescale
// prescale=30 means bit rate =12MHz/30 = 400 kHz
void I2CB1_Init(uint32_t prescale){
    // initialize eUSCI
  EUSCI_B1->CTLW0 |= 0x0001;     // hold in reset
```

MSP432 I²C Initialization (2 of 4)

UCB1CTLW0

bit15	UCA10 = 0; own address is 7-bit address
bit14	UCSLA10 = 0; address slave with 7-bit address
bit13	UCMM = 0; single master environment
bit11	UCMST = 1; master mode
bits10-9	UCMODEx = 3; I2C mode
bit8	UCSYNC = 1; synchronous mode
bits7-6	UCSSELx = 3; eUSCI clock SMCLK (12 MHz)
bit0	UCSWRST = 1; reset enabled

```
EUSCI_B1->CTLW0 = 0x0FC1;
```

MSP432 I²C Initialization (3 of 4)

UCB1CTLW1

bits7-6 UCCLTO = 0; disable timeout clock
bit5 UCSTPNACK = 0; send negative ack before stop in receive
bit4 UCSWACK = 0; slave address ack controlled by hardware
bits3-2 UCASTPx = 0; no automatic stop condition after UCB0TBCNT
bits1-0 UCGLITx = 0 deglitch time of 50 ns

```
EUSCI_B1->CTLW1 = 0;
```

MSP432 I²C Initialization (4 of 4)

eUSCI gets its clock from SMCLK
Clock_Init48MHz() SMCLK = HFXTCLK/4 = 12 MHz
prescale is divide by 120 for 100 kHz bit rate clock
prescale is divide by 30 for 400 kHz bit rate clock

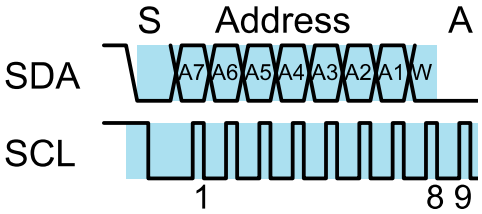
```
  EUSCI_B1->BRW = prescale; // bit clock prescaler
  P6->SEL0 |= 0x30;
  P6->SEL1 &= ~0x30;  // P6.4, 6.5 as UCB1SDA, UCB1SCL
  EUSCI_B1->CTLW0 &= ~0x0001; // enable eUSCI_B1
  EUSCI_B1->IE = 0x0000;      // disable all interrupts
}
```

MSP432 I²C Transmit Data


```
void I2CB1_Send1(uint8_t slaveAddr, uint8_t data){
  while(EUSCI_B1->STATW&0x0010){}; // wait UCBBUSY
```

Bandwidth = 1 byte*400kHz/20 = 20 kbytes/sec

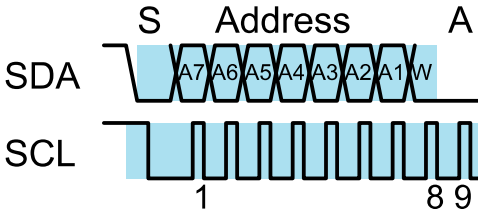
MSP432 I²C Transmit Data




```
void I2CB1_Send1(uint8_t slaveAddr, uint8_t data){
  while(EUSCI_B1->STATW&0x0010){}; // wait UCBBUSY
  EUSCI_B1->I2CSA = slaveAddr; // set slave address
// I2C master transmit mode
  EUSCI_B1->CTLW0 = (EUSCI_B1->CTLW0 & ~0x0004)|0x0012;
// Master transmit, no stop, start condition
  while((EUSCI_B1->IFG&0x0002)==0){};// wait UCTXIFG0
```

Bandwidth = 1 byte*400kHz/20 = 20 kbytes/sec

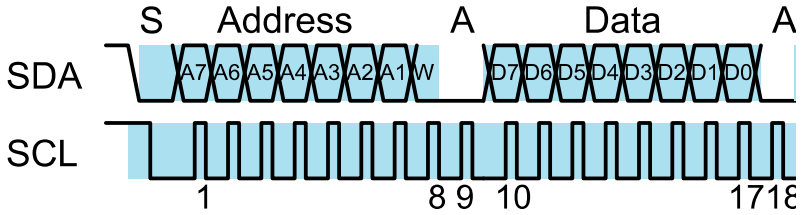
MSP432 I²C Transmit Data




```
void I2CB1_Send1(uint8_t slaveAddr, uint8_t data){
  while(EUSCI_B1->STATW&0x0010){}; // wait UCBBUSY
  EUSCI_B1->I2CSA = slaveAddr; // set slave address
// I2C master transmit mode
  EUSCI_B1->CTLW0 = (EUSCI_B1->CTLW0 & ~0x0004)|0x0012;
// Master transmit, no stop, start condition
  while((EUSCI_B1->IFG&0x0002)==0){};// wait UCTXIFG0
```

Bandwidth = 1 byte*400kHz/20 = 20 kbytes/sec

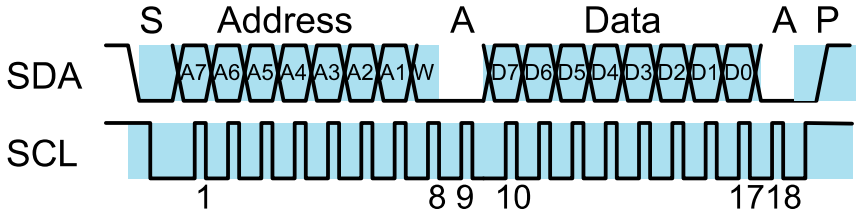
MSP432 I²C Transmit Data




```
void I2CB1_Send1(uint8_t slaveAddr, uint8_t data){
  while(EUSCI_B1->STATW&0x0010){}; // wait UCBBUSY
  EUSCI_B1->I2CSA = slaveAddr; // set slave address
// I2C master transmit mode
  EUSCI_B1->CTLW0 = (EUSCI_B1->CTLW0 & ~0x0004)|0x0012;
// Master transmit, no stop, start condition
  while((EUSCI_B1->IFG&0x0002)==0){};// wait UCTXIFG0
  EUSCI_B1->TXBUF = data;
  while((EUSCI_B1->IFG&0x0002)==0){}; // wait UCTXIFG0
```

Bandwidth = 1 byte*400kHz/20 = 20 kbytes/sec

MSP432 I²C Transmit Data




```
void I2CB1_Send1(uint8_t slaveAddr, uint8_t data){
  while(EUSCI_B1->STATW&0x0010){}; // wait UCBBUSY
  EUSCI_B1->I2CSA = slaveAddr; // set slave address
// I2C master transmit mode
  EUSCI_B1->CTLW0 = (EUSCI_B1->CTLW0 & ~0x0004)|0x0012;
// Master transmit, no stop, start condition
  while((EUSCI_B1->IFG&0x0002)==0){};// wait UCTXIFG0
  EUSCI_B1->TXBUF = data;
  while((EUSCI_B1->IFG&0x0002)==0){}; // wait UCTXIFG0
  EUSCI_B1->CTLW0 |= 0x0004; // generate UCTXSTP
  EUSCI_B1->IFG &= ~0x0002;  // clear UCTXIFG0
}
```

Bandwidth = 1 byte*400kHz/20 = 20 kbytes/sec

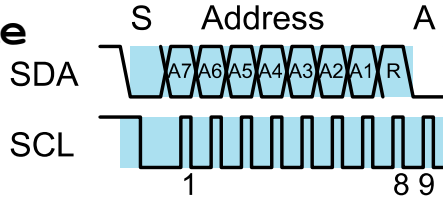
MSP432 I²C Receive Data

```
uint8_t  I2CB1_Recv1(int8_t slaveAddr){ int8_t data;
  while(EUSCI_B1->STATW&0x0010){};  // wait I2C ready
```

Bandwidth = 1 byte*400kHz/20 = 20 kbytes/sec

MSP432 I²C Receive Data

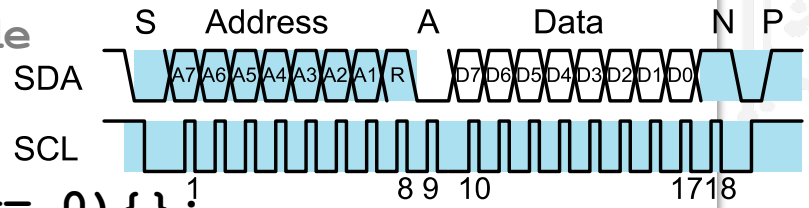
```
uint8_t  I2CB1_Recv1(int8_t slaveAddr){ int8_t data;
  while(EUSCI_B1->STATW&0x0010){};  // wait I2C ready
  EUSCI_B1->CTLW0 |= 0x0001;    // hold in reset mode
  EUSCI_B1->TBCNT = 1; // generate stop after 1 byte
  EUSCI_B1->CTLW0 &= ~0x0001;   // enable eUSCI module
  EUSCI_B1->I2CSA = slaveAddr;  // slave address
  EUSCI_B1->CTLW0 = ((EUSCI_B1->CTLW0&~0x0010)|0x0006);
// bit4=0 (UCTR) for receive mode
// bit2=1 (UCTXSTP) for stop
// bit1=1 (UCTXSTT) for start
```



Bandwidth = 1 byte*400kHz/20 = 20 kbytes/sec

MSP432 I²C Receive Data

```
uint8_t  I2CB1_Recv1(int8_t slaveAddr){ int8_t data;
  while(EUSCI_B1->STATW&0x0010){};  // wait I2C ready
  EUSCI_B1->CTLW0 |= 0x0001;    // hold in reset mode
  EUSCI_B1->TBCNT = 1; // generate stop after 1 byte
  EUSCI_B1->CTLW0 &= ~0x0001;   // enable eUSCI module
  EUSCI_B1->I2CSA = slaveAddr;  // slave address
  EUSCI_B1->CTLW0 = ((EUSCI_B1->CTLW0&~0x0010)|0x0006);
// bit4=0 (UCTR) for receive mode
// bit2=1 (UCTXSTP) for stop
// bit1=1 (UCTXSTT) for start
  while((EUSCI_B1->IFG&0x0001) == 0){};
// if no slave at this address then this hangs
  data = EUSCI_B1->RXBUF;       // get the reply
  return data;
}
```

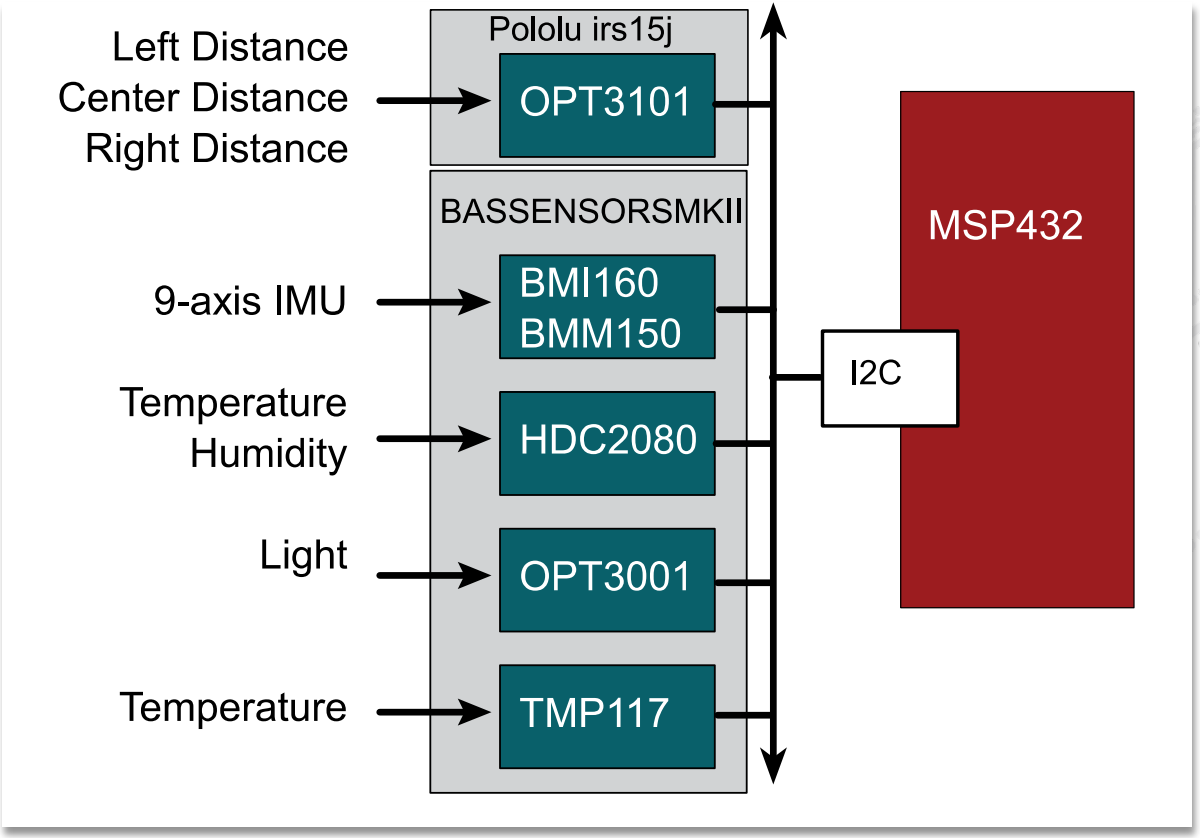


Bandwidth = 1 byte*400kHz/20 = 20 kbytes/sec

Summary

Serial Communication

- One bit at a time
  Data pin SDA
- Master-slave
- Synchronous
  Clock pin SCL
- Digital encoding
  Active low
  10k pullup passive high
- Busy-wait
- Bit rate
- Bandwidth



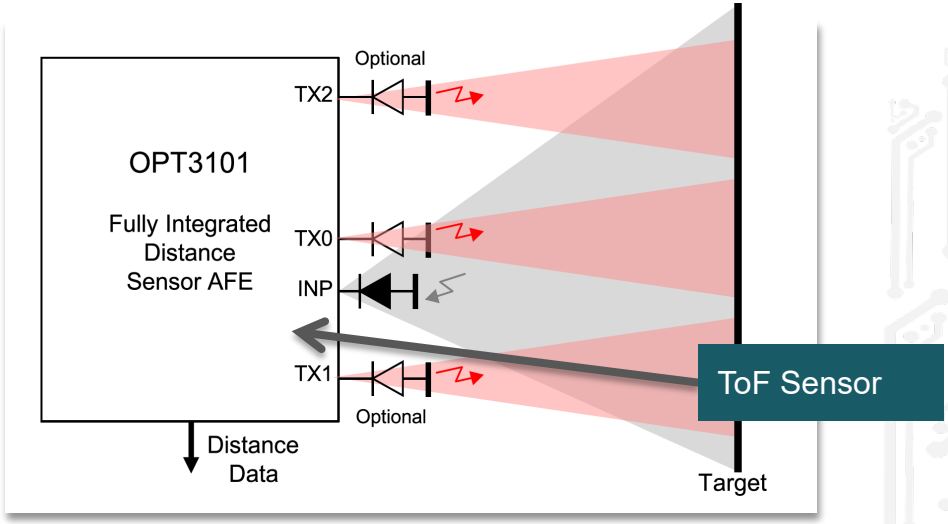
Module 21

Lecture: Sensor Integration - OPT3101 ToF Distance Sensor

OPT3101 Time Of Flight (ToF) Distance Sensor

You will learn in this module

- Integrating Sensors
  - Physical to electrical conversion
  - Analog to digital conversion
  - Range, resolution, precision
- Sensor Interfacing
  - I2C protocol
  - MSP432 software driver
  - Periodic sampling, Nyquist Theorem
- Digital Signal Processing
  - Digital filters, Central Limit Theorem
  - Sensor integration



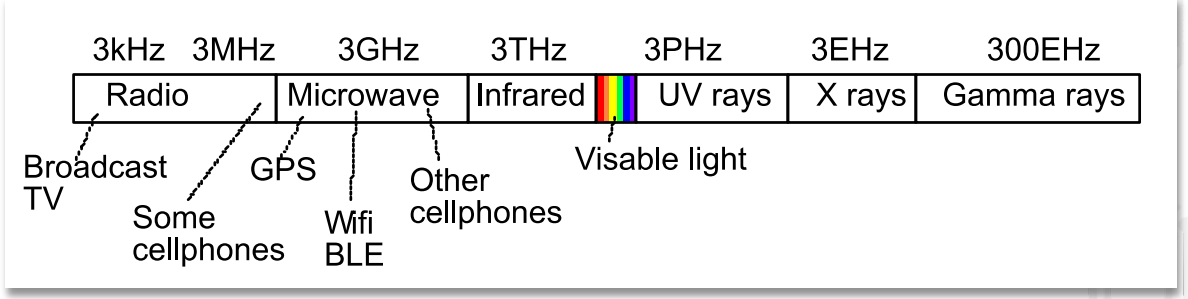
Pololu 3-Channel, Wide FOV Time-of-Flight Distance Sensor using OPT3101 for TI-RSLK MAX (part number Pololu #3680)

TI OPT3101 reference document: SBAU305B Introduction to Time-of-Flight Long Range Proximity and Distance Sensor System Design

Light

In a vacuum

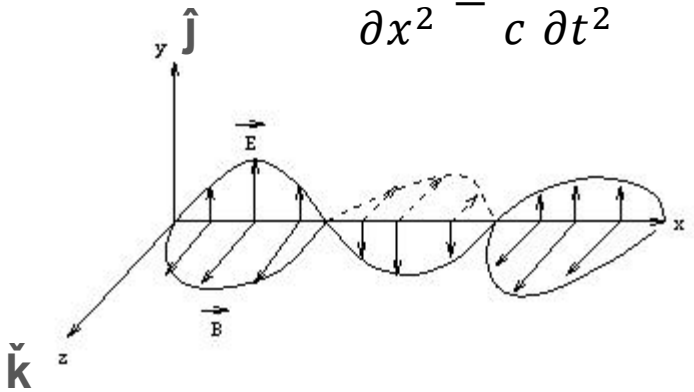
- **c** = 299,792.458 km/s
- Maxwell's equations
  $\mathbf{E}(x,t) = E_{max}\sin(2\pi(\mathbf{f}\,t - x/\boldsymbol{\lambda} + \varphi))\hat{\jmath}$,
  $\mathbf{B}(x,t) = B_{max}\sin(2\pi(\mathbf{f}\,t - x/\boldsymbol{\lambda} + \varphi))\check{k}$
  $\mathbf{c} = E_{max}/B_{max}$
- **f** is the frequency
- **λ** is the wavelength



In a medium

- **ν** speed at which light travels in a medium
  $\nu = \mathbf{f}\,\boldsymbol{\lambda}$
- **n** refractive index (**c** / **ν**) of a medium
- **ν** = 299,700 km/s in air

$$\frac{\partial^2 E}{\partial x^2} = \frac{1}{c}\frac{\partial^2 E}{\partial t^2}$$

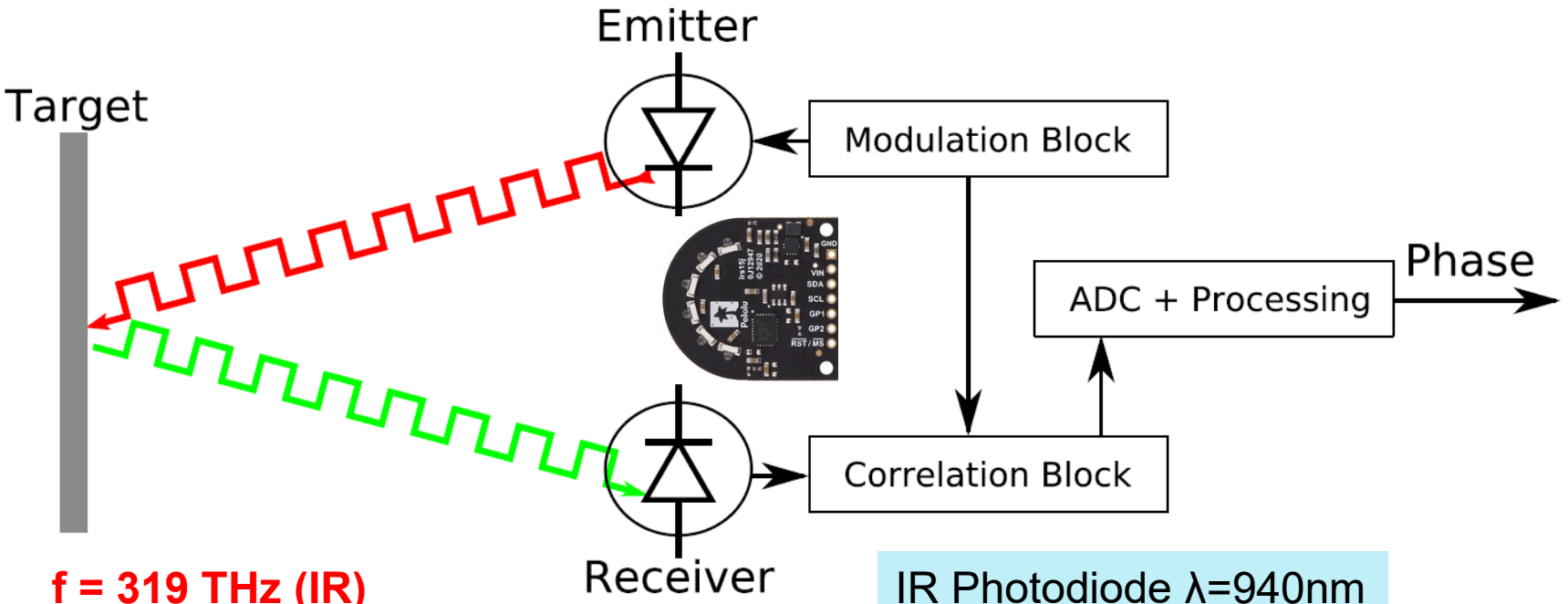


Time of Flight

- **d** = 500 mm from robot to wall
- **Δt** =2**d**/**ν** = 3.34ns

Time of Flight

Infrared (IR) Emitter λ=940nm

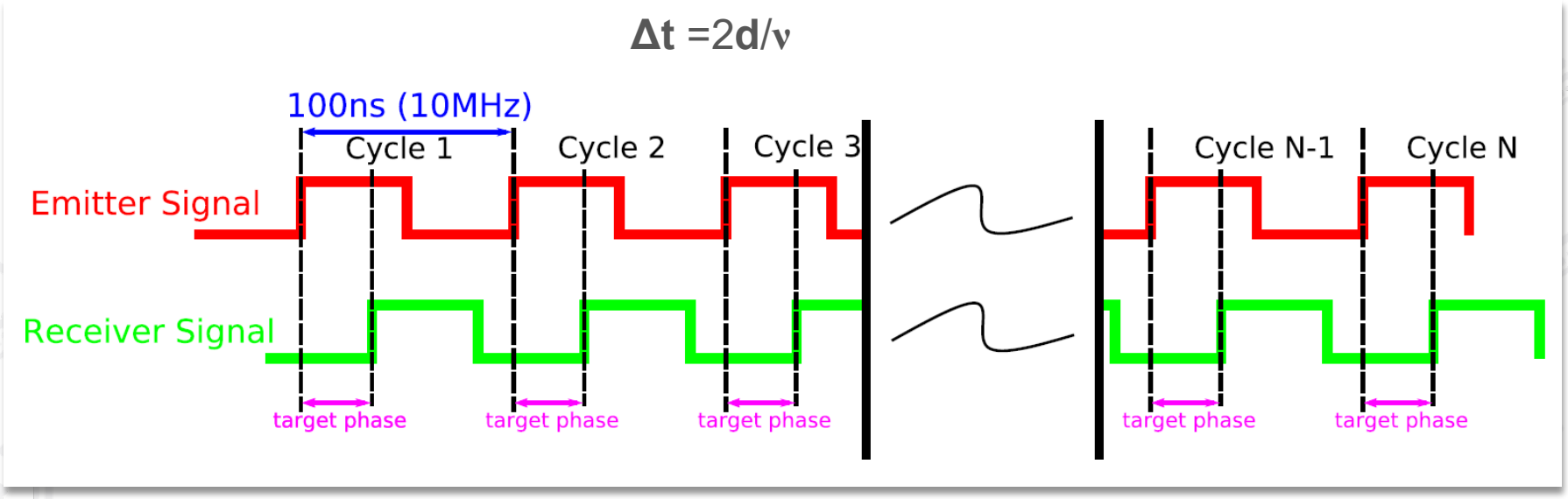


f = 319 THz (IR)
Modulated at 100 MHz

IR Photodiode λ=940nm

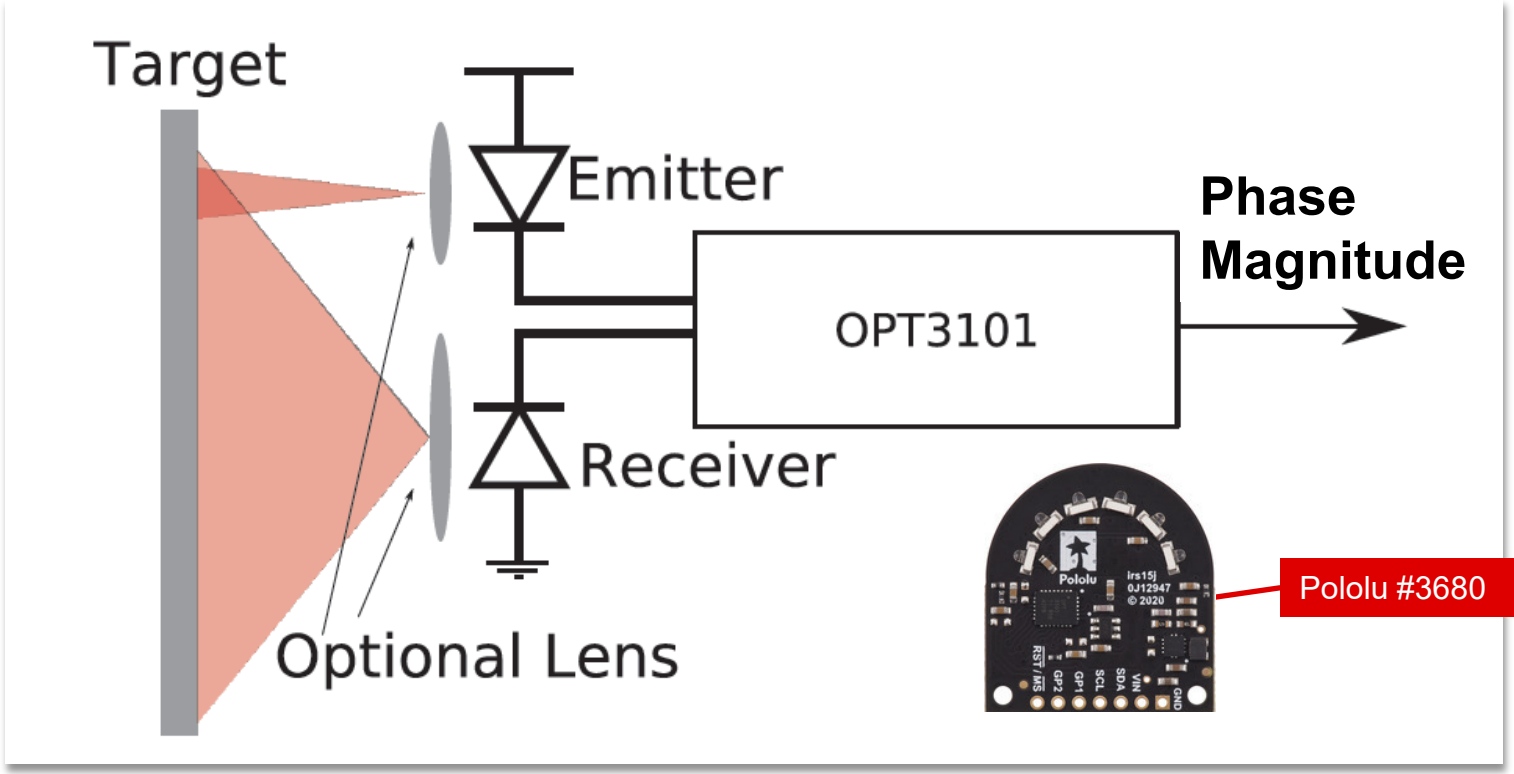
Pololu #3680

OPT3101 Measures Phase

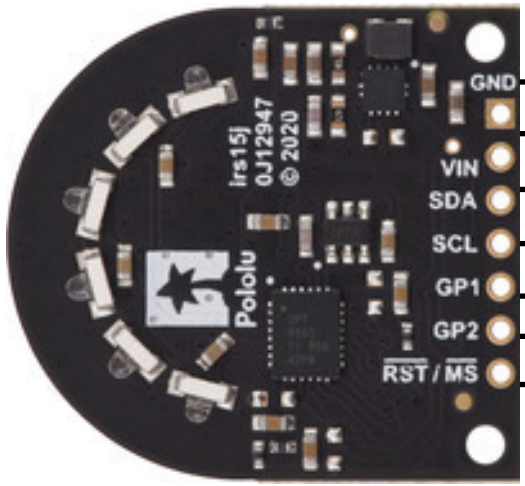


Phase is independent of amplitude of receiver signal
Needs averaging (recall CLT)
Magnitude is a measure of amount of received light

OPT3101 Analog Front End (AFE)



OPT3101 TI-RSLK MAX Interface



MSP432

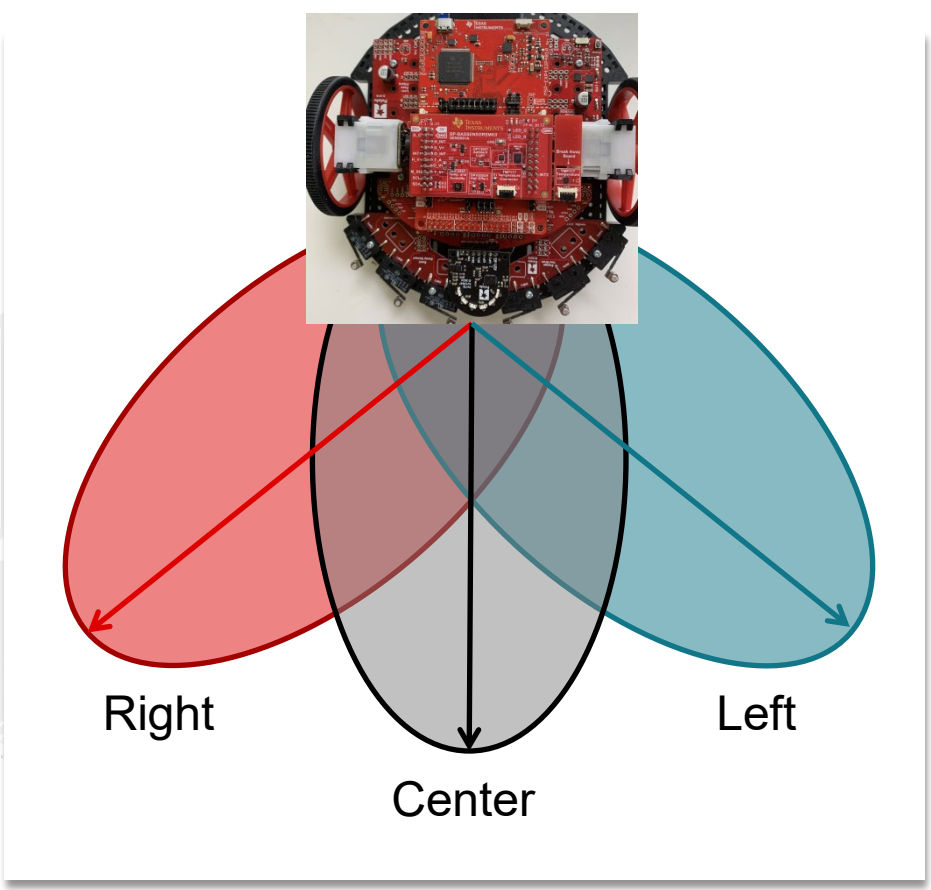
Ground
+3.3V
P6.4 (I2C SDA)
P6.5 (I2C SCL)
P6.2 (interrupt input)
P10.1 (not used)
P6.3 (GPIO output)

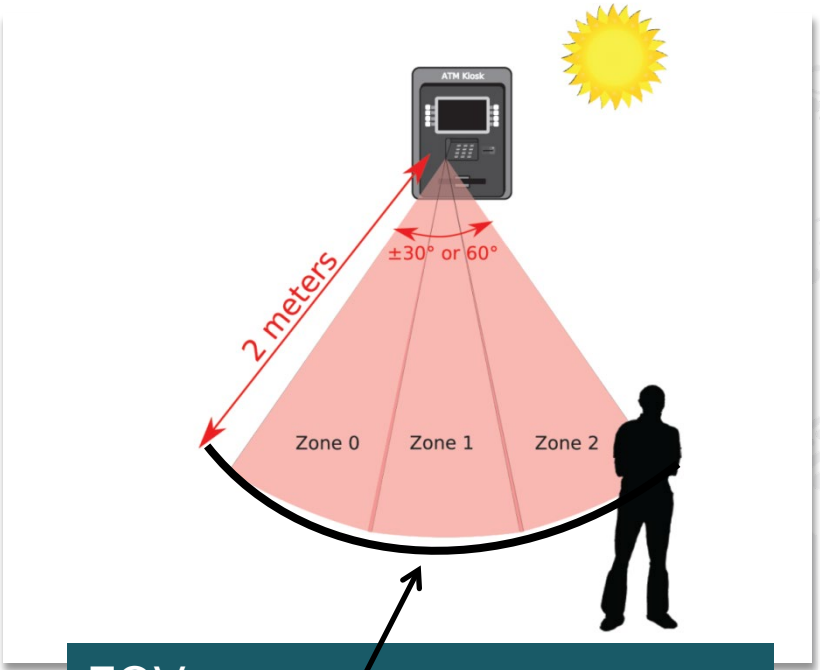
Pololu #3680

RSLK

Pololu OPT3101 uses 10k pullups

Measure Field of View in Lab 21





FOV

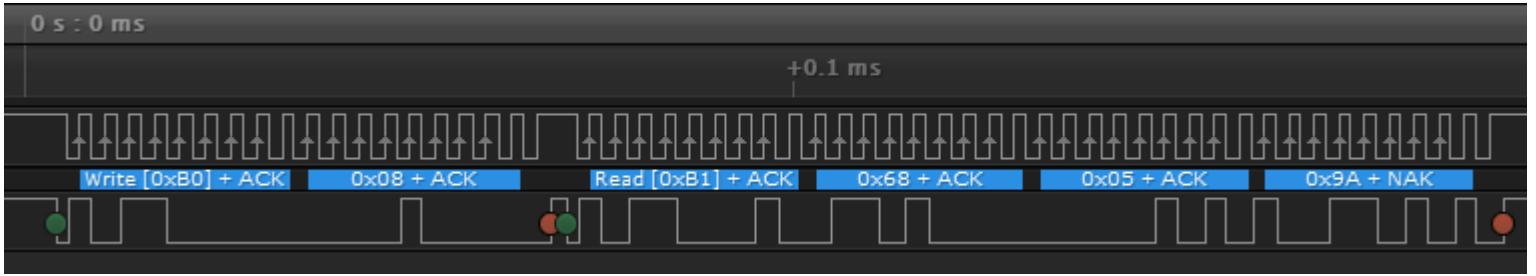
Consider an arc of constant distance

Let M be the max amplitude along arc

FOV is area with amplitude > ½ M

I2C Example: Master Receives Six Data Bytes from OPT3101

void OPT3101_ReadMeasurement(void); Address=0x58



https://www.saleae.com/

Bandwidth = 6 bytes/400us

OPT3101 Data Unpacking

```
void OPT3101_ReadMeasurement(void){
  reg08 = OPT3101_ReadRegister(0x08);
  reg09 = OPT3101_ReadRegister(0x09);
}
uint32_t OPT3101_GetPhase(void){
  return reg08 & 0xFFFF;
}
uint32_t OPT3101_GetDistanceMillimeters(void){
  return (OPT3101_GetPhase()*14990)>>16;
}
int32_t Left(int32_t left){
  return (1247*left)/2048 + 22;
}
uint32_t MyExample(void){ uint32_t distance;
  OPT3101_ReadMeasurement();
  distance  = OPT3101_GetDistanceMillimeters();
  return (Left(distance));
}
```

Pololu

Lab 21

Chan
0 is Left
1 is Center
2 is Right

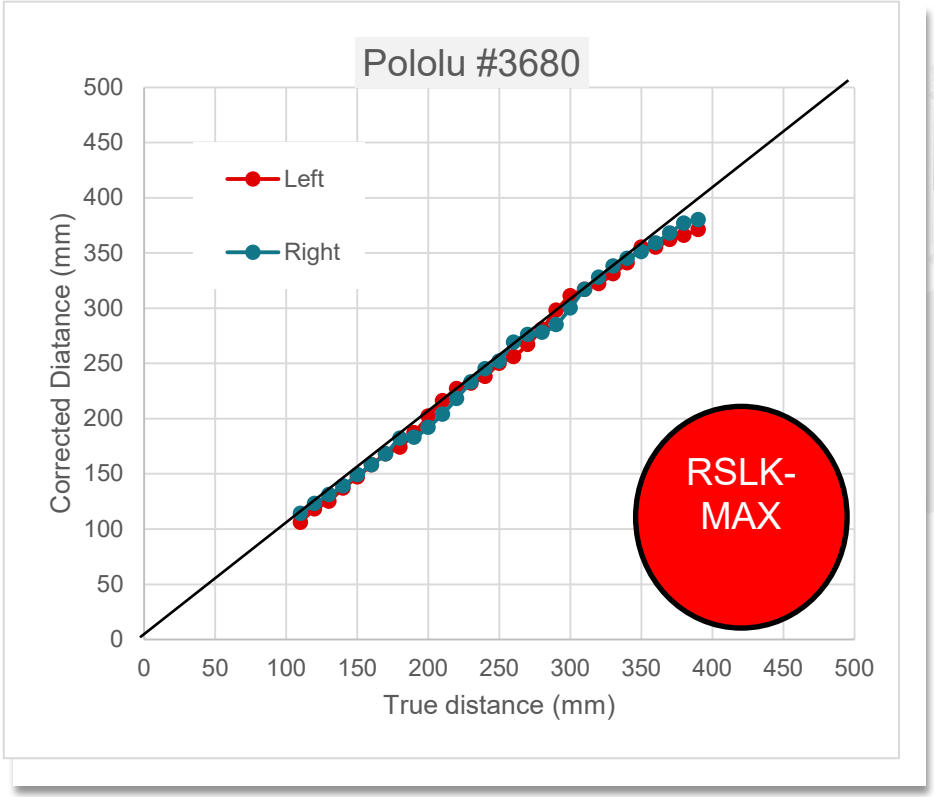
Summary

OPT3101

- Time of flight
  d = Δt * v / 2
- Modulated pulse output
- Analog front end
- Correlation to measure phase
- Three fields of view (FOV)
  Left, Center, Right

Practical Aspects

- I2C speed is slower than 400 kHz
- 30 Hz (10 Hz per channel)
- Works well up to 350 mm
- Software overhead 400 µs in ISR
- Requires user calibration



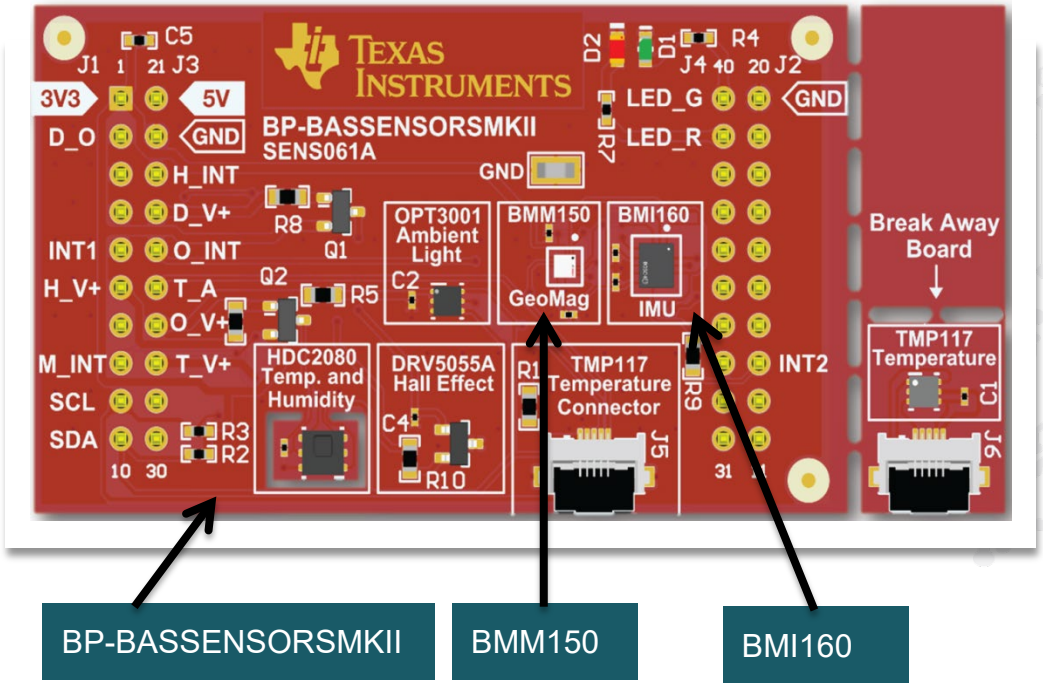
Module 21

Lecture: Sensor Integration – BMI160 / BMM150 Inertial Measurement Unit

BMI160/BMM150 Inertial Measurement Unit (IMU)

You will learn in this module

- BMI160
  - 16-bit triaxial accelerometer
  - 16-bit triaxial gyroscope
  - Step counter
  - I2C interface
- BMM150 Geomagnetic Sensor
  - 16-bit triaxial magnetic field
  - I2C interface slave to BMI160



Bosch BST-BMI160-DS0001-08
Bosch BST-BMM150-DS001-04

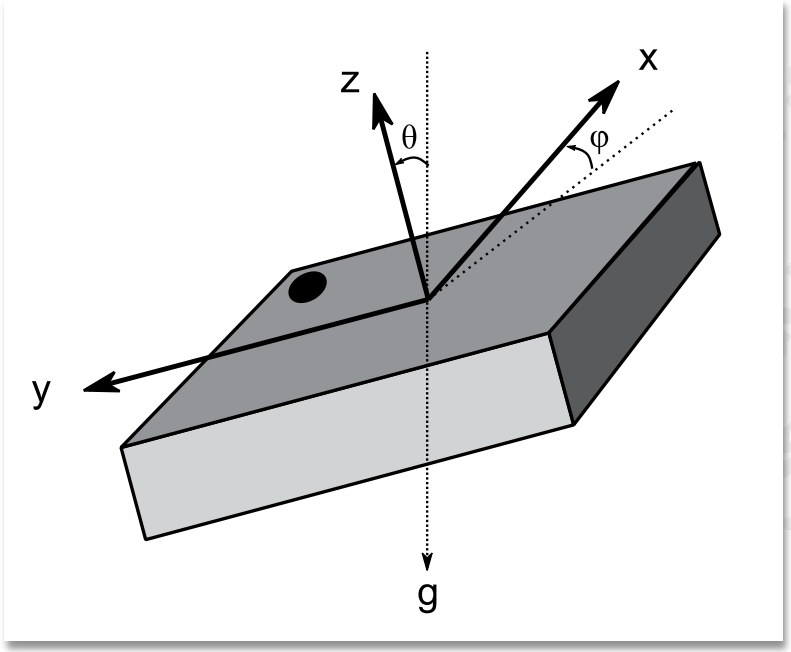
TI BASSENSORSMKII uses 10k pullups

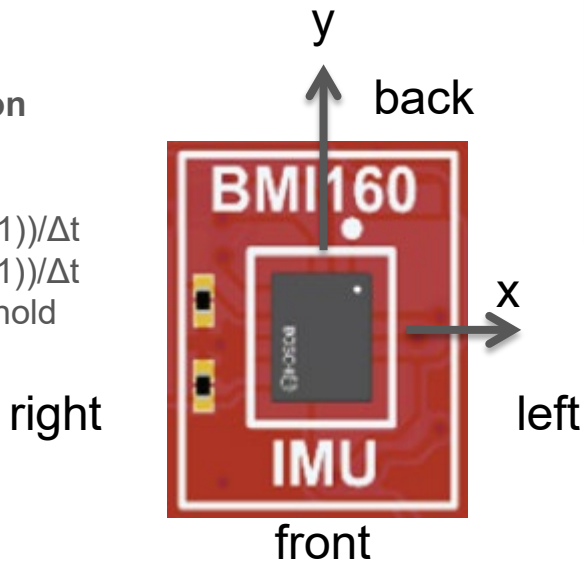
https://www.ti.com/tool/BP-BASSENSORSMKII

Physics: Accelerometer

Acceleration, position x, y, z

- **Motion F = ma**
  $a_x = d^2x/dt^2$
  $a_y = d^2y/dt^2$
  $a_z = d^2z/dt^2$
- **Orientation** relative to the center of the earth
  $a_x = g * \sin(\theta) * \cos(\varphi)$
  $a_y = -g * \sin(\theta) * \sin(\varphi)$
  $a_z = g * \cos(\theta)$
  $a_y / a_x = -\tan(\varphi)$
- **Collision Detection**
  Jerk $j = da/dt$
  If moving, every Δt
  $j_x = (a_x(n) - a_x(n-1))/\Delta t$
  $j_y = (a_y(n) - a_y(n-1))/\Delta t$
  $(j_x^2 + j_y^2) >$ threshold
  $\Phi = \arctan(j_y / j_x)$

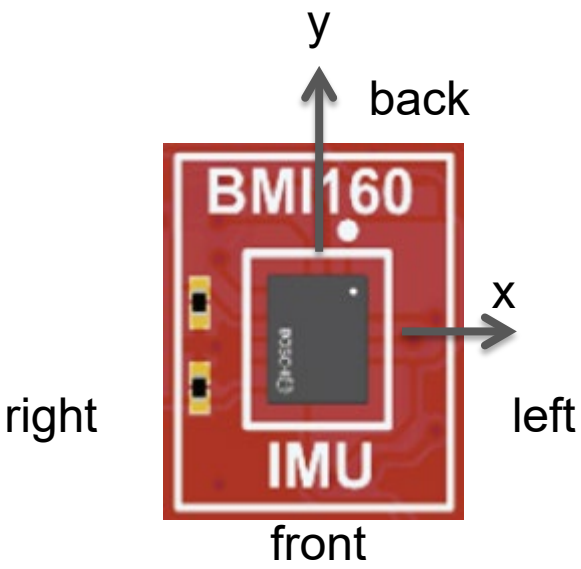


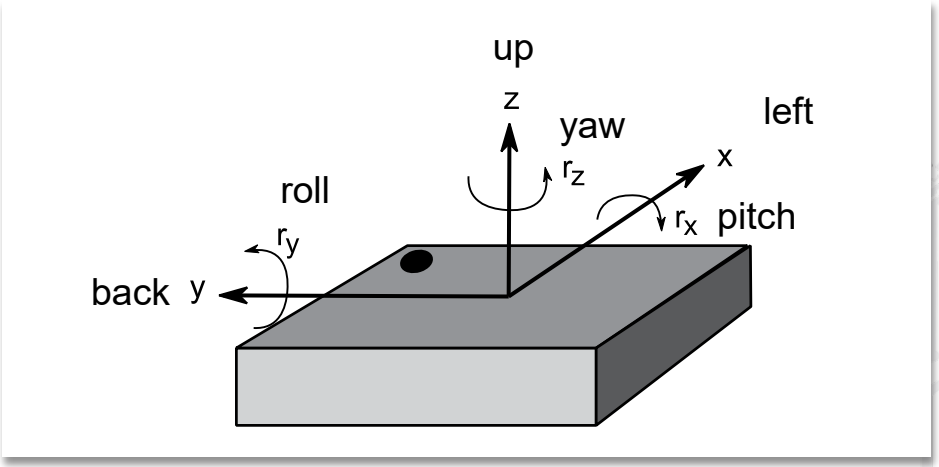


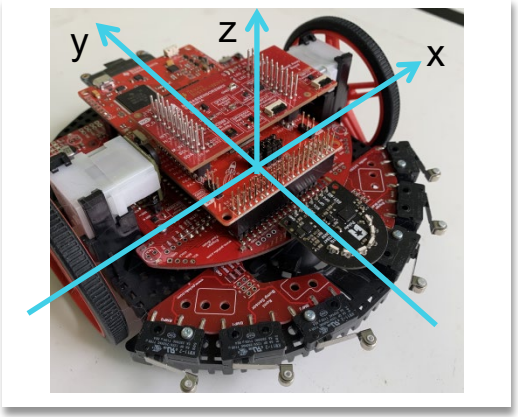
Physics: Gyroscope

Rotation: yaw, pitch, roll

- r_x degrees/sec, pitch
- r_y degrees/sec, roll
- r_z degrees/sec, yaw







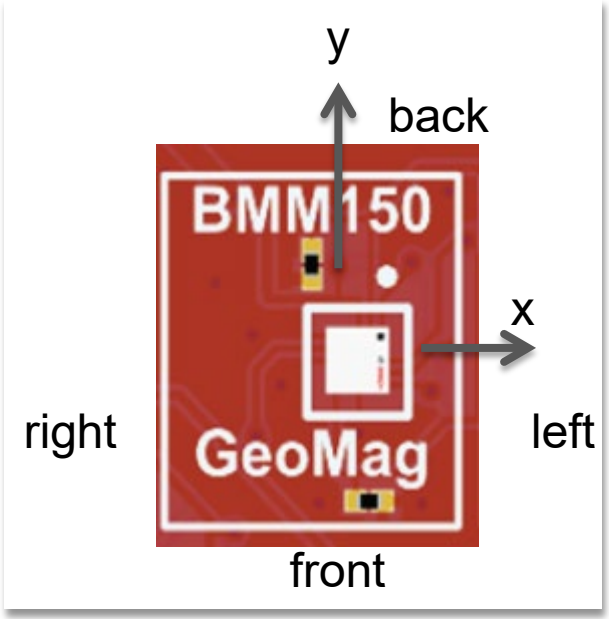
Physics Geomagnetic Sensor

Magnetic Field, direction to magnetic North

- Measures **B** field
- $\mathbf{m_x}$, $\mathbf{m_y}$ ±1300 µT
- $\mathbf{m_y}$ ±2500 µT
- Resolution 0.3 µT
- Needs calibration when placed on the robot

FlipCore measurement principle

- Magnetic layer of just a few millionth of a millimeter thickness is magnetically reversed periodically, generating a pulse
- Temporal distance between applied and detected signal depends on the strength of the operating terrestrial magnetic field

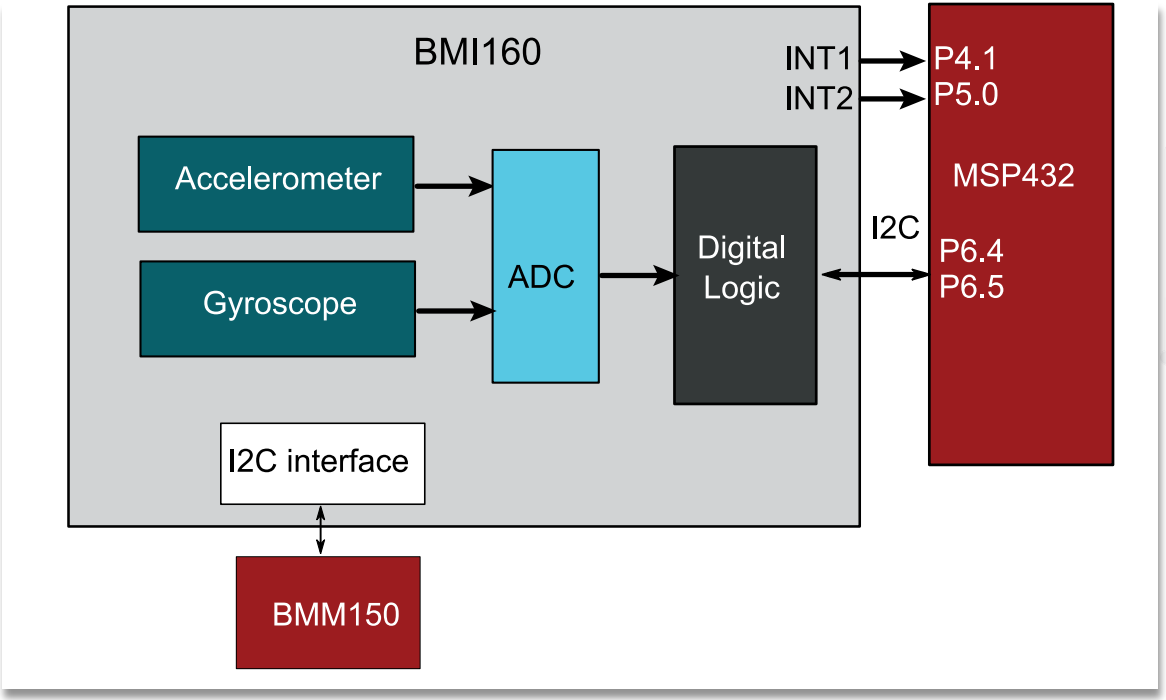


BMI160 Inertial Measurement Unit

Properties

- Mixed analog-digital IC
  Sensors
  Analog circuits
  ADC
  Digital signal processing
  I2C interface
- Speed-noise tradeoff
- Averaging (CLT)
- 117 registers inside chip
- Step count
- High-g detection
- FIFO queue for streaming
- Hardware interrupts
  INT1 can be used
  INT2 conflict with ERB

Address = 0x69

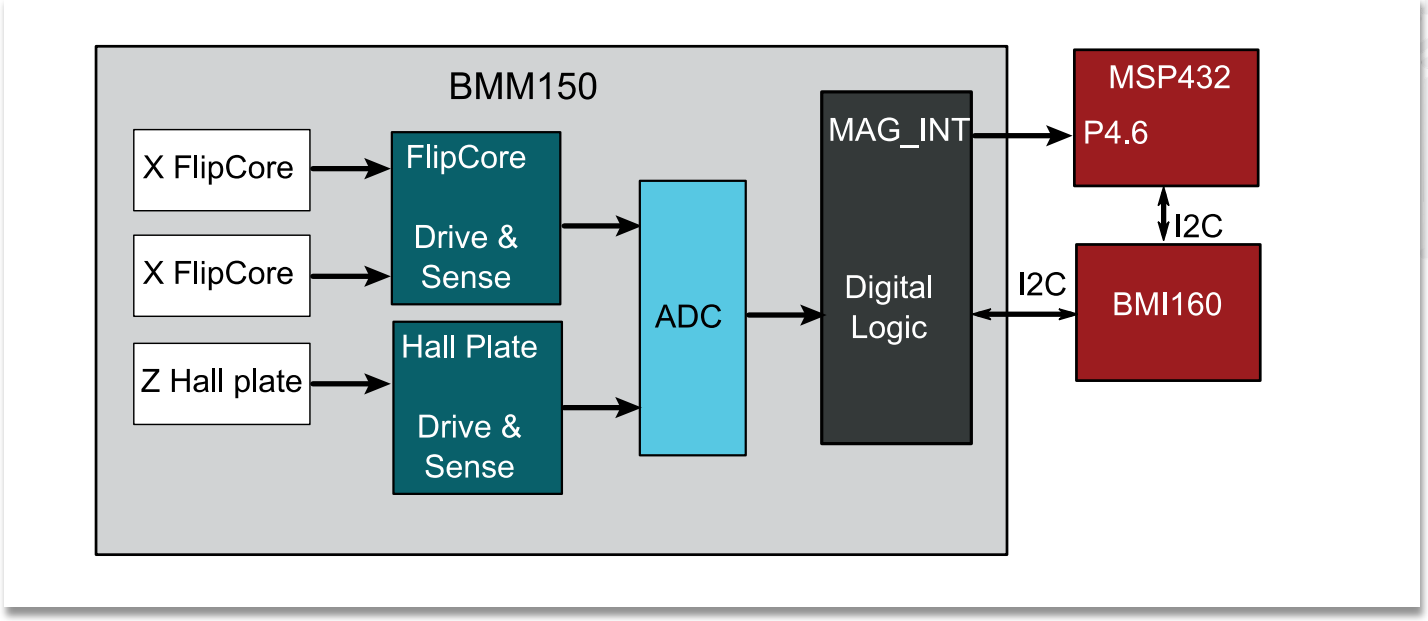


BMM150 Geomagnetic Sensor

Properties

- Mixed analog-digital IC
- Speed-noise tradeoff, output data rate (ODR)
- 18 registers inside chip
- MAG_INT conflicts with bumper switch 4

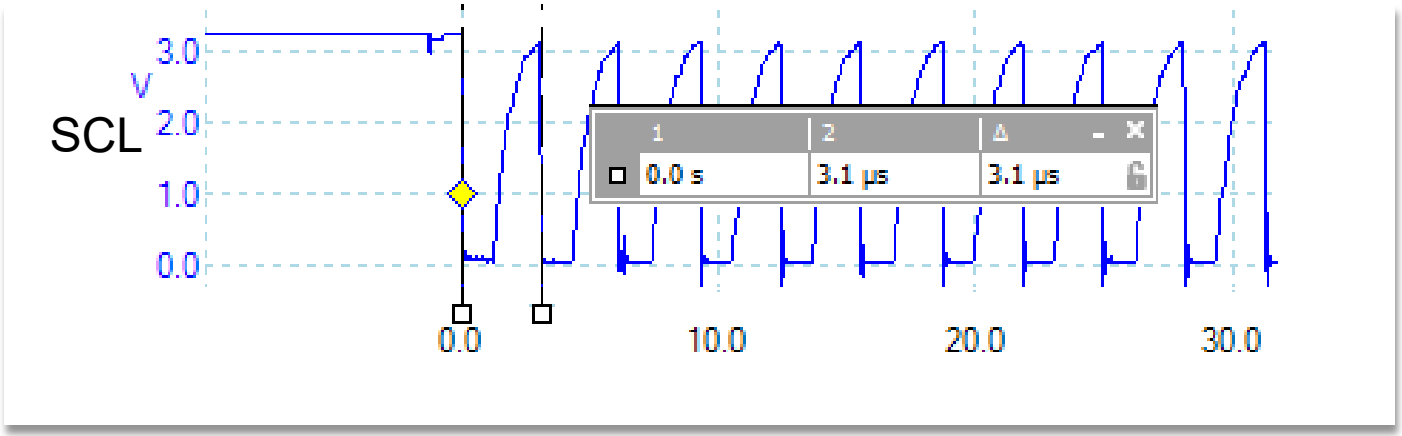
Address = 0x13



I2C protocol

I2C signals

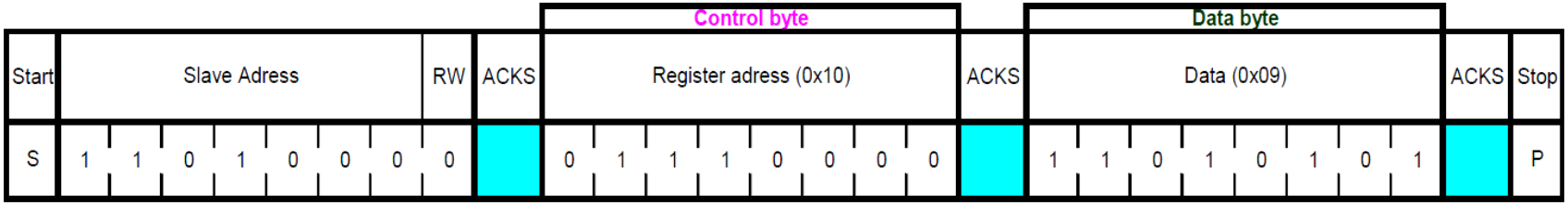
- Active low
- **Passive high**
- Master waits for clock to go high



Software set bit rate to 400 kHz

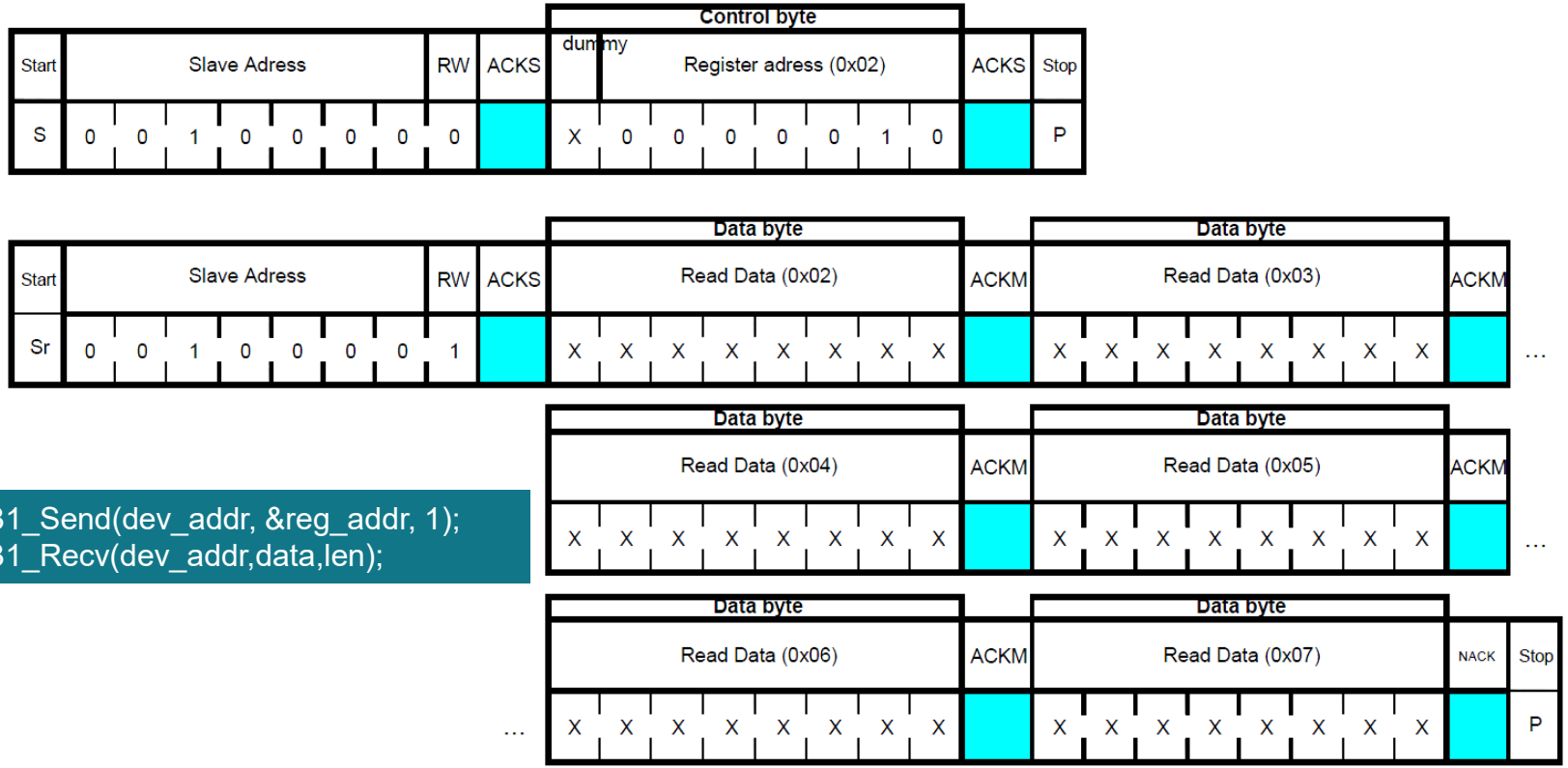
Actual bit rate is 323 kHz

I2C protocol write data




```
I2CB1_Send2(dev_addr, reg_addr, data[0]); // 1-byte
I2CB1_Send3(dev_addr, reg_addr, data); // 2 bytes
I2CB1_Send4(dev_addr, reg_addr, data); // 3 bytes
```

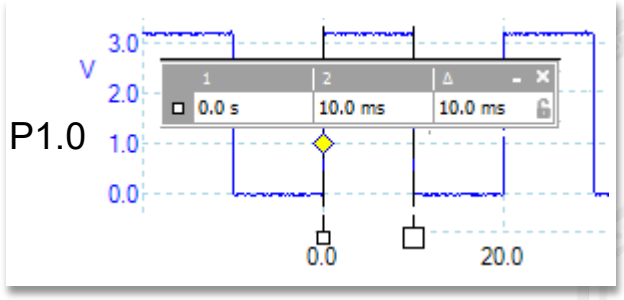
I2C protocol read data



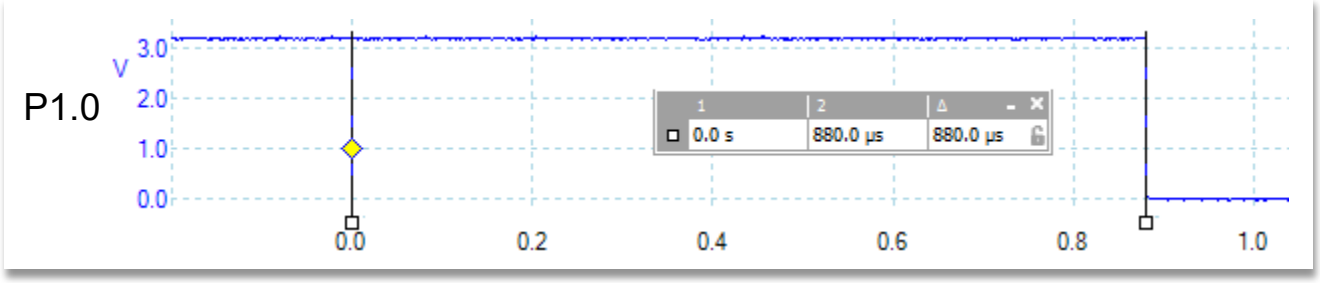

```
I2CB1_Send(dev_addr, &reg_addr, 1);
I2CB1_Recv(dev_addr,data,len);
```

BMI160/BMM150 Read measurement

```
void Background_ISR(void){ // 100 Hz real-time sampling
    P1->OUT ^= 0x01;        // profile
    P1->OUT ^= 0x01;        // profile
    bmi.fifo->length = FIFO_SIZE;
    uint8_t aux_inst = 1, gyr_inst = 1, acc_inst = 1;
    bmi160_extract_aux(&aux_data, &aux_inst, &bmi);
    bmi160_extract_gyro(&gyro_data, &gyr_inst, &bmi);
    bmi160_extract_accel(&accel_data, &acc_inst, &bmi);
    bmm150_aux_mag_data(&aux_data.data[0], &bmm);
    mag_data = bmm.data;
    semaphore++;
    P1->OUT ^= 0x01;        // profile
}
```



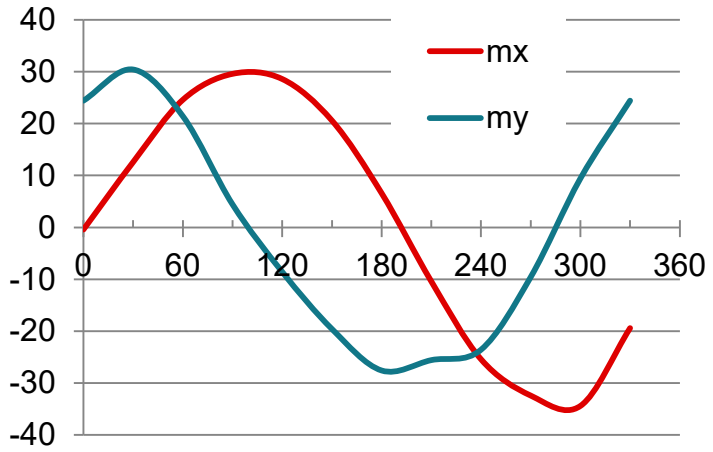
8.8% overhead

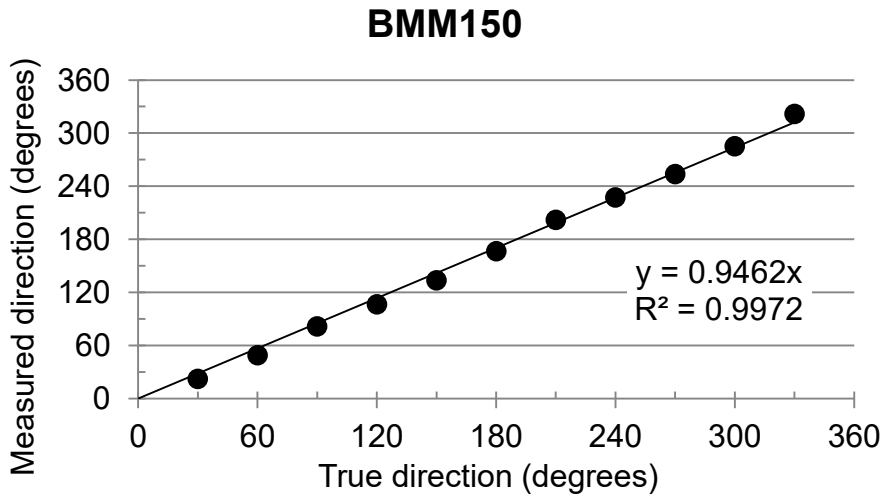


BMM150 Calibration

```
void Direction(void){
   mx = mag_data.x+120;
   my = mag_data.y+120;
}
```

- RSLK motors create offset
- Very noisy with motors running
  Needs digital filtering
- θ = 180*atan(mx/my)/π
  if my<0 then add +180

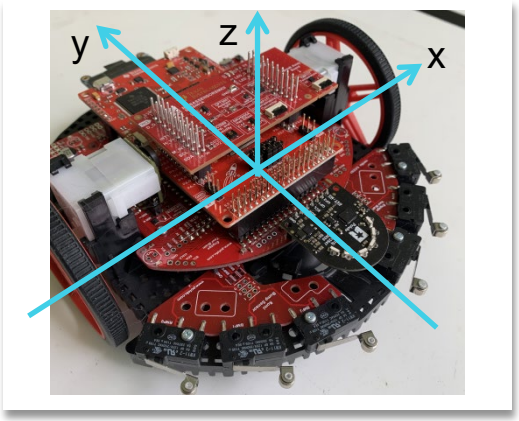




Summary

BMI160/BMM150

- 9-axis IMU
  Acceleration
  Yaw, pitch, roll
  Magnetic field
- Mixed analog-digital IC
  Sensor
  Analog circuits
  ADC
  Digital signal processing



Practical Aspects

- I2C speed is slower than 400 kHz
- 100 Hz sensor limitation
- Software overhead 880 µs in ISR
- Requires user calibration

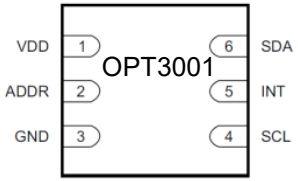
Module 21

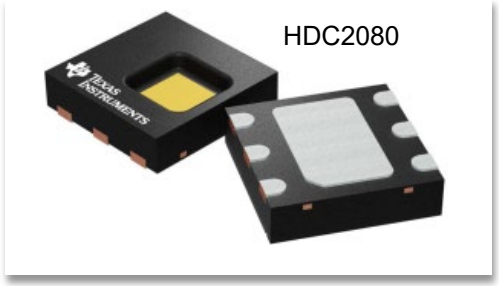
Lecture: Sensor Integration – Environmental Sensors

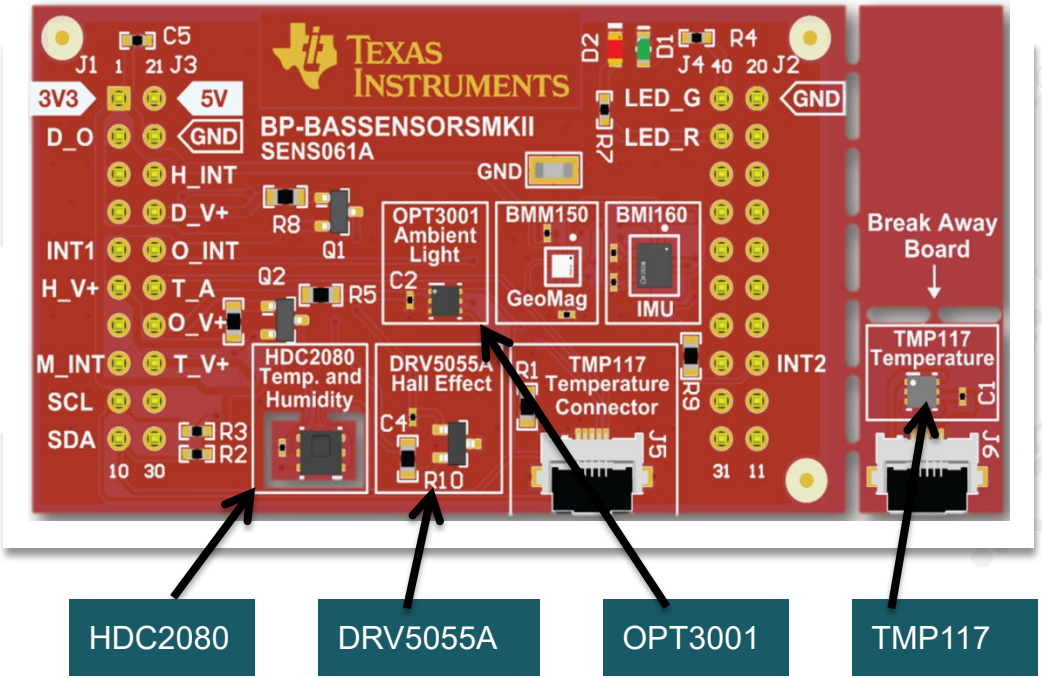
Environmental Sensors

You will learn in this module

- TMP117
  - Temperature
- HDC2080
  - Temperature
  - Humidity
- OPT3001
  - Light intensity
- DRV5055A
  - Hall Effect







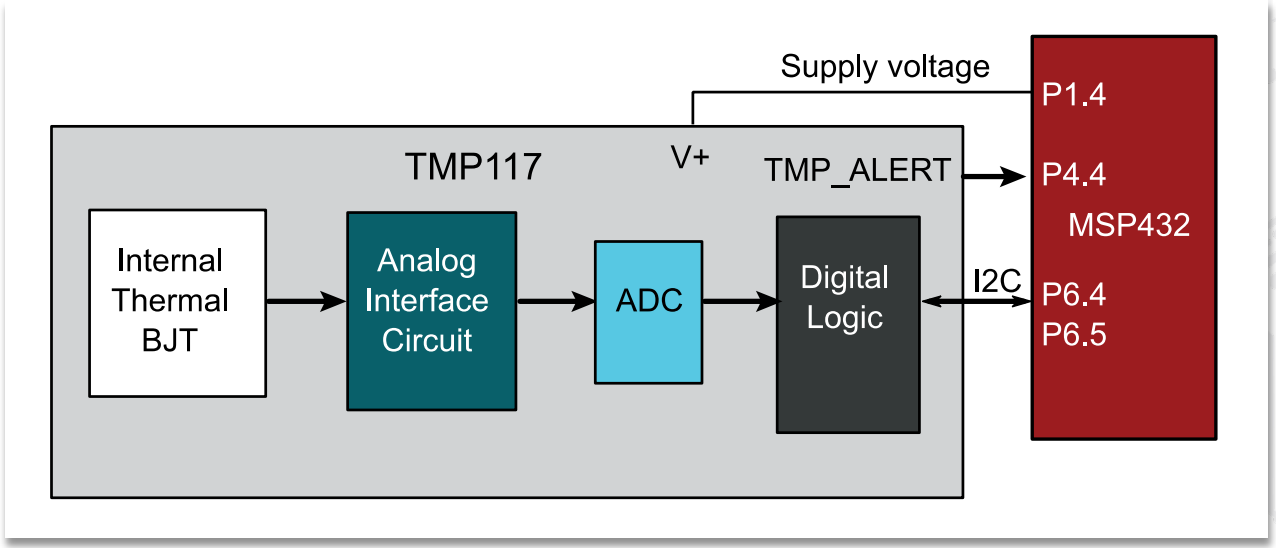
TI TMP117, HDC2080, DRV5055A, and OPT3001 data sheets are in the datasheets folder of RSLK software installation

TI BP- BASSENSORSMKII uses 10k pullups

TMP117 Digital Temperature Sensor

Properties

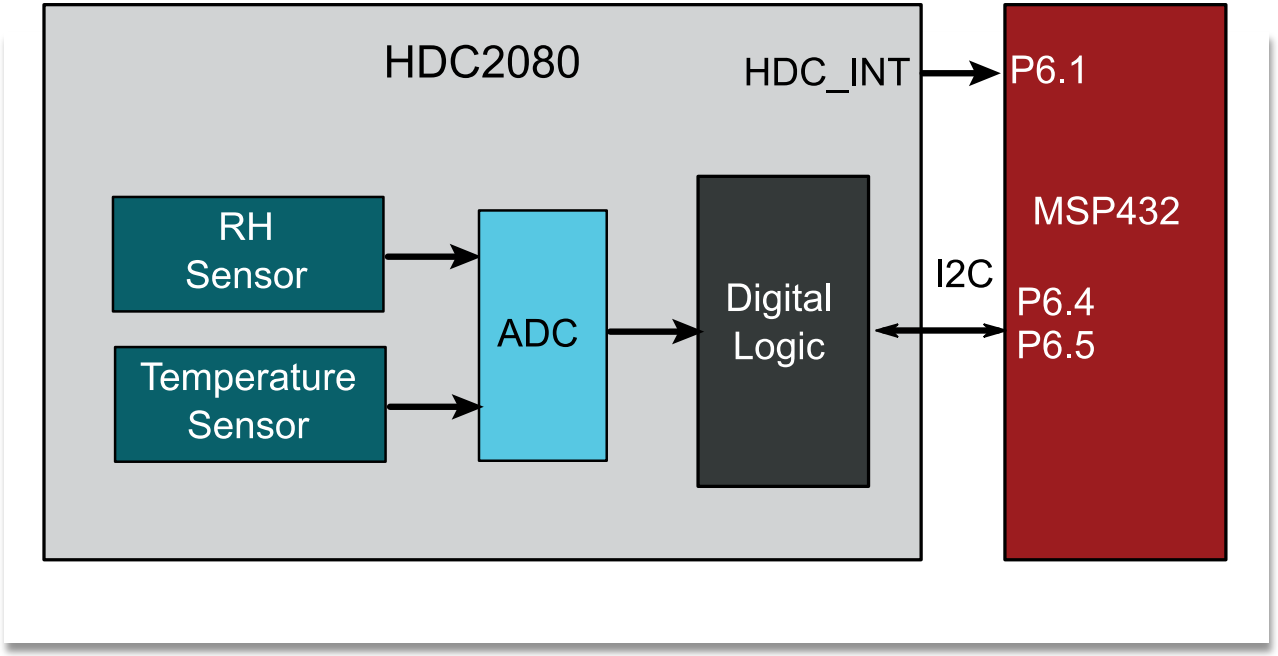
- Mixed analog-digital IC
  Sensor, Bipolar Junction Transistor
  Analog circuits
  ADC
  Digital signal processing
  I2C interface
- Range -20 to +50 °C
- Resolution 0.0078 °C
- Accuracy ±0.1°C
- 10 registers inside chip
- Powered by P1.4
  Same as BMP5
- TMP_ALERT = P4.4
  Same as TExaS scope



HDC2080 Low-Power Humidity and Temperature Digital Sensor

Properties

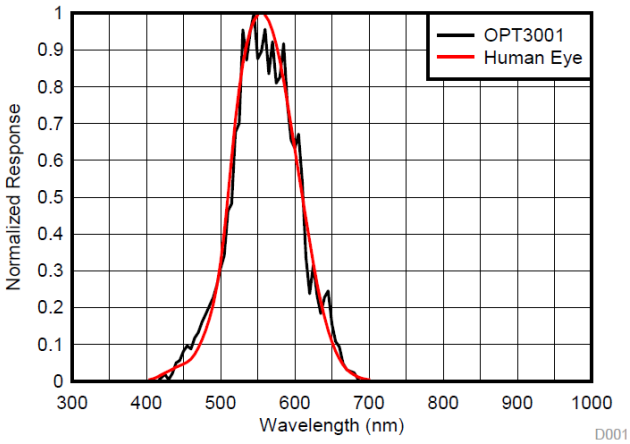
- Mixed analog-digital IC
- Adjustable sampling rate and resolution
- 20 registers inside chip
- Humidity
  Accuracy: ±3%
- Temperature
  Accuracy: ±0.4°C

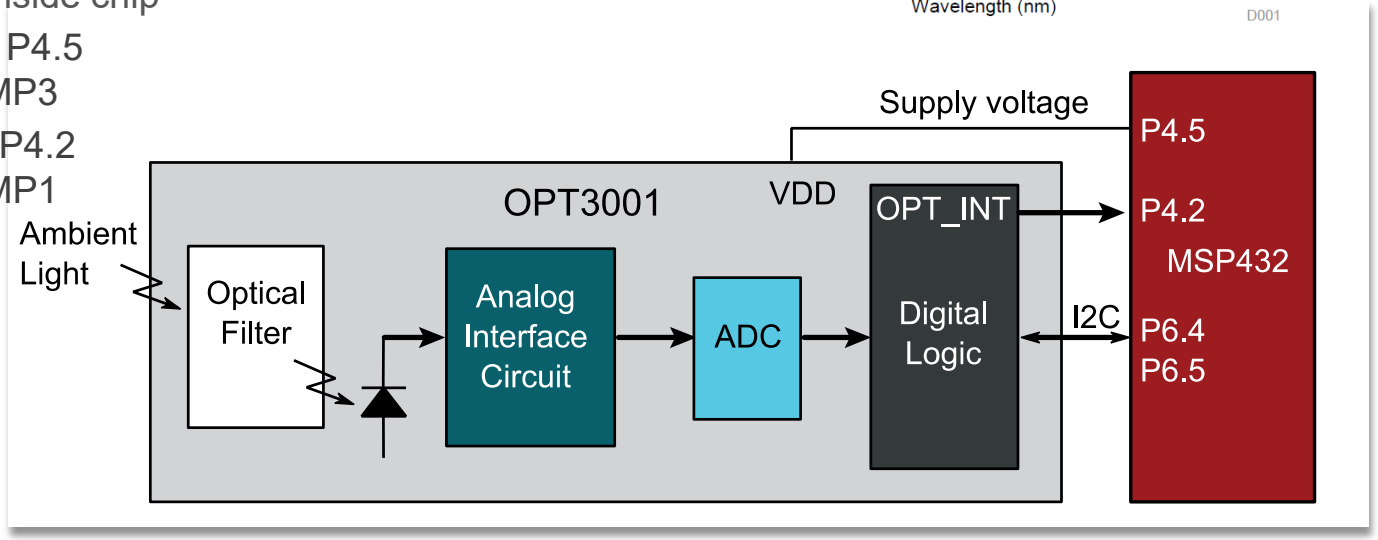


OPT3001 Ambient Light Sensor

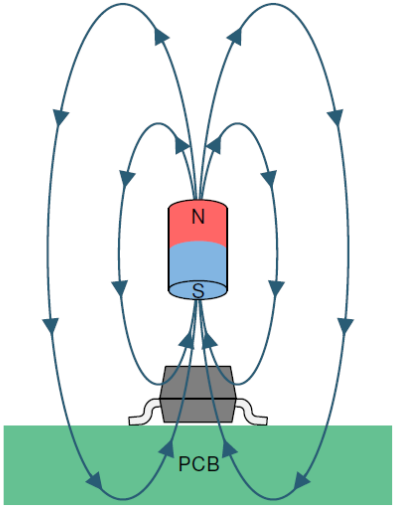
Properties

- Mixed analog-digital IC
- Spectral response matched to human eye
- Adjustable range/resolution
  Range 41 lux to 83 k lux
  Resolution 0.01 to 20 lux/bit
- 6 registers inside chip
- Powered by P4.5
  Same as BMP3
- OPT_INT = P4.2
  Same as BMP1

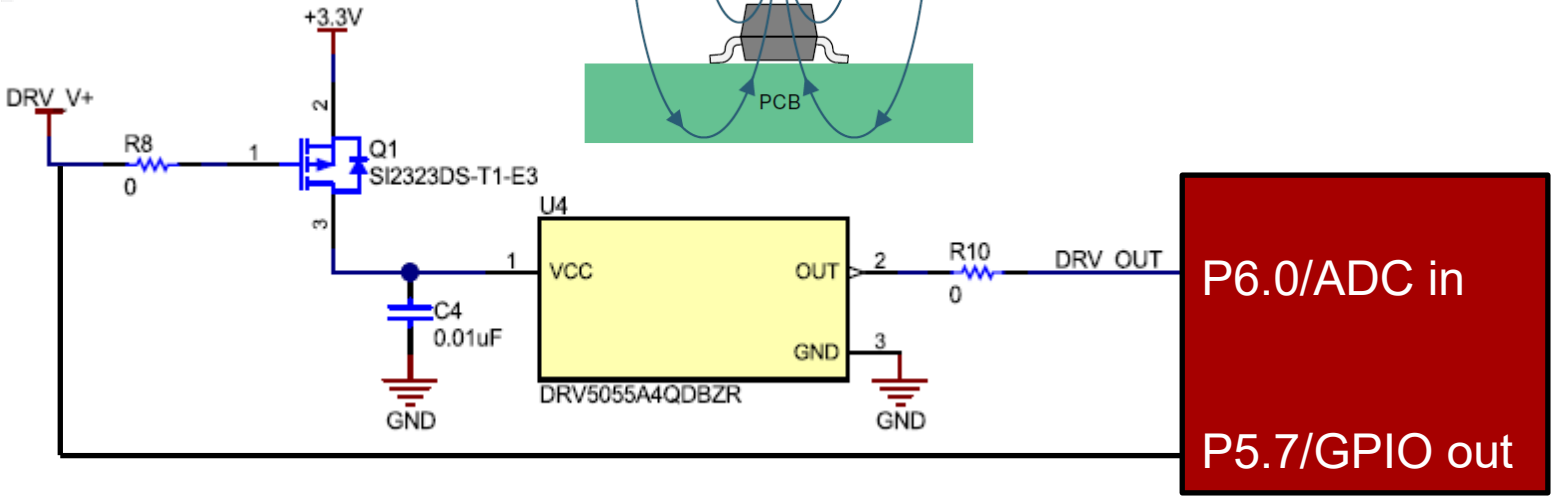




DRV5055 Hall Effect Sensor

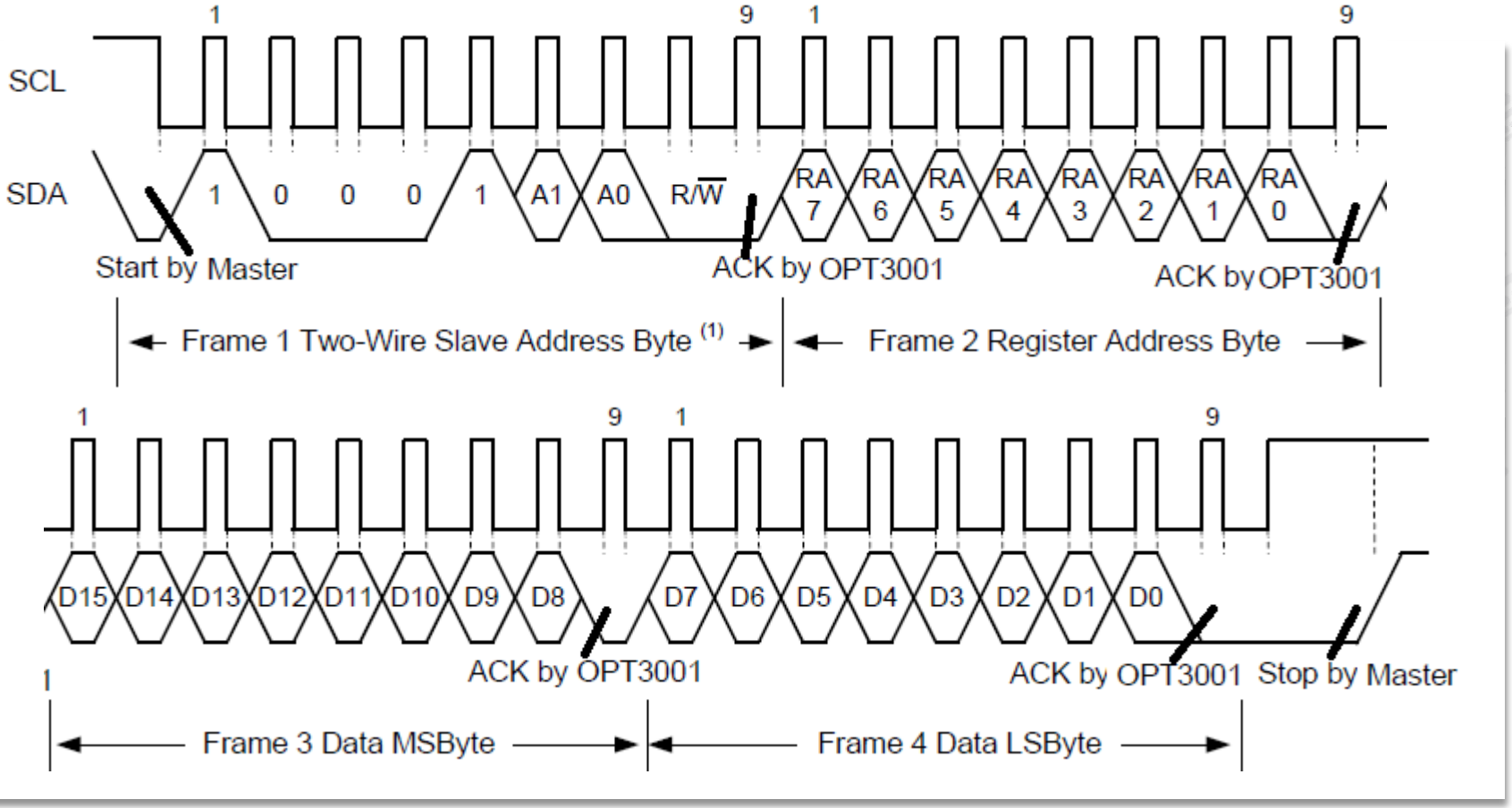


DRV_Out = 1.65±signal

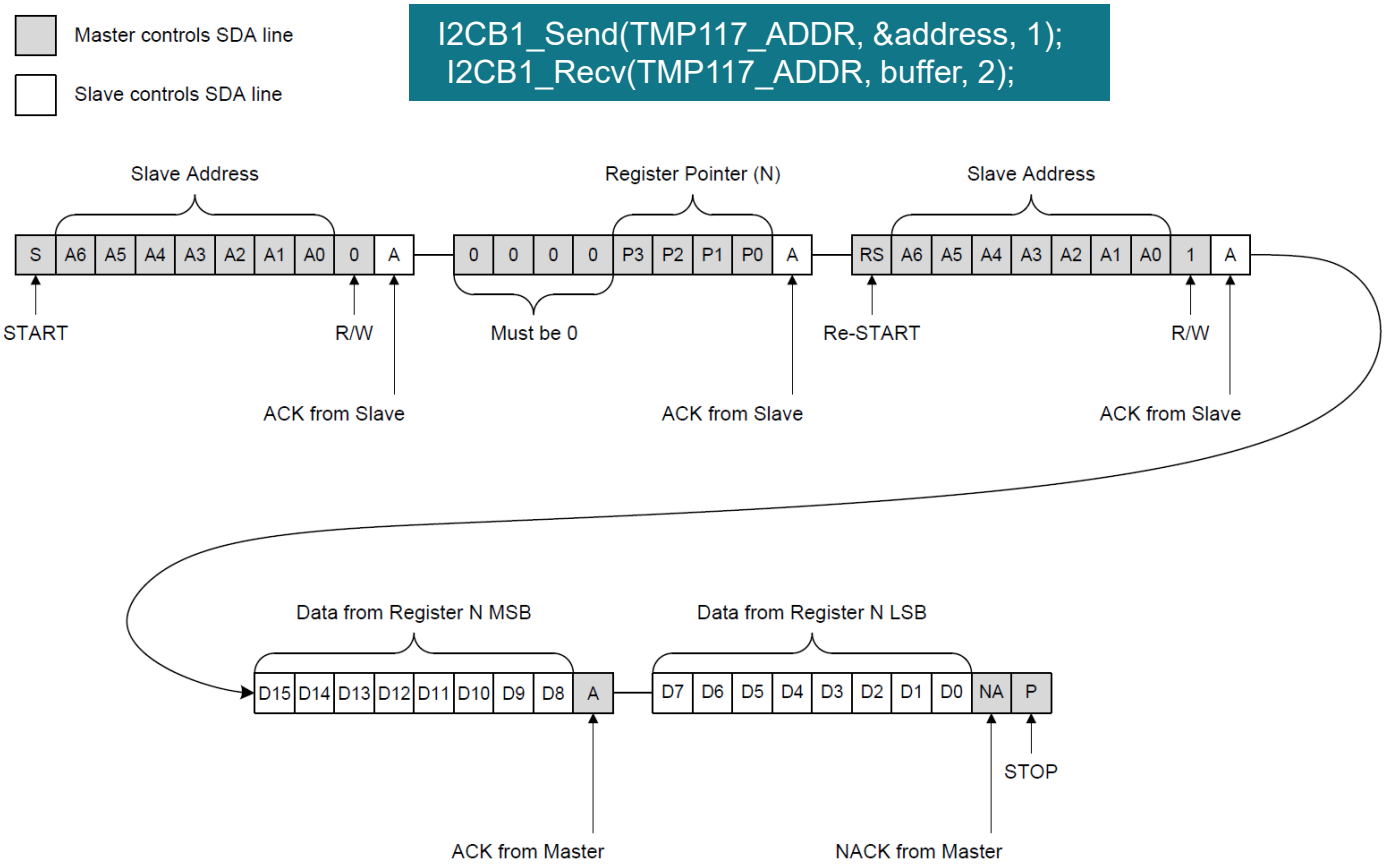


I2C Protocol Write Data

I2CB1_Send(OPT3001_ADDRESS, buffer, 3);



I2C Protocol Read Data



TMP117 Read Measurement

```
int32_t Temperature; // units are 0.1 C
int32_t Semaphore;
void Background(int16_t temperature){
  Temperature = (10*temperature)/128; // 0.1C
  Semaphore = 1;
}
void main(void){ Semaphore=0;
  DisableInterrupts();  Clock_Init48MHz();
  I2CB1_Init(30); // bit rate = 12MHz/30=400kHz
  TMP117_Arm(&Background);
  Init(); Clear();   OutString("TMP117 test\ninterrupt driven\n");
  SetCursor(0,2); OutString("T(C)= ");
  EnableInterrupts();
  while(1){
    WaitForInterrupt();
    if(Semaphore){ Semaphore=0;
      SetCursor(6,2); OutUFix1(Temperature);
    }
  }
}
```

Initialize clock

Initialize I2C

Initialize TMP117

Initialize display

TMP117 Read Measurement

```
int32_t Temperature; // units are 0.1 C
int32_t Semaphore;
void Background(int16_t temperature){
  Temperature = (10*temperature)/128; // 0.1C
  Semaphore = 1;
}
void main(void){ Semaphore=0;
  DisableInterrupts();  Clock_Init48MHz();
  I2CB1_Init(30); // bit rate = 12MHz/30=400kHz
  TMP117_Arm(&Background);
  Init(); Clear();   OutString("TMP117 test\ninterrupt driven\n");
  SetCursor(0,2); OutString("T(C)= ");
  EnableInterrupts();
  while(1){
    WaitForInterrupt();
    if(Semaphore){ Semaphore=0;
      SetCursor(6,2); OutUFix1(Temperature);
    }
  }
}
```

Interrupt driven:
Called when new temperature data is ready
Temperature units are 0.0078125 C

TMP117 Read Measurement

```
int32_t Temperature; // units are 0.1 C
int32_t Semaphore;
void Background(int16_t temperature){
  Temperature = (10*temperature)/128; // 0.1C
  Semaphore = 1;
}
void main(void){ Semaphore=0;
  DisableInterrupts();  Clock_Init48MHz();
  I2CB1_Init(30); // bit rate = 12MHz/30=400kHz
  TMP117_Arm(&Background);
  Init(); Clear();   OutString("TMP117 test\ninterrupt driven\n");
  SetCursor(0,2); OutString("T(C)= ");
  EnableInterrupts();
  while(1){
    WaitForInterrupt();
    if(Semaphore){ Semaphore=0;
      SetCursor(6,2); OutUFix1(Temperature);
    }
  }
}
```

Mailbox consists of data and semaphore

Sleep while waiting

Display output in main

HDC2080 Read Measurement

```
uint32_t Humidity;
int32_t Temperature;
void main(void){
  DisableInterrupts();
  Clock_Init48MHz();
  I2CB1_Init(30); // bit rate = 12MHz/30=400kHz
  HDC2080_Init();
  Init();  Clear();
  OutString("HDC2080 test\n");   SetCursor(0,2); OutString("H,T = ");
  while(1){
    Clock_Delay1ms(2000);
    Humidity = HDC2080_ReadHumidity();
    Temperature = HDC2080_ReadTemperature();
    SetCursor(5,1);
    UART0_OutUFix1(Humidity);
    OutChar(',');
    UART0_OutUFix1(Temperature);
  }
}
```

Initialize clock

Initialize I2C

Initialize HDC2080

Initialize display

Busy-wait synchronization

Display output in main

OPT3001 Read Measurement

```
OPT3001_Result Light; // lux = 0.01*(2^Exponent)*Result
void main(void){
  DisableInterrupts();
  Clock_Init48MHz();
  I2CB1_Init(30); // bit rate = 12MHz/30=400kHz
  OPT3001_Init();
  Init();   Clear();
  OutString("OPT3001 test\n");
  SetCursor(0,2); OutString("L = ");
  while(1){
    Clock_Delay1ms(2000);
    Light = OPT3001_ReadLight();
    SetCursor(4,1);
    OutUDec(Light.Exponent);
    OutChar(',');
    OutUDec(Light.Result);
  }
}
```

Initialize clock

Initialize I2C

Initialize OPT3001

Initialize display

Busy-wait synchronization

Display output in main

Summary

TMP117

- Range -20 to +50 °C
- Resolution 0.0078 °C
- Accuracy ±0.1°C
- Interrupt driven software

HDC2080

- Humidity Accuracy: ±3%
- Temperature Accuracy: ±0.4°C
- Busy-wait software

OPT3001

- Adjustable range/resolution
  Range 41 lux to 83 k lux
  Resolution 0.01 to 20 lux/bit

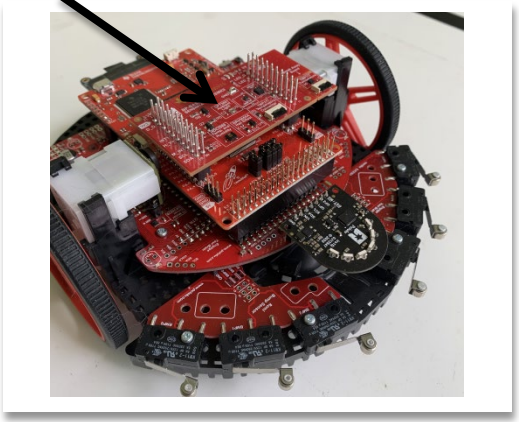
DRV5055A

- Hall Effect
- Analog output

Components In One IC

- Transducer
- Analog front end
- Analog to digital conversion
- Digital processing
- I2C communication

BP-BASSENSORSMKII



Conflicts with RSLK BMP1 BMP3 BMP5

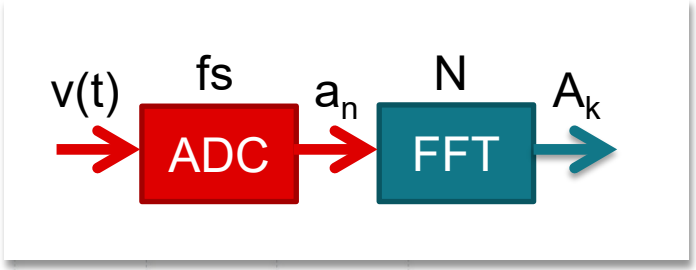
Module 21

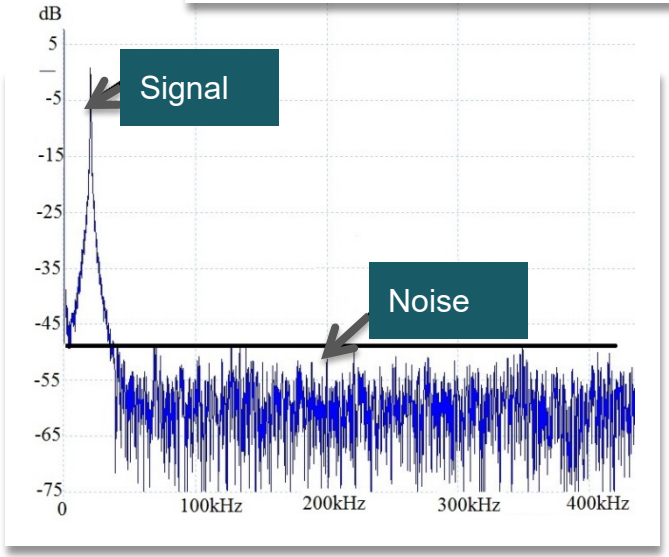
Lecture: Sensor Integration - Fast Fourier Transform

Fast Fourier Transform

You will learn in this module

- Discrete Fourier Transform
  - Conversion from time domain to frequency domain
  - Sampling rate, f_s
  - Number of samples, N (sample time $T = N/f_s$)
  - Windowing
- FFT implementation
  - Constants
  - N logN speed
  - Execution on MSP432
- FFT applications
  - Spectrum analyzer
  - Noise analysis





Sampling: conversion from physical to analog to digital

Sensor: physical to analog

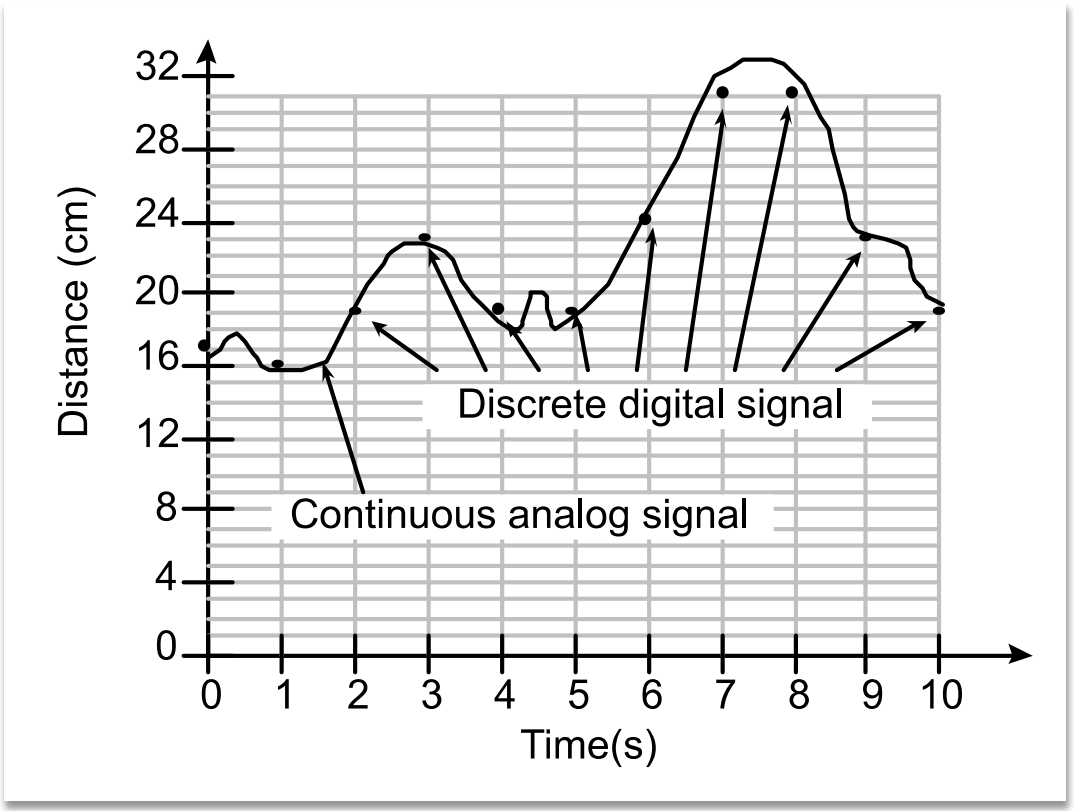
- Analog signal is voltage
- Analog signal is time

Amplitude limitations

- Range, min to max
- Resolution, Δx
- Precision, (max-min)/Δx

Time limitations

- Sampling rate, f_s
  0 to ½ f_s
- Number of samples
  Buffer size N
- Frequency resolution
  $\Delta f = f_s/N$



Transform: Time Domain to Frequency Domain

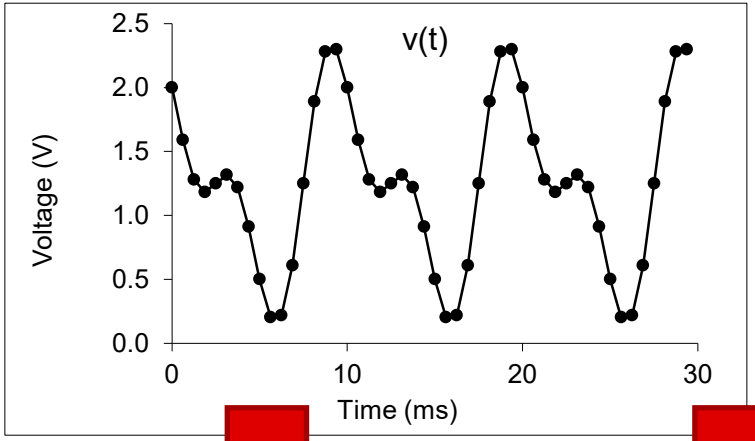
Let V(t) be time domain input

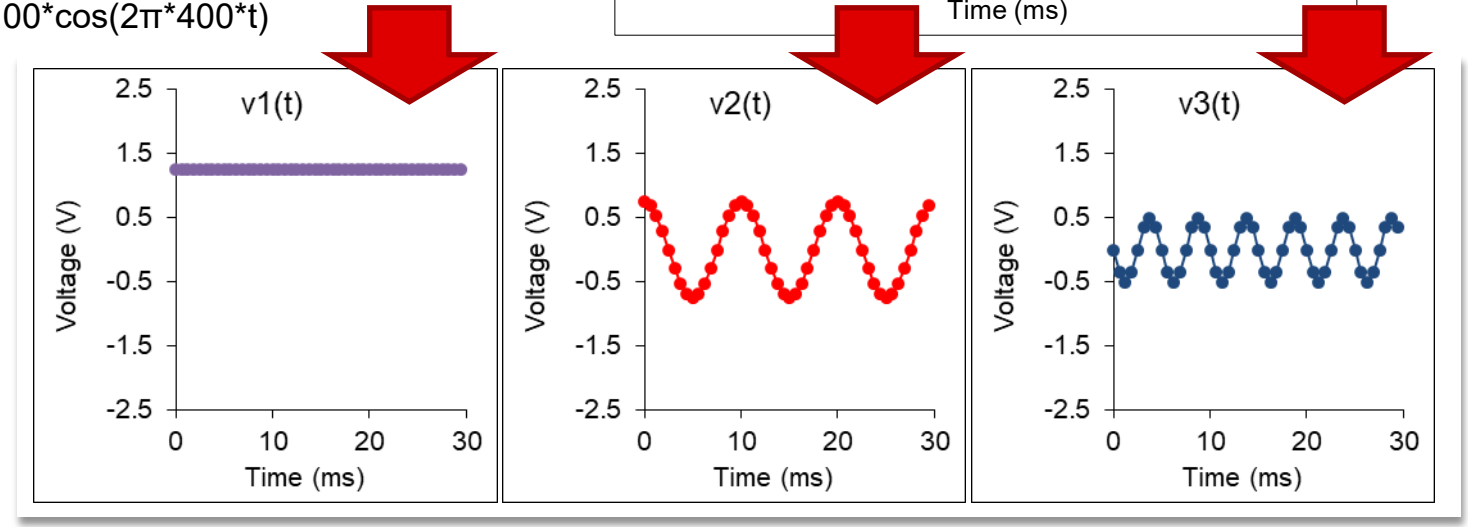
- Arbitrary periodic wave

Decompose into a sum of sinusoids

- V1 = 1.25
- V2 = 0.75*cos(2π*100*t)
- V3 = 0.50*cos(2π*200*t+ π/2)
- V4 = 0.00*cos(2π*300*t)
- V5 = 0.00*cos(2π*400*t)

…





Transform: Time Domain to Frequency Domain

Let v(t) be time domain input

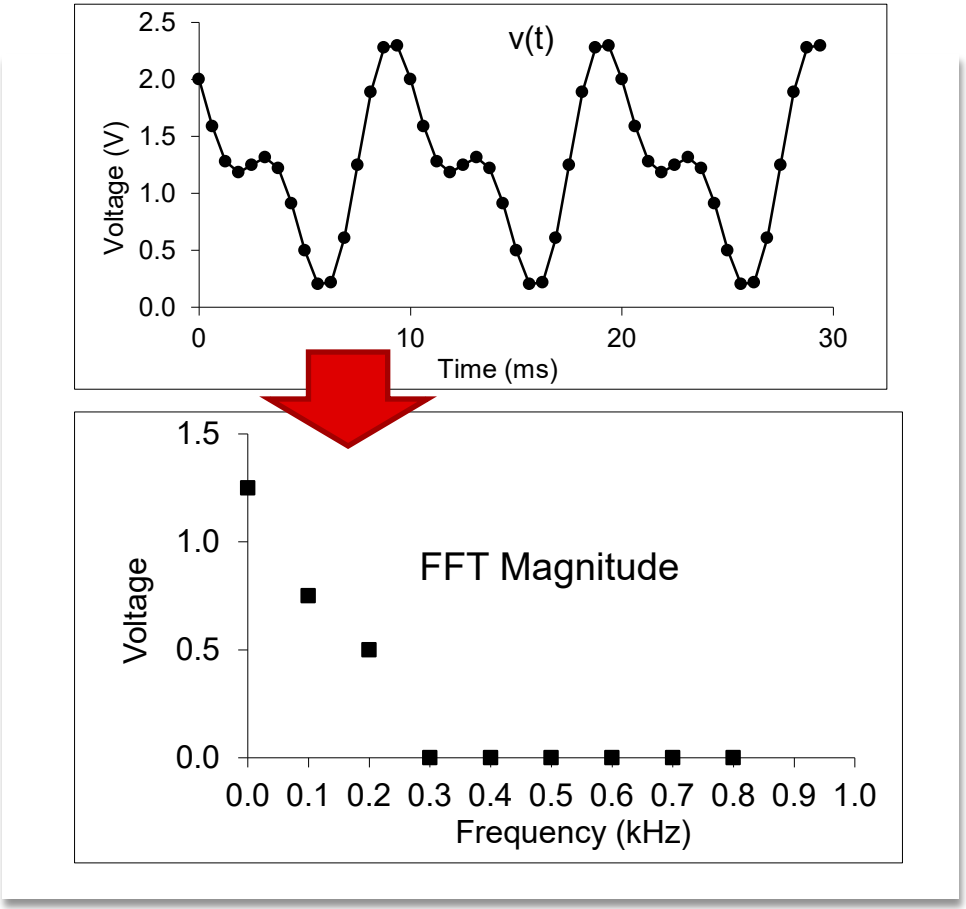
- Arbitrary periodic wave
- f_s = 1600 Hz
- N = 16

Decompose into a sum of sinusoids

- V1 = 1.25
- V2 = 0.75*cos(2π*100*t)
- V3 = 0.50*cos(2π*200*t+ π/2)
- v(t) = V1+V2+V3+…

Important design factors

- Sampling rate, f_s
  0 to ½ f_s
- Number of samples, buffer size N
- Frequency resolution
  $\Delta f = f_s/N$



Discrete Fourier Transform

Let V(t) be time domain input

- Arbitrary periodic wave, real
- f_s is sampling rate
- N is buffer size
- $\{a_n\} = \{a_0, a_1, a_2, \ldots, a_{N-1}\}$
- Index n is time (n/f_s)

Discrete Fourier Transform

- Calculates N complex values
- $\{A_k\} = \{A_0, A_1, A_2, \ldots, A_{N-1}\}$
- Index k is frequency ($k*f_s/N$)
- Frequency resolution
  $\Delta f = f_s/N$
- Order N*N in complexity

Fast Fourier Transform

- Same outputs $\{A_k\} = \{A_0, A_1, A_2, \ldots, A_{N-1}\}$
- Order N*logN in complexity

$$A_k = \sum_{n=0}^{N-1} a_n W_N^{kn}$$

$$W_N = e^{-j2\pi/N}$$

FFT on MSP432

Sample input

- $\{a_n\} = \{a_0, a_1, a_2, \ldots, a_{N-1}\}$ units volts
- Set real part of N inputs to signal
- Set imaginary parts to 0
- f_s is sampling rate
- Window if needed

Fast Fourier Transform

- Calculates N complex values
- $\{A_k\} = \{A_0, A_1, A_2, \ldots, A_{N-1}\}$

Interpret FFT

- Index k is frequency ($k*f_s/N$)
- Frequency resolution $\Delta f = f_s/N$
- DC Magnitude = $|A_0|/N$ in volts
- Results A_k, for k = 1 to ½ N
  Magnitude = $|A_k|/(N/2)$ in volts
  Phase = $\arctan(\mathrm{Im}(A_k)/\mathrm{Re}(A_k))$
- $dB_{FS} = 20 \log_{10}(\text{Magnitude/FullScale})$

```
#include "../inc/FFT.h"
#define q   11      // for 2^11 points
#define N   (1<<q)  // 2048-point FFT
complex_t a[N], scratch[N];
void Sample(void){
  for(int k=0; k<N; k++){
    a[k].Real = 3.3*ADC_In()/16384;
    a[k].Imag = 0;
    Sync(); // set sampling rate
  }
}
void SpectrumAnalyzer(void){
  Sample();   // set a[]
  fft(a, N, scratch); // result put in a[]
// plot a[]
}
```

For a more full featured approach see,
https://www.ti.com/tool/MSP-IQMATHLIB

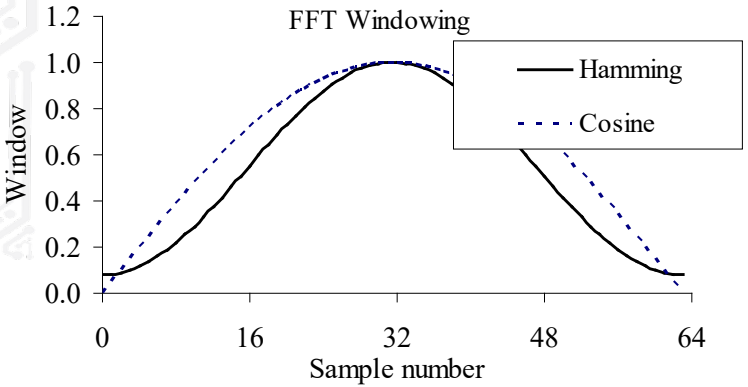
Spectral Leakage

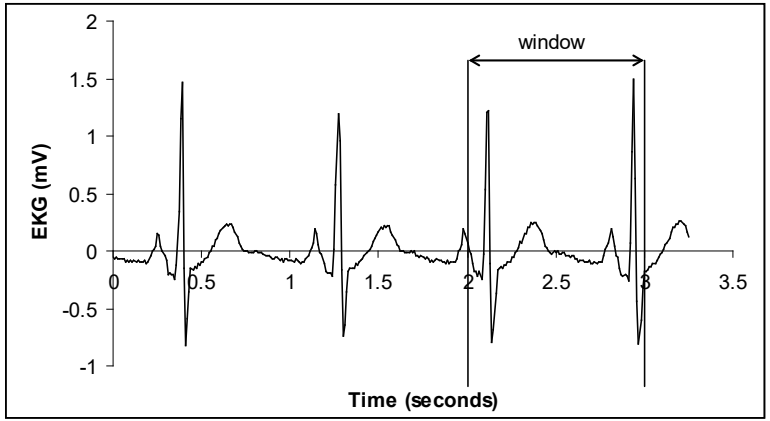
Interpreted as infinite periodic signal

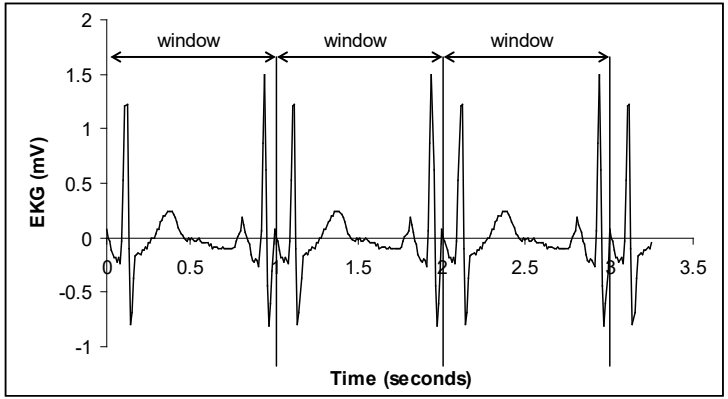
- Sample $\{a_0,a_1,a_2,\ldots,a_{N-1}\}$
- $\{\ldots a_0,a_1,a_2,\ldots,a_{N-1},a_0,a_1,a_2,\ldots,a_{N-1},\ldots\}$

Windowing

- Window shape removes error
  Taper down to zero at the beginning
  Taper down to zero at the end
- Multiply samples by window







Use Spectrum Analyzer to Verify Sampling Rate, Analog Filter

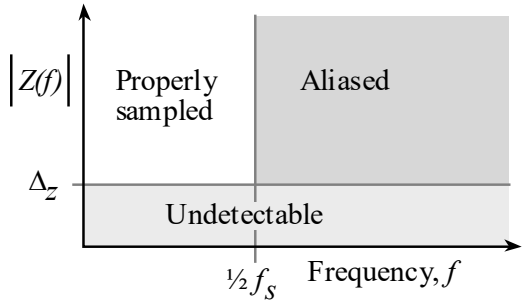
Proposed data acquisition system

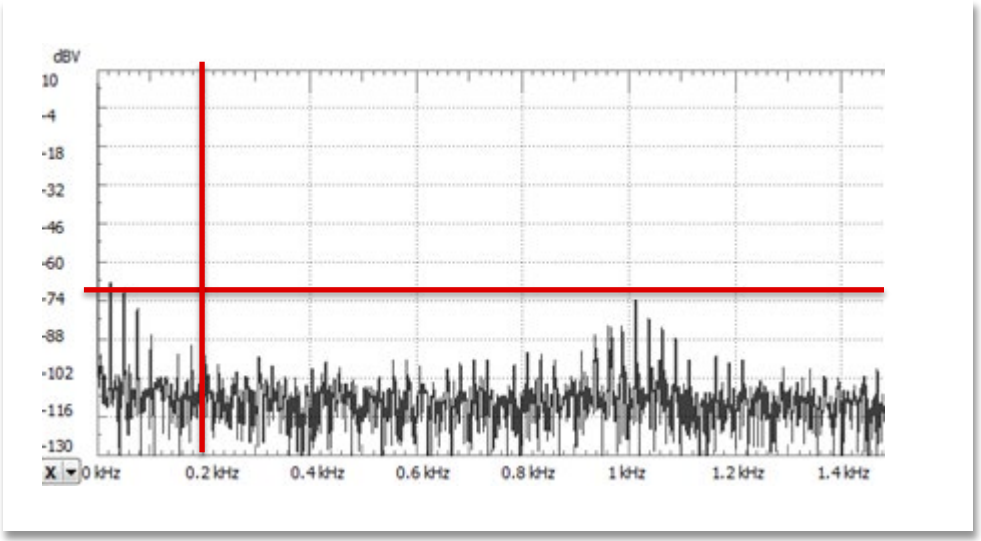
- f_s = 400 Hz
- 12-bit ADC
- 0 to 3.3V ADC range

Large high frequency data will alias

To avoid aliasing

- Consider f ≥ ½ f_s = 200 Hz
- Large if signal > 3.3V/4096
- $dB_{FS} = 20 \log_{10}(2^{-12}) = -72.2$





Use Spectrum Analyzer to Filter to Analyze Noise

60 Hz (large)

- EM pickup from AC power line
- 60, 120, 180, 240,… harmonics

DC motor noise (large)

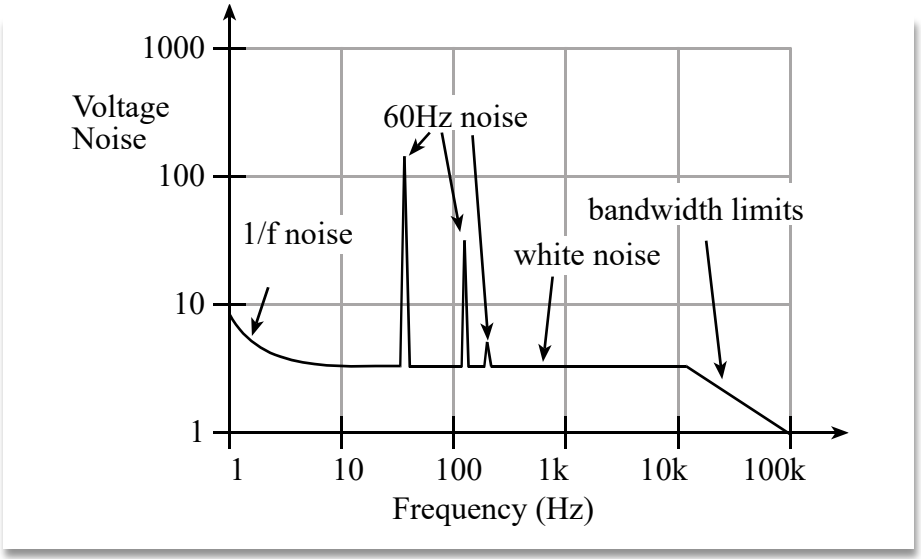
- PWM_Init34(15000, rightDuty, leftDuty);
- EM pickup from motors
- 50, 100, 150, 200,… harmonics

White noise (small)

- Constant noise amplitude
- Thermal uncertainty

Pink or 1/f noise (very small)

- Low frequency
- Variable conductivity



Summary

Discrete Fourier Transform

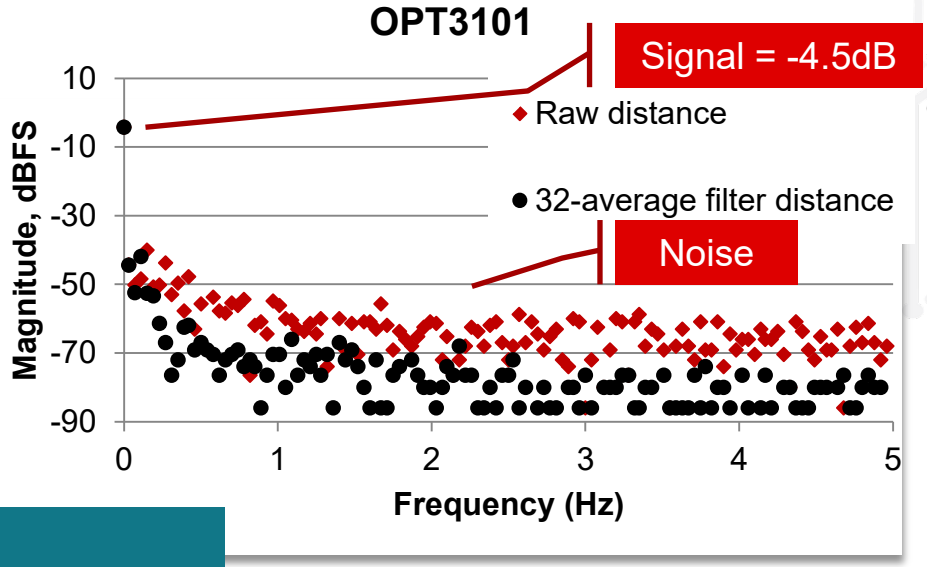
- Sampling rate, f_s
  - 0 to ½ f_s
- Number of samples N
- Frequency resolution $\Delta f = f_s/N$

$$A_k = \sum_{n=0}^{N-1} a_n W_N^{kn}$$

$$W_N = e^{-j2\pi/N}$$

Fast Fourier Transform

- N logN speed
- Index k is frequency ($k*f_s/N$)
- For k = 0 to ½ N
- Output A_k has same units as input
  Magnitude = $|A_k|/(N/2)$
  Phase = $\arctan(Im(A_k)/Re(A_k))$



SNR = 39 dB
$10^{39/20}$= 195 alternatives
ENOB = $\log_2(195)$ = 7.6 bits

ti.com/rslk

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265