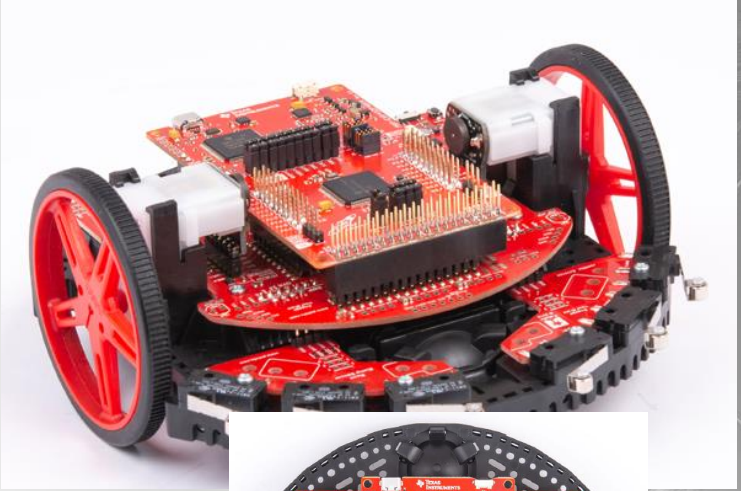
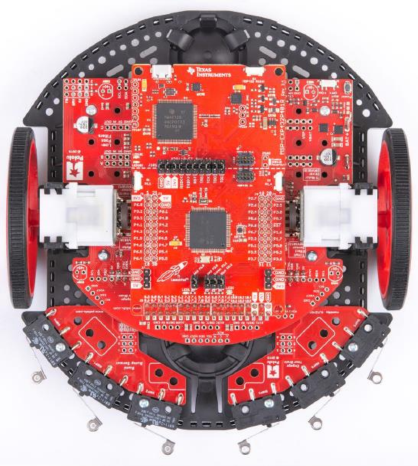
TI-RSLK MAX Workshop

Thorsten Lorenzen

TI-RSLK MAX getting started





Before we can start

Register for myTI account:

- ti.com
- Top right of the home page

Get access to CCS Cloud:

- https://dev.ti.com/
- Click on the buttons „Install“ and „Download“ to install the browser plugin
- Start CCS Cloud: https://dev.ti.com/ide

Review the home page of TI-RSLK robot:

- ti.com/rslk

Load the TI-RSLK User guide:

- http://www.ti.com/lit/ml/sekp166/sekp166.pdf

Review the TI page of TI-RSLK

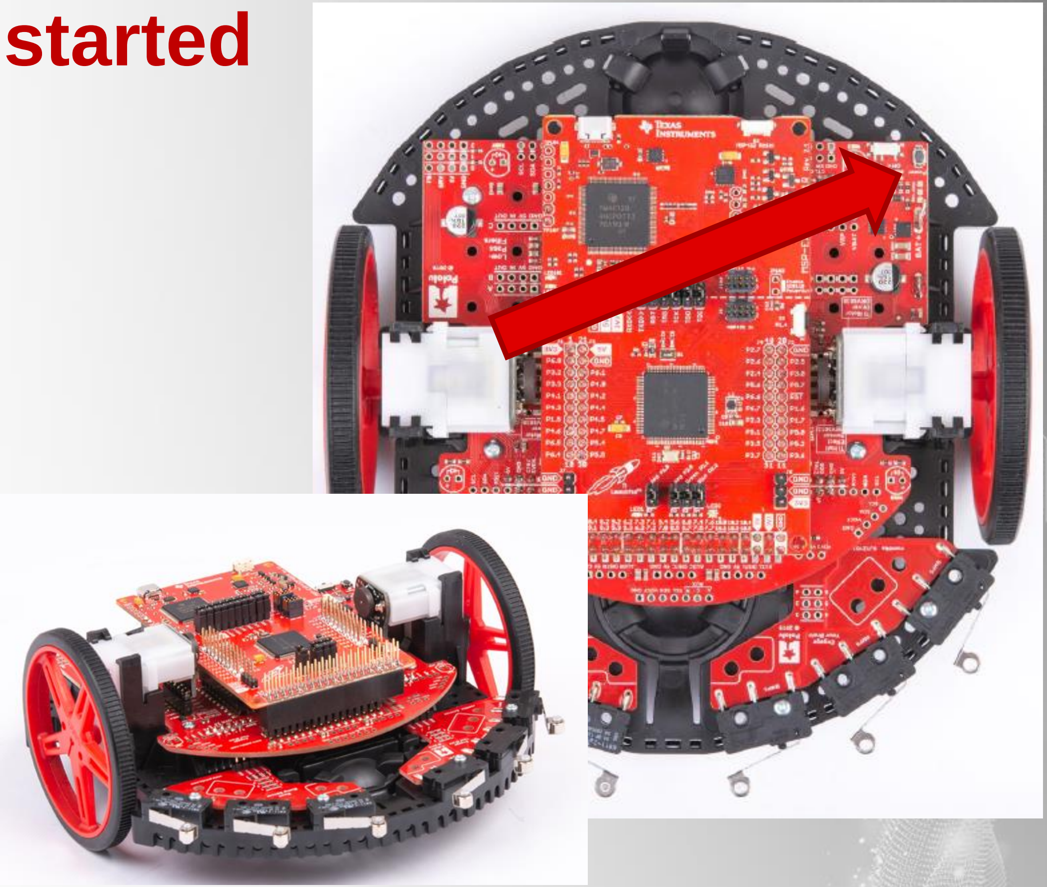
- http://www.ti.com/tool/TIRSLK-EVM?keyMatch=RSLK%20MAX&tisearch=Search-EN-everything&usecase=part-number

TI-RSLK MAX getting started

First steps for getting started

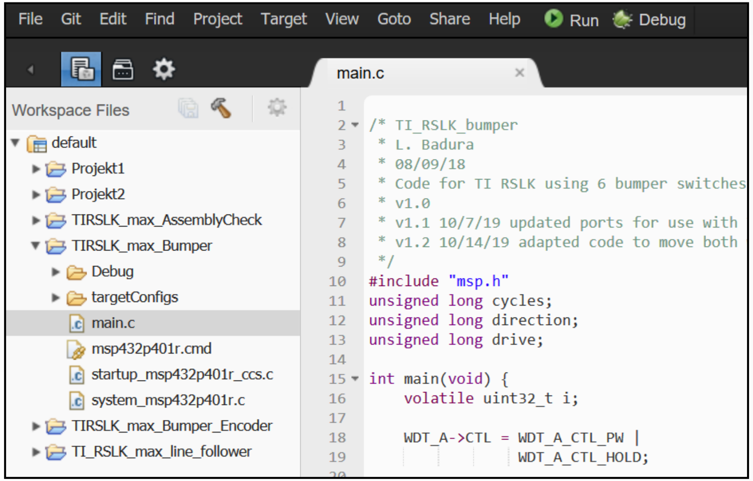
Wake up the robot for first out of the box experience

- Put the robot on your desk on its wheels
- Get familiar with your robot components it is built up
- Now flip your robot to get access to the battery compartment
- Put 6 batteries in the compartment with orientation MINUS pole to the spring
- Flip the robot back on its wheels on your desk
    - During the next step you will experience the move of the robut for 10cm. Don‘t worry ☺
- Hit the power button (RED arrow)
    - You will notice one Blue LED
- Robot should move for 10cm but halt after.
    - This is the prove the robot to be alive ☺
- Hit the power button again to turn off the robot

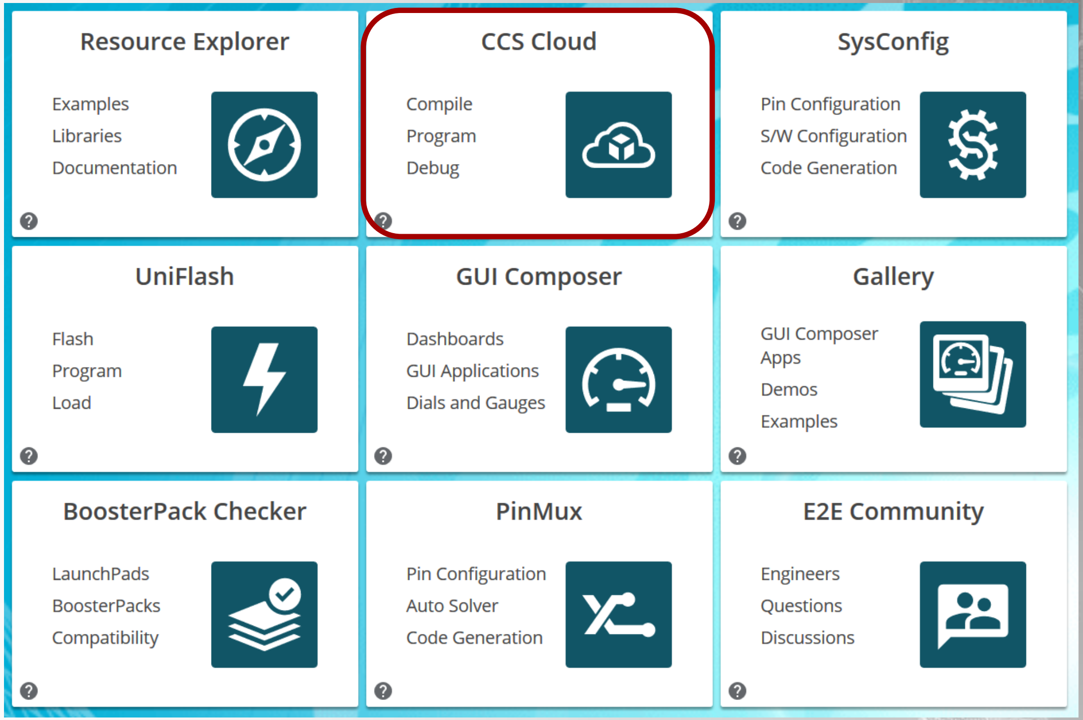


TI-RSLK MAX programming

- Programming with CCS cloud: https://dev.ti.com/
- Accessible from every computer (online account)
- Ready for coding in 5 minutes (browser add on download)



```

/* TI_RSLK_bumper
 * L. Badura
 * 08/09/18
 * Code for TI RSLK using 6 bumper switches
 * v1.0
 * v1.1 10/7/19 updated ports for use with
 * v1.2 10/14/19 adapted code to move both
 */
#include "msp.h"
unsigned long cycles;
unsigned long direction;
unsigned long drive;

int main(void) {
    volatile uint32_t i;

    WDT_A->CTL = WDT_A_CTL_PW |
                 WDT_A_CTL_HOLD;
```



Resource Explorer
Examples
Libraries
Documentation

CCS Cloud
Compile
Program
Debug

SysConfig
Pin Configuration
S/W Configuration
Code Generation

UniFlash
Flash
Program
Load

GUI Composer
Dashboards
GUI Applications
Dials and Gauges

Gallery
GUI Composer Apps
Demos
Examples

BoosterPack Checker
LaunchPads
BoosterPacks
Compatibility

PinMux
Pin Configuration
Auto Solver
Code Generation

E2E Community
Engineers
Questions
Discussions

TI-RSLK MAX programming

First steps for getting started

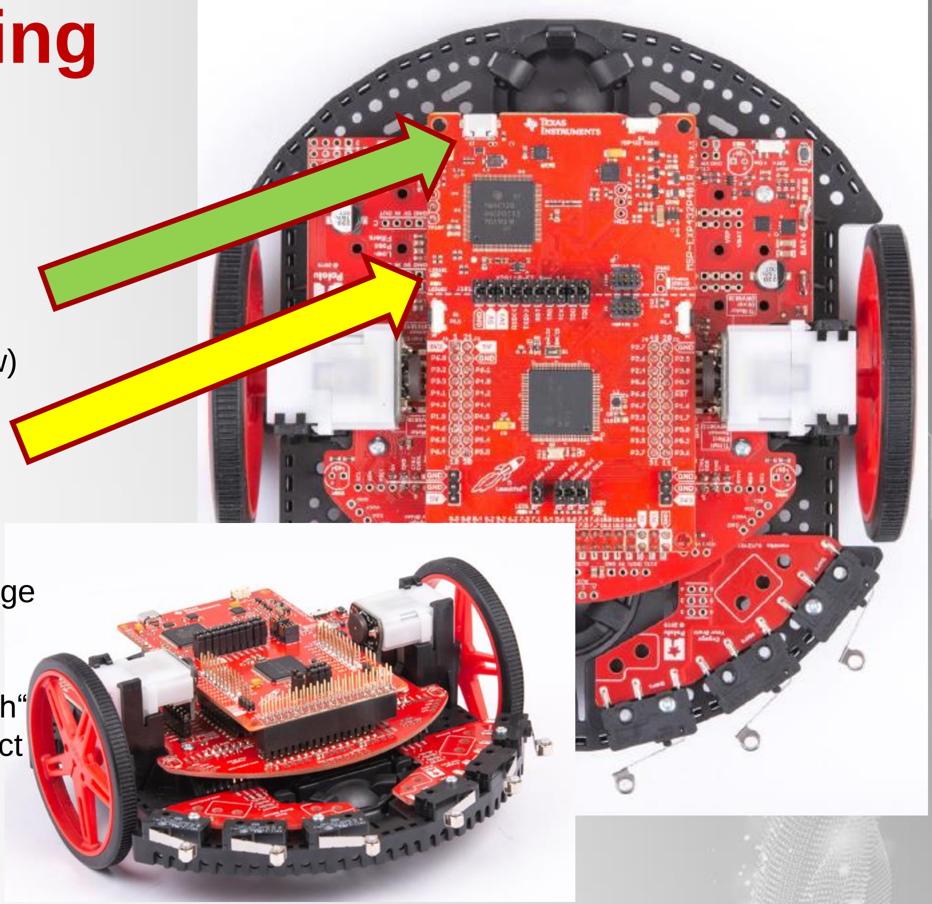
Establishing the connection RSLK to your computer

Connect USB cable to the connector (GREEN arrow)

- You should see the green LED to be illuminated (YELLOW arrow)
- It indicates the Computer to recognize the robot

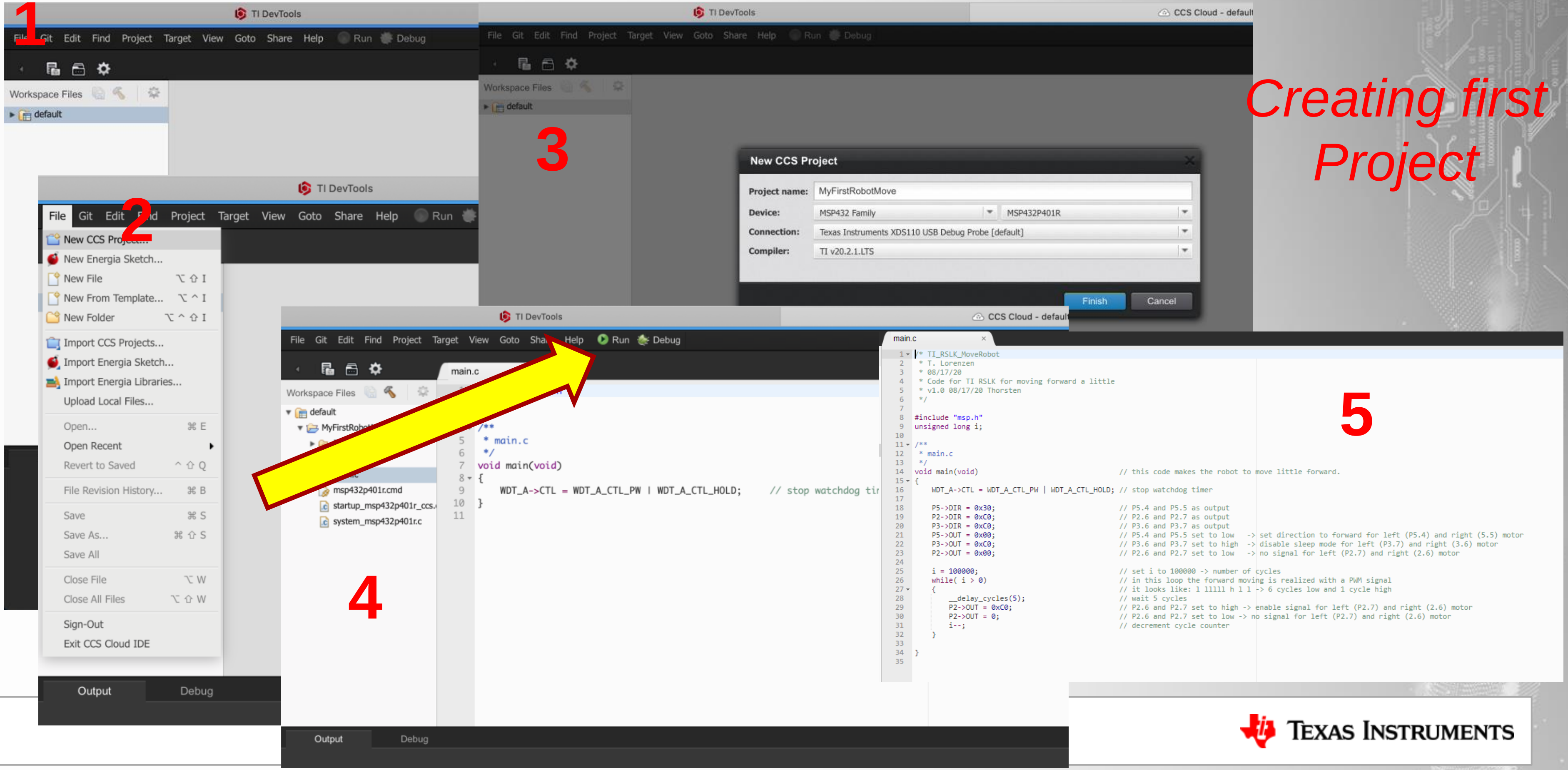
Open CCS Cloud Software Development Environment

- Start the link in your browser: https://dev.ti.com/
- Login with your myTI account details (top right of the page)
- On the left you see the statement „Start browsing“. Below the page indicates the robot‘s processor name „MSP432P401R “
- If not done so far click both the buttons „Install“ and „Download
- When finished scroll further down and click on the button „Launch“ in the field „CCS Cloud“ to start creating your first software project



TI-RSLK MAX Programming

Creating first Project



3 New CCS Project

Project name: MyFirstRobotMove

Device: MSP432 Family | MSP432P401R

Connection: Texas Instruments XDS110 USB Debug Probe [default]

Compiler: TI v20.2.1.LTS

4 main.c

```
/**
 * main.c
 */
void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog tir
}
```

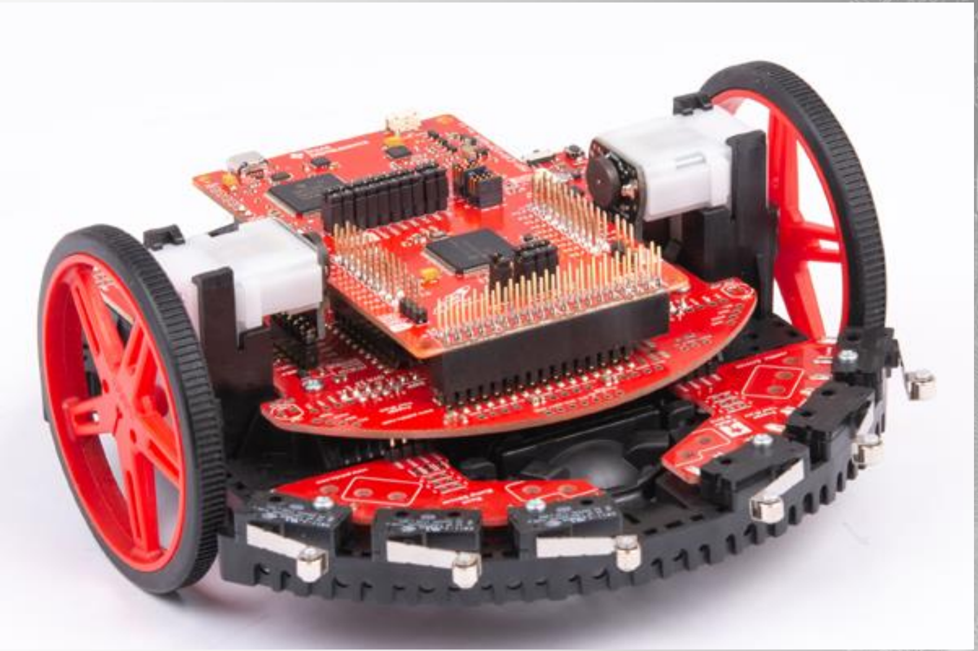
5 main.c

```
/* TI_RSLK_MoveRobot
 * T. Lorenzen
 * 08/17/20
 * Code for TI RSLK for moving forward a little
 * v1.0 08/17/20 Thorsten
 */

#include "msp.h"
unsigned long i;

/**
 * main.c
 */
void main(void)                                  // this code makes the robot to move little forward.
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD; // stop watchdog timer

    P5->DIR = 0x30;                              // P5.4 and P5.5 as output
    P2->DIR = 0xC0;                              // P2.6 and P2.7 as output
    P3->DIR = 0xC0;                              // P3.6 and P3.7 as output
    P5->OUT = 0x00;                              // P5.4 and P5.5 set to low   -> set direction to forward for left (P5.4) and right (5.5) motor
    P3->OUT = 0xC0;                              // P3.6 and P3.7 set to high  -> disable sleep mode for left (P3.7) and right (3.6) motor
    P2->OUT = 0x00;                              // P2.6 and P2.7 set to low   -> no signal for left (P2.7) and right (2.6) motor

    i = 100000;                                  // set i to 100000 -> number of cycles
    while( i > 0)                                // in this loop the forward moving is realized with a PWM signal
    {                                            // it looks like: l lllll h l l -> 6 cycles low and 1 cycle high
        __delay_cycles(5);                       // wait 5 cycles
        P2->OUT = 0xC0;                          // P2.6 and P2.7 set to high -> enable signal for left (P2.7) and right (2.6) motor
        P2->OUT = 0;                             // P2.6 and P2.7 set to low -> no signal for left (P2.7) and right (2.6) motor
        i--;                                     // decrement cycle counter
    }

}
```

TI-RSLK MAX introduction

MSP432P401R

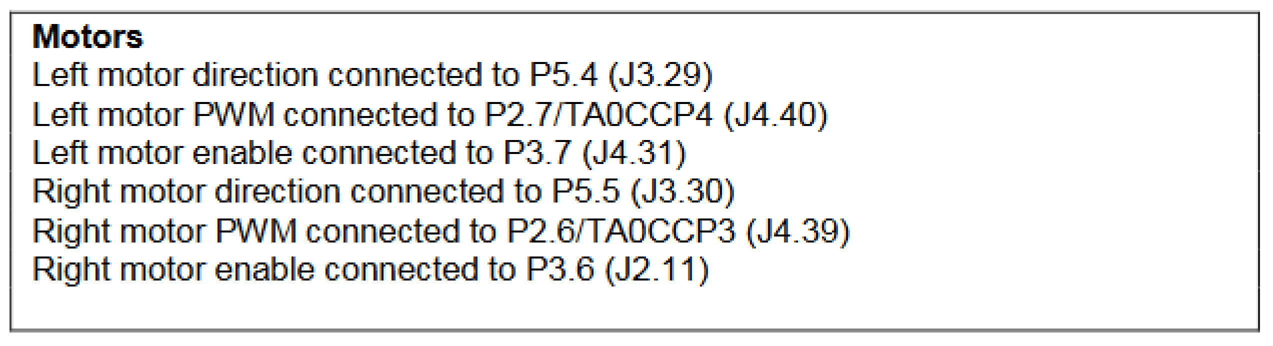
- Arm® 32-bit Cortex®-M4F CPU with floating-point unit and memory protection unit
- Up to 256KB of flash main memory (organized into two banks enabling simultaneous read or execute during erase)
- Up to 64KB of SRAM (including 6KB of backup memory)
- LPM4.5: 25 nA



TI-RSLK MAX Motor Control

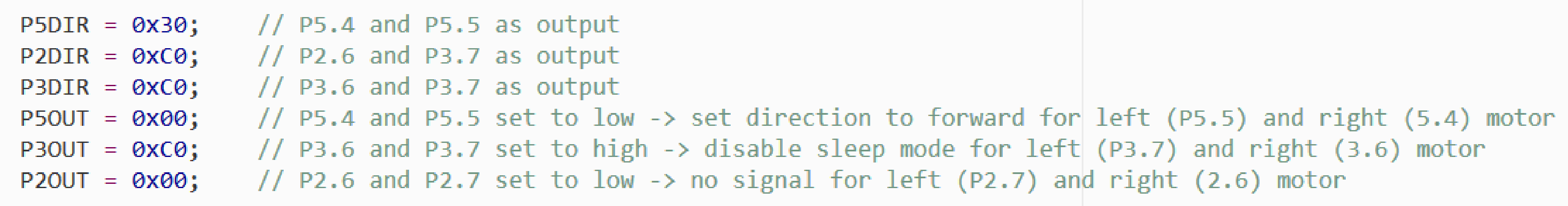
TI-RSLK MAX motors

- TI-RSLK MAX contains 2 motors with motor drivers and encoders.



Motors
Left motor direction connected to P5.4 (J3.29)
Left motor PWM connected to P2.7/TA0CCP4 (J4.40)
Left motor enable connected to P3.7 (J4.31)
Right motor direction connected to P5.5 (J3.30)
Right motor PWM connected to P2.6/TA0CCP3 (J4.39)
Right motor enable connected to P3.6 (J2.11)

- Motor driver pins are connected to MSP432 GPIO



```
P5DIR = 0x30;     // P5.4 and P5.5 as output
P2DIR = 0xC0;     // P2.6 and P3.7 as output
P3DIR = 0xC0;     // P3.6 and P3.7 as output
P5OUT = 0x00;     // P5.4 and P5.5 set to low -> set direction to forward for left (P5.5) and right (5.4) motor
P3OUT = 0xC0;     // P3.6 and P3.7 set to high -> disable sleep mode for left (P3.7) and right (3.6) motor
P2OUT = 0x00;     // P2.6 and P2.7 set to low -> no signal for left (P2.7) and right (2.6) motor
```

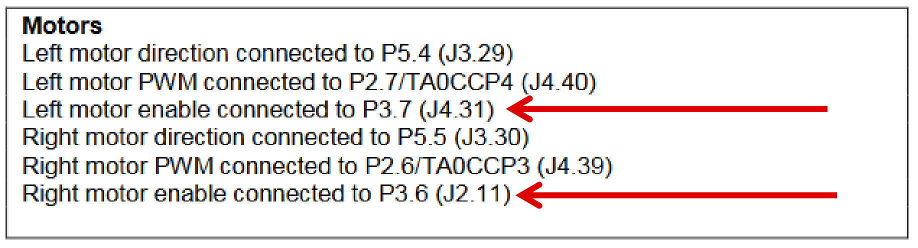
PORT register syntax

Output Registers (PxOUT)

- Bit = 0: Output is low
- Bit = 1: Output is high

Direction Registers (PxDIR)

- Bit = 0: Port pin is switched to input direction
- Bit = 1: Port pin is switched to output direction



Motors
Left motor direction connected to P5.4 (J3.29)
Left motor PWM connected to P2.7/TA0CCP4 (J4.40)
Left motor enable connected to P3.7 (J4.31)
Right motor direction connected to P5.5 (J3.30)
Right motor PWM connected to P2.6/TA0CCP3 (J4.39)
Right motor enable connected to P3.6 (J2.11)

P3OUT = 0xC0;

P3OUT Bits	7	6	5	4	3	2	1	0
register value	8	4	2	1	8	4	2	1
pin state	1	1	0	0	0	0	0	0

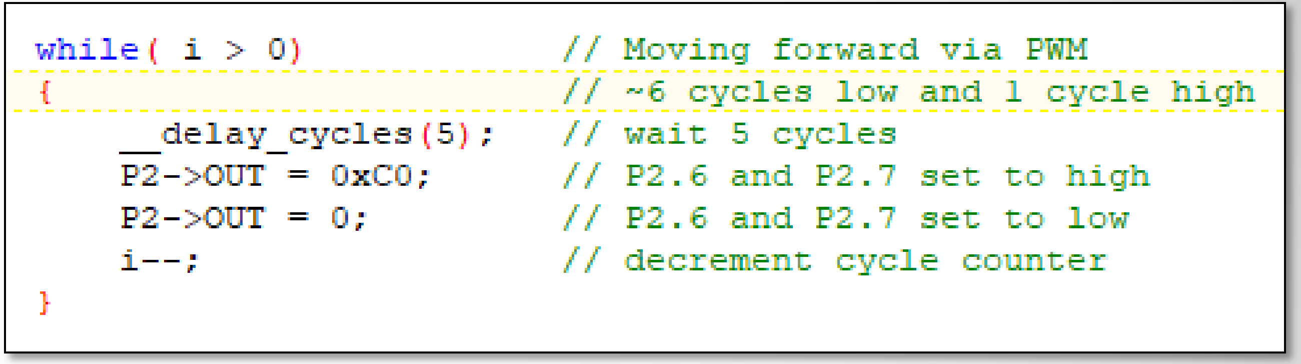
P3DIR = 0xC0;

P3DIR Bits	7	6	5	4	3	2	1	0
register value	8	4	2	1	8	4	2	1
pin state	1	1	0	0	0	0	0	0

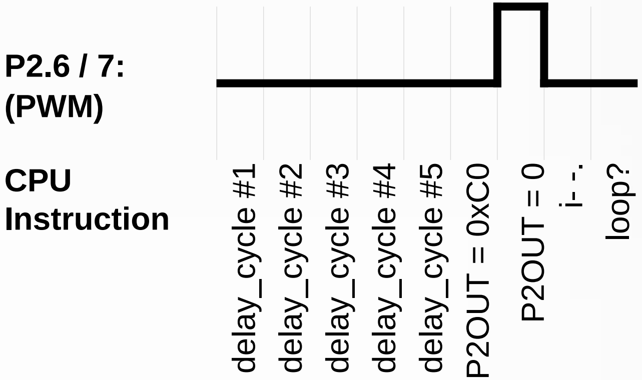
8 + 4 + 0 + 0
= 12 -> 0xC

Pulse Width Modulation for motor control

- Software PWM with CPU control



```
while( i > 0)                // Moving forward via PWM
{                            // ~6 cycles low and 1 cycle high
    __delay_cycles(5);       // wait 5 cycles
    P2->OUT = 0xC0;          // P2.6 and P2.7 set to high
    P2->OUT = 0;             // P2.6 and P2.7 set to low
    i--;                     // decrement cycle counter
}
```

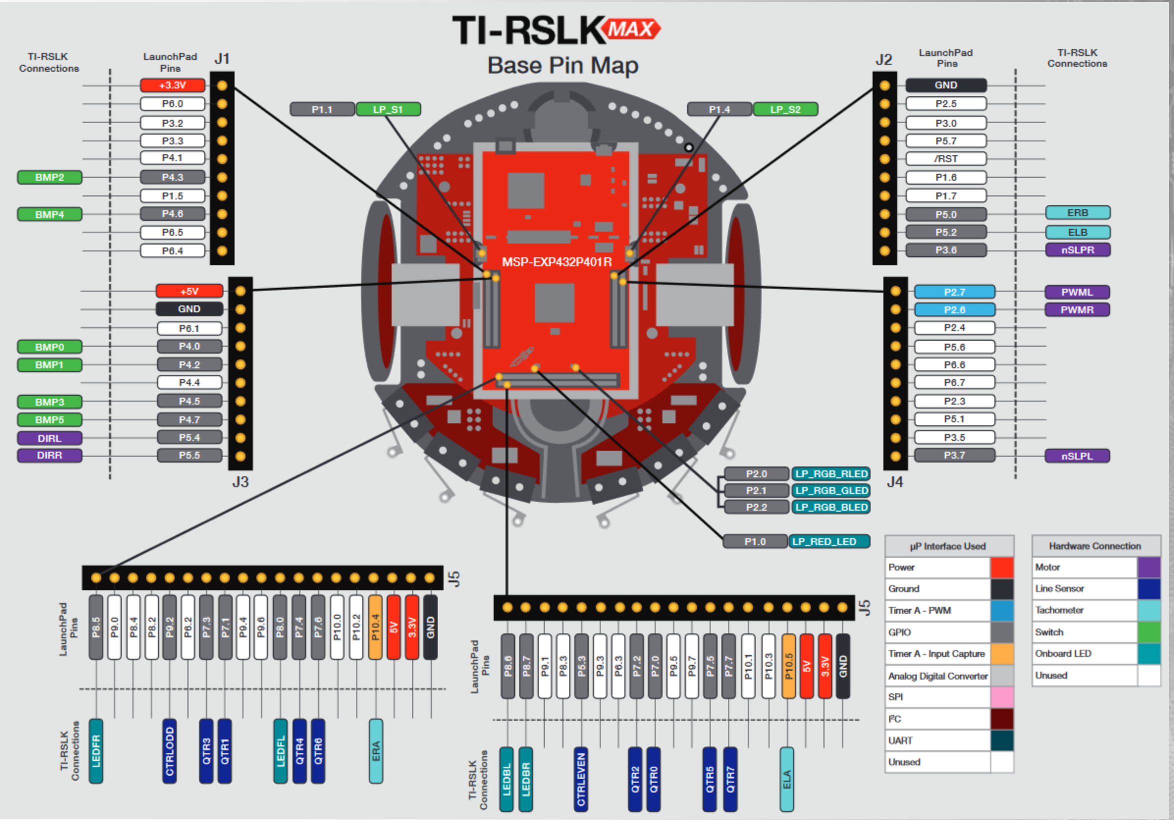


A few details to keep in mind:

- Initialization of pins and variables (not shown here)
- Number of CPU clocks per instruction (see datasheet, or Oscilloscope)
- CPU power used for software PWM (anything else would need interrupts)
- Handling of interrupts during high phase?

TI-RSLK MAX pin map

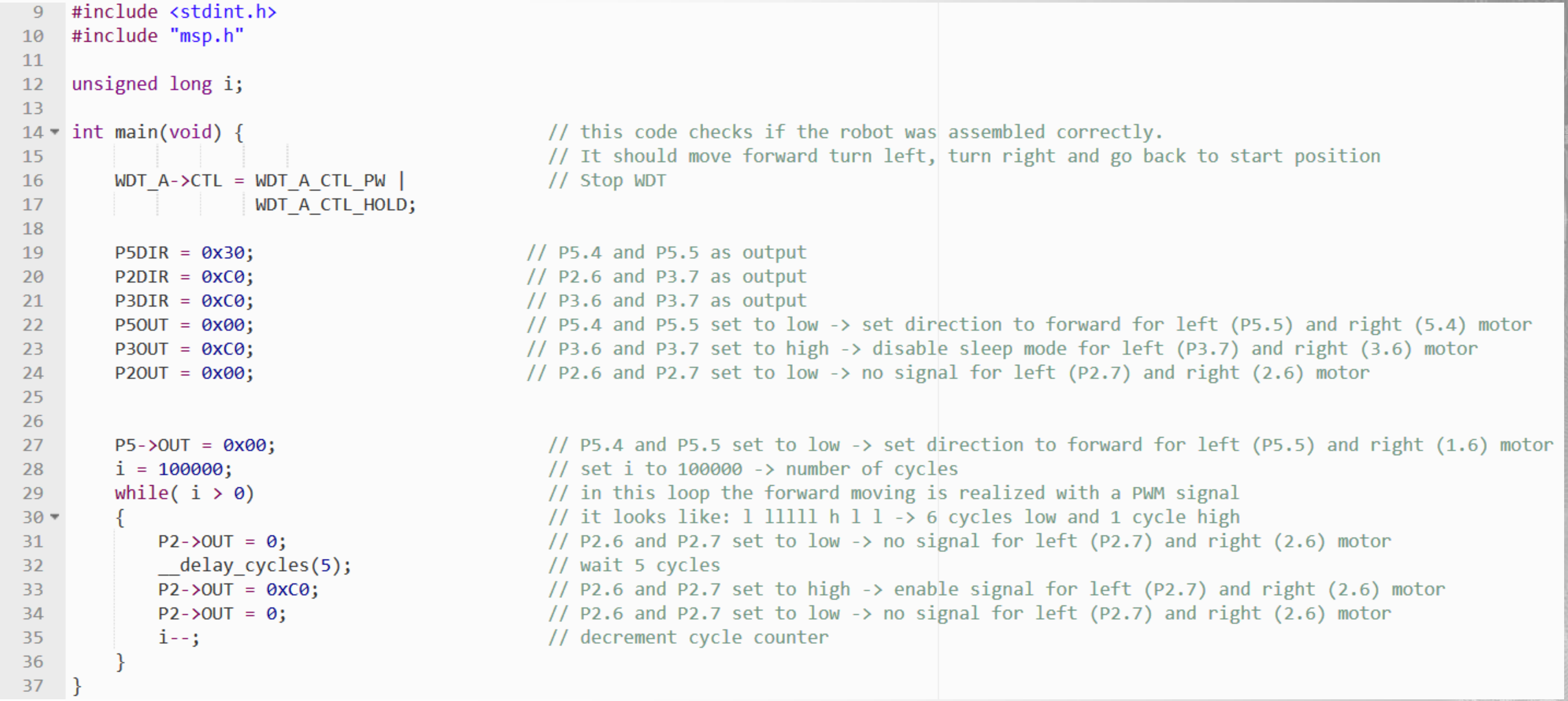
http://www.ti.com/lit/pdf/SEKP171



TEXAS INSTRUMENTS

TI-RSLK MAX First Move

Final code for driving the robot forward



```c
#include <stdint.h>
#include "msp.h"

unsigned long i;

int main(void) {                            // this code checks if the robot was assembled correctly.
                                            // It should move forward turn left, turn right and go back to start position
    WDT_A->CTL = WDT_A_CTL_PW |             // Stop WDT
                 WDT_A_CTL_HOLD;

    P5DIR = 0x30;                         // P5.4 and P5.5 as output
    P2DIR = 0xC0;                         // P2.6 and P3.7 as output
    P3DIR = 0xC0;                         // P3.6 and P3.7 as output
    P5OUT = 0x00;                         // P5.4 and P5.5 set to low -> set direction to forward for left (P5.5) and right (5.4) motor
    P3OUT = 0xC0;                         // P3.6 and P3.7 set to high -> disable sleep mode for left (P3.7) and right (3.6) motor
    P2OUT = 0x00;                         // P2.6 and P2.7 set to low -> no signal for left (P2.7) and right (2.6) motor


    P5->OUT = 0x00;                         // P5.4 and P5.5 set to low -> set direction to forward for left (P5.5) and right (1.6) motor
    i = 100000;                             // set i to 100000 -> number of cycles
    while( i > 0)                           // in this loop the forward moving is realized with a PWM signal
    {                                       // it looks like: l lllll h l l -> 6 cycles low and 1 cycle high
        P2->OUT = 0;                        // P2.6 and P2.7 set to low -> no signal for left (P2.7) and right (2.6) motor
        __delay_cycles(5);                  // wait 5 cycles
        P2->OUT = 0xC0;                     // P2.6 and P2.7 set to high -> enable signal for left (P2.7) and right (2.6) motor
        P2->OUT = 0;                        // P2.6 and P2.7 set to low -> no signal for left (P2.7) and right (2.6) motor
        i--;                                // decrement cycle counter
    }
}
```

Final code for driving the robot forward

Source code robot move

- You can copy and paste the code to your CCS Cloud project and hit “Run”

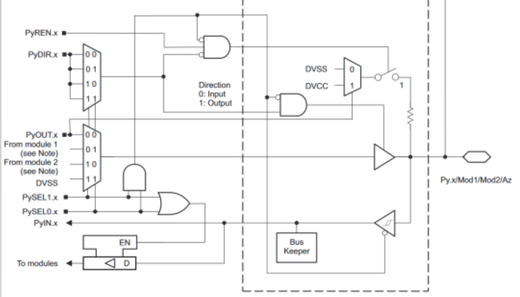
```
/* TI_RSLK_MoveRobot
 * T. Lorenzen
 * 08/17/20
 * Code for TI RSLK for moving forward a little
 * v1.0 08/17/20 Thorsten
 */

#include "msp.h"
unsigned long i;

/**
 * main.c
 */
void main(void)                              // this code makes the robot to move little forward.
{
                WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD; // stop watchdog timer

   P5->DIR = 0x30;                           // P5.4 and P5.5 as output
   P2->DIR = 0xC0;                           // P2.6 and P2.7 as output
   P3->DIR = 0xC0;                           // P3.6 and P3.7 as output
   P5->OUT = 0x00;                           // P5.4 and P5.5 set to low   -> set direction to forward for left (P5.4) and right (5.5) motor
   P3->OUT = 0xC0;                           // P3.6 and P3.7 set to high  -> disable sleep mode for left (P3.7) and right (3.6) motor
   P2->OUT = 0x00;                           // P2.6 and P2.7 set to low   -> no signal for left (P2.7) and right (2.6) motor

   i = 100000;                           // set i to 100000 -> number of cycles
   while( i > 0)                         // in this loop the forward moving is realized with a PWM signal
   {                                 // it looks like: l lllll h l l -> 6 cycles low and 1 cycle high
      __delay_cycles(5);                     // wait 5 cycles
      P2->OUT = 0xC0;                         // P2.6 and P2.7 set to high -> enable signal for left (P2.7) and right (2.6) motor
      P2->OUT = 0;                          // P2.6 and P2.7 set to low -> no signal for left (P2.7) and right (2.6) motor
      i--;                           // decrement cycle counter
   }

}
```

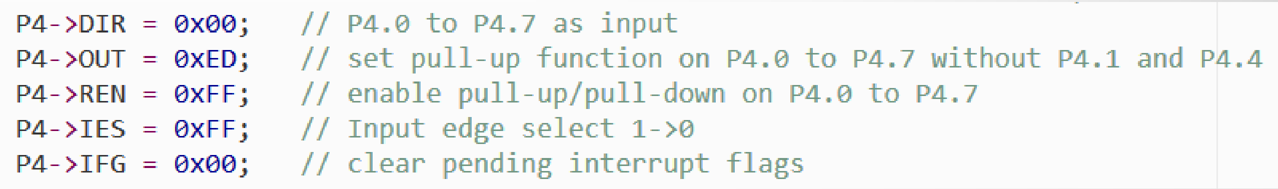
TI-RSLK MAX Use Bumper Support

Bumper code implementation

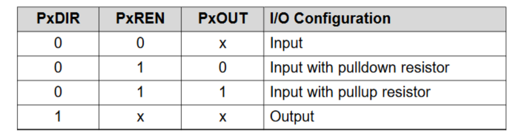
- TI-RSLK MAX contains 6 bumpers realized as switches configured as pull-up

Negative logic bump sensors
P4.7 Bump5, left side of robot
P4.6 Bump4
P4.5 Bump3
P4.3 Bump2
P4.2 Bump1
P4.0 Bump0, right side of robot

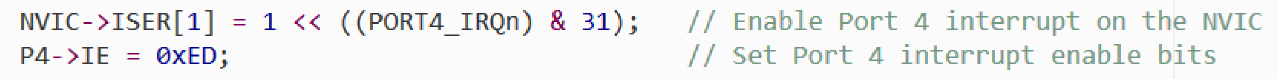




```
P4->DIR = 0x00;    // P4.0 to P4.7 as input
P4->OUT = 0xED;    // set pull-up function on P4.0 to P4.7 without P4.1 and P4.4
P4->REN = 0xFF;    // enable pull-up/pull-down on P4.0 to P4.7
P4->IES = 0xFF;    // Input edge select 1->0
P4->IFG = 0x00;    // clear pending interrupt flags
```



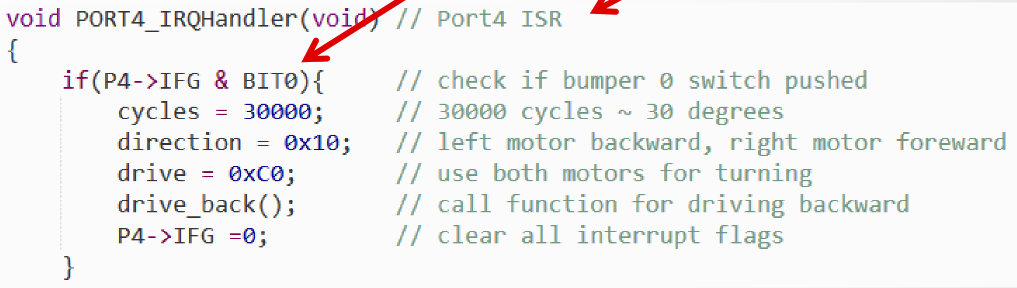
PxDIR	PxREN	PxOUT	I/O Configuration
0	0	x	Input
0	1	0	Input with pulldown resistor
0	1	1	Input with pullup resistor
1	x	x	Output



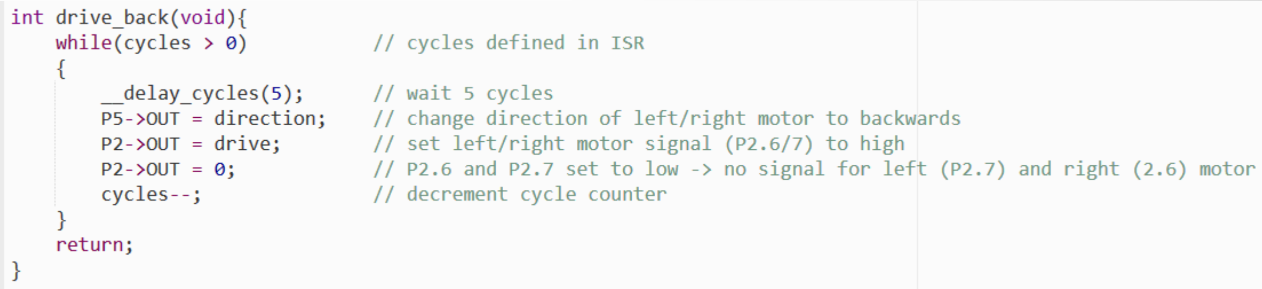
```
NVIC->ISER[1] = 1 << ((PORT4_IRQn) & 31);   // Enable Port 4 interrupt on the NVIC
P4->IE = 0xED;                              // Set Port 4 interrupt enable bits
```

Bumper code implementation

- Pushing one of the switches, results in a jump to the interrupt service routine.
- Within the ISR it is checked which switch was pushed

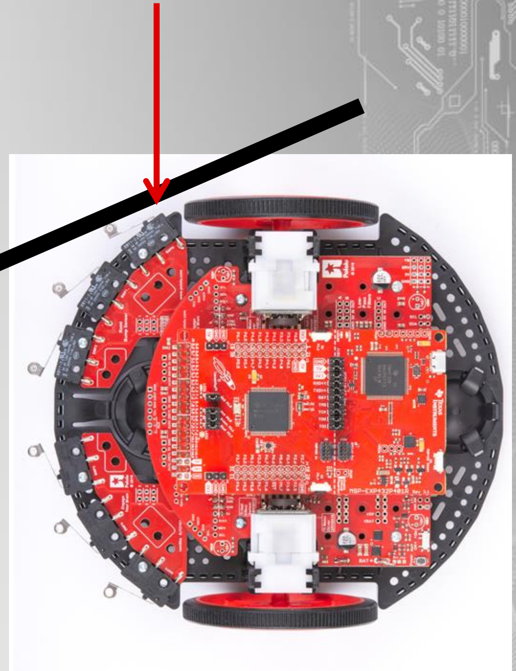


```
void PORT4_IRQHandler(void) // Port4 ISR
{
    if(P4->IFG & BIT0){       // check if bumper 0 switch pushed
        cycles = 30000;       // 30000 cycles ~ 30 degrees
        direction = 0x10;     // left motor backward, right motor foreward
        drive = 0xC0;         // use both motors for turning
        drive_back();         // call function for driving backward
        P4->IFG =0;           // clear all interrupt flags
    }
```



```
int drive_back(void){
    while(cycles > 0)           // cycles defined in ISR
    {
        __delay_cycles(5);      // wait 5 cycles
        P5->OUT = direction;    // change direction of left/right motor to backwards
        P2->OUT = drive;        // set left/right motor signal (P2.6/7) to high
        P2->OUT = 0;            // P2.6 and P2.7 set to low -> no signal for left (P2.7) and right (2.6) motor
        cycles--;               // decrement cycle counter
    }
    return;
}
```

Bumper 0
=> P4.0



Final code for controlling the robot with bumpers

Source code robot bumper

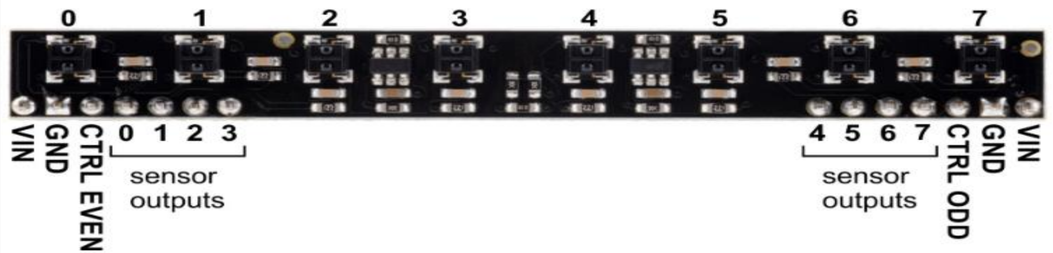
- You can copy and paste the code to your CCS Cloud project and hit “Run”

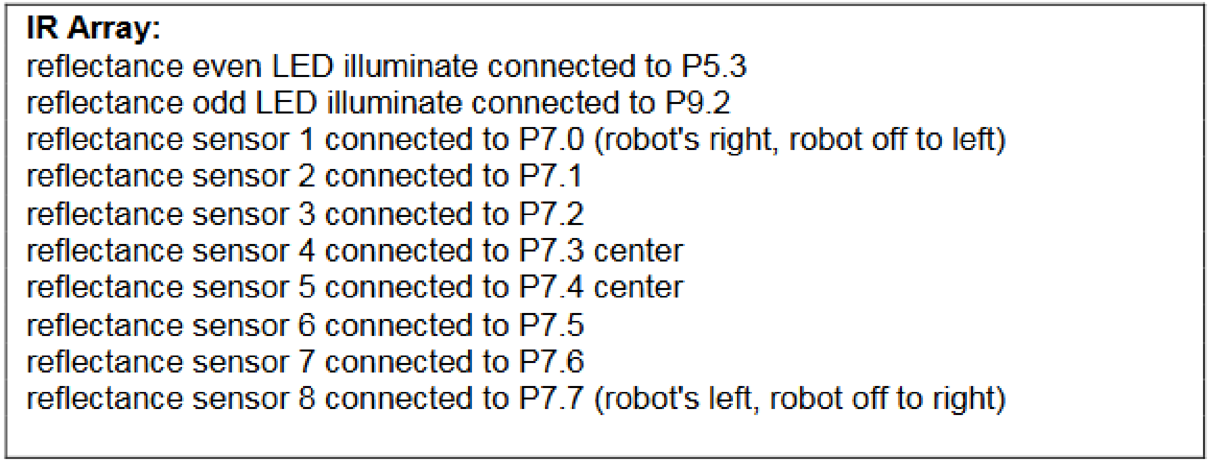
```
/* TI_RSLK_bumper
 * T. Lorenzen
 * 08/17/20
 * Code for TI RSLK using 6 bumper switches
 * v1.4 08/17/20 Thorsten
 */

#include "msp.h"
unsigned long cycles;
unsigned long direction;
unsigned long drive;

int main(void) {                        // this code uses its bumper switches to walk around obstacles while moving around.
    volatile uint32_t i;

    WDT_A->CTL = WDT_A_CTL_PW |              // Stop WDT
             WDT_A_CTL_HOLD;

    P5->DIR = 0x30;                     // P5.4 and P5.5 as output -> set direction to forward for left (P5.4) and right (5.5) motor
    P2->DIR = 0xC0;                     // P2.6 and P3.7 as output -> no signal for left (P2.7) and right (2.6) motor
    P3->DIR = 0xC0;                     // P3.6 and P3.7 as output -> disable sleep mode for left (P3.7) and right (3.6) motor
    P4->DIR = 0x00;                     // P4.0 to P4.7 as input -> Interrupt setup for bumper switches
    P4->OUT = 0xED;                      // set pull-up function on P4.0 to P4.7 without P4.1 and P4.4
    P4->REN = 0xFF;                      // enable pull-up/pull-down on P4.0 to P4.7
    P4->IES = 0xFF;                     // Input edge select 1->0
    P4->IFG = 0x00;                     // clear pending interrupt flags

    P5->OUT = 0x00;                      // P5.4 and P5.5 set to low -> set output to forward for left (P5.4) and right (5.5) motor
    P3->OUT = 0xC0;                      // P3.6 and P3.7 set to high -> disable sleep mode for left (P3.7) and right (3.6) motor
    P2->OUT = 0x00;                      // P2.6 and P2.7 set to low -> no signal for left (P2.7) and right (2.6) motor

    NVIC->ISER[1] = 1 << ((PORT4_IRQn) & 31);   // Enable Port 4 interrupt on the NVIC
    P4->IE = 0xED;                      // Set Port 4 interrupt enable bits

    while(1)                          // in this loop the forward moving is realized with a PWM signal
    {                               // it looks like: l lllll h l l -> 6 cycles low and 1 cycle high
        P5->OUT = 0;                    // P5.4 and P5.5 set to low -> reset output to forward for left (P5.4) and right (5.5) motor
        P2->OUT = 0;                    // P2.6 and P2.7 set to low -> no signal for left (P2.7) and right (2.6) motor
        __delay_cycles(5);               // wait 5 cycles
        P2->OUT = 0xC0;                   // P2.6 and P2.7 set to high -> enable signal for left (P2.7) and right (2.6) motor
        P2->OUT = 0;                    // P2.6 and P2.7 set to low -> no signal for left (P2.7) and right (2.6) motor
    }
}

int drive_back(void){
    while(cycles > 0)                    // cycles defined in ISR
    {
        P2->OUT = 0;                    // P2.6 and P2.7 set to low -> no signal for left (P2.7) and right (2.6) motor
        __delay_cycles(5);               // wait 5 cycles
        P5->OUT = direction;              // change direction of left/right motor to backwards
        P2->OUT = drive;                                     // set left/right motor signal (P2.6/7) to high
        P2->OUT = 0;                    // P2.6 and P2.7 set to low -> no signal for left (P2.7) and right (2.6) motor
        cycles--;                     // decrement cycle counter
    }
    return;
}

void PORT4_IRQHandler(void)                 // Port4 ISR //
{
    if(P4->IFG){                        // check if one bumpers switch high
        cycles = 70000;                      // Turn 70000 cycles ~ 90 degrees
        direction = 0x20;                 // Used for turn bit of one motor P5 to make a turn
        drive = 0xC0;                    // Motor drive bit P2
        drive_back();                    // call function for driving backward
        P4->IFG =0;                      // clear all interrupt flags
    }
}
```

TI-RSLK MAX Build the Line Follower

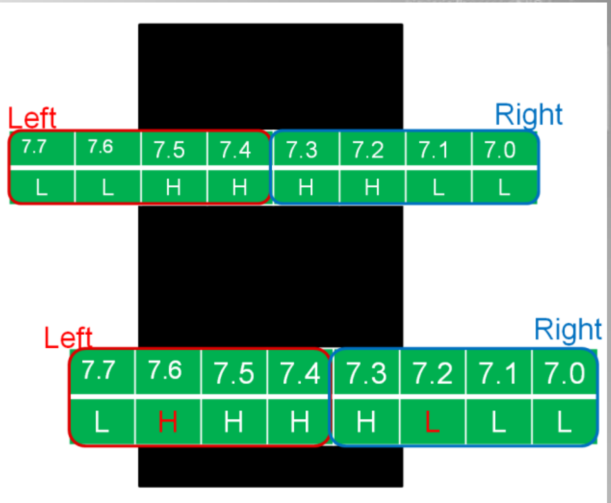
Line follower implementation

- TI-RSLK MAX contains an IR sensor connected to P7
- The sensor detects bright/dark surface on each of the 8 photo transistors.
- Based on this detection the TI-RSLK is moving along a black line




IR Array:

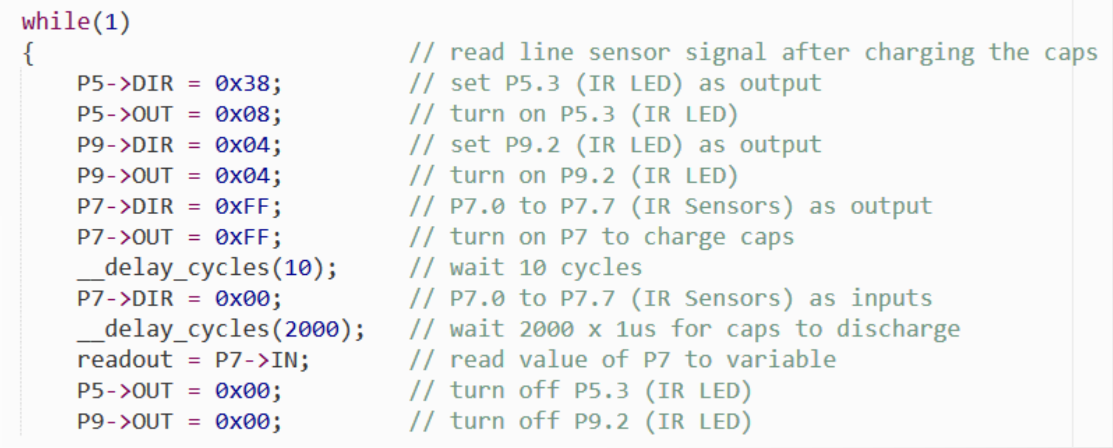
reflectance even LED illuminate connected to P5.3
reflectance odd LED illuminate connected to P9.2
reflectance sensor 1 connected to P7.0 (robot's right, robot off to left)
reflectance sensor 2 connected to P7.1
reflectance sensor 3 connected to P7.2
reflectance sensor 4 connected to P7.3 center
reflectance sensor 5 connected to P7.4 center
reflectance sensor 6 connected to P7.5
reflectance sensor 7 connected to P7.6
reflectance sensor 8 connected to P7.7 (robot's left, robot off to right)



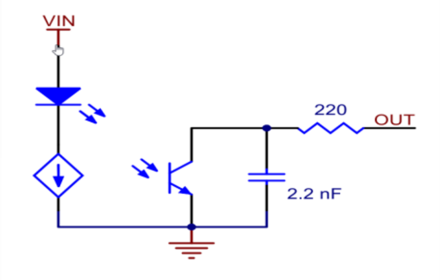
7.7	7.6	7.5	7.4	7.3	7.2	7.1	7.0
L	L	H	H	H	H	L	L

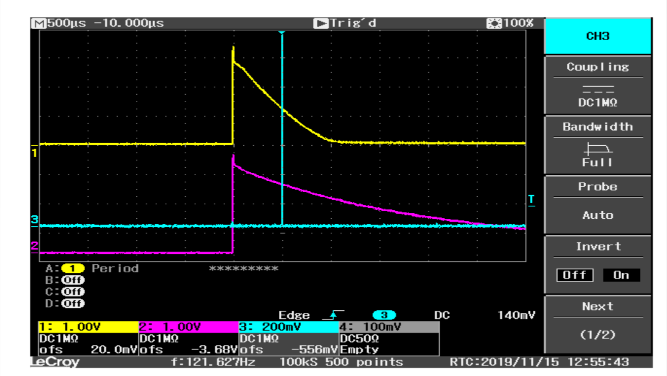
7.7	7.6	7.5	7.4	7.3	7.2	7.1	7.0
L	H	H	H	H	L	L	L

Line follower implementation



```
while(1)
{                                   // read line sensor signal after charging the caps
    P5->DIR = 0x38;                 // set P5.3 (IR LED) as output
    P5->OUT = 0x08;                 // turn on P5.3 (IR LED)
    P9->DIR = 0x04;                 // set P9.2 (IR LED) as output
    P9->OUT = 0x04;                 // turn on P9.2 (IR LED)
    P7->DIR = 0xFF;                 // P7.0 to P7.7 (IR Sensors) as output
    P7->OUT = 0xFF;                 // turn on P7 to charge caps
    __delay_cycles(10);             // wait 10 cycles
    P7->DIR = 0x00;                 // P7.0 to P7.7 (IR Sensors) as inputs
    __delay_cycles(2000);           // wait 2000 x 1us for caps to discharge
    readout = P7->IN;               // read value of P7 to variable
    P5->OUT = 0x00;                 // turn off P5.3 (IR LED)
    P9->OUT = 0x00;                 // turn off P9.2 (IR LED)
```



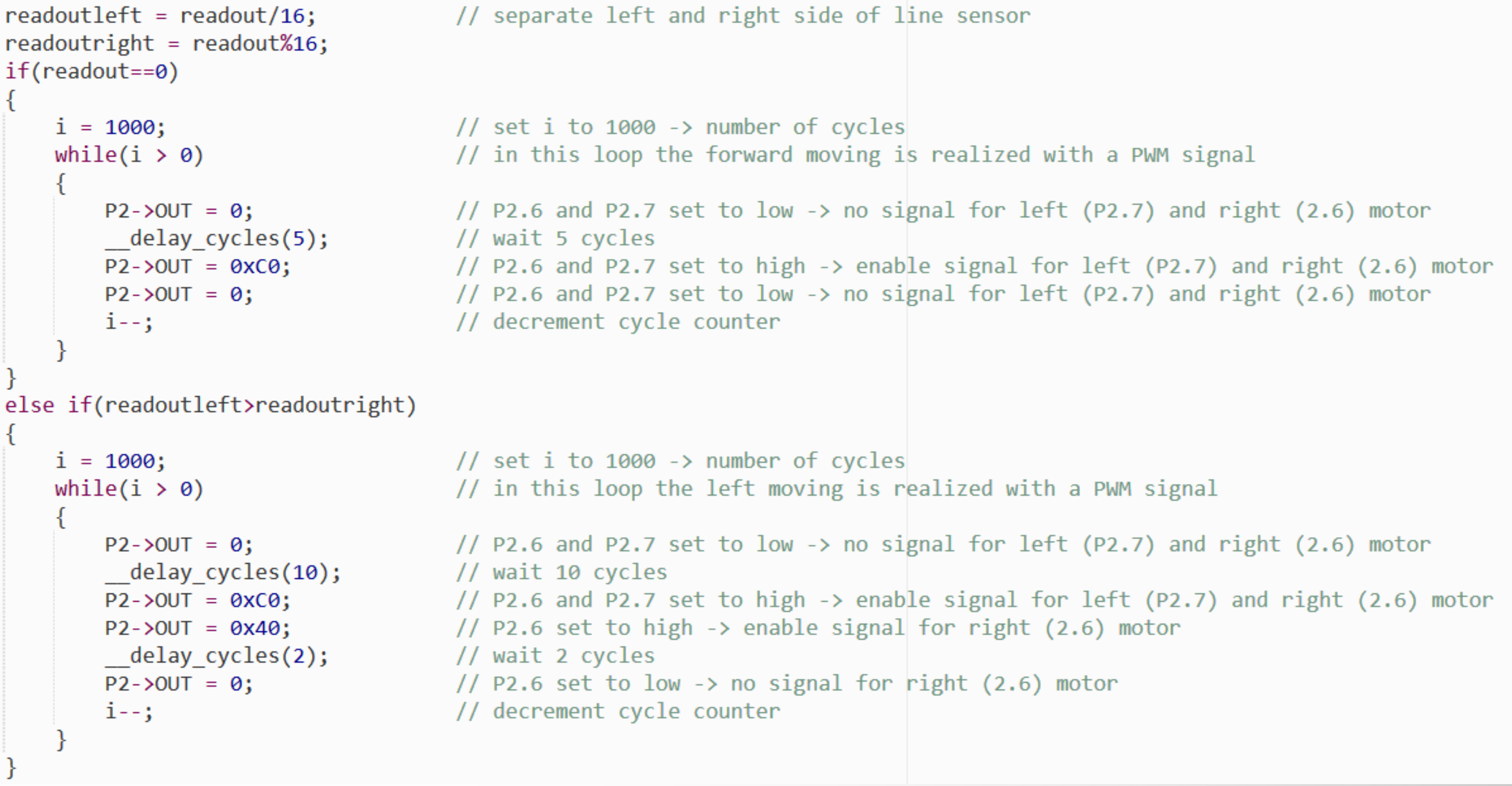


Reading the sensors

The typical sequence for reading a sensor is:

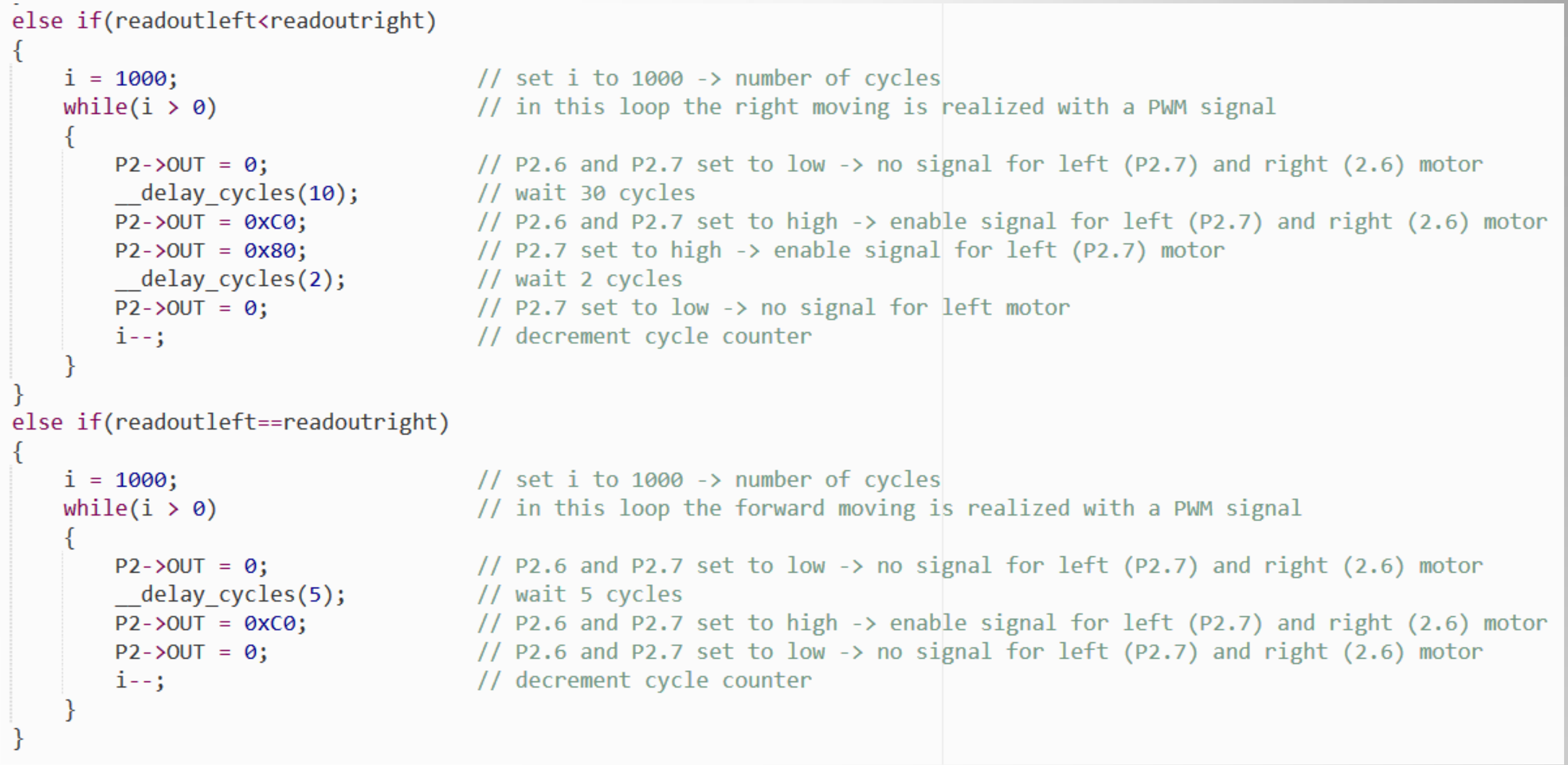
1. Turn on IR LEDs (optional).
2. Set the I/O line to an output and drive it high.
3. Allow at least 10 µs for the sensor output to rise.
4. Make the I/O line an input (high impedance).
5. Measure the time for the voltage to decay by waiting for the I/O line to go low.
6. Turn off IR LEDs (optional).
7. These steps can typically be executed in parallel on multiple I/O lines.

Line follower implementation



```
readoutleft = readout/16;           // separate left and right side of line sensor
readoutright = readout%16;
if(readout==0)
{
    i = 1000;                       // set i to 1000 -> number of cycles
    while(i > 0)                    // in this loop the forward moving is realized with a PWM signal
    {
        P2->OUT = 0;                // P2.6 and P2.7 set to low -> no signal for left (P2.7) and right (2.6) motor
        __delay_cycles(5);          // wait 5 cycles
        P2->OUT = 0xC0;             // P2.6 and P2.7 set to high -> enable signal for left (P2.7) and right (2.6) motor
        P2->OUT = 0;                // P2.6 and P2.7 set to low -> no signal for left (P2.7) and right (2.6) motor
        i--;                        // decrement cycle counter
    }
}
else if(readoutleft>readoutright)
{
    i = 1000;                       // set i to 1000 -> number of cycles
    while(i > 0)                    // in this loop the left moving is realized with a PWM signal
    {
        P2->OUT = 0;                // P2.6 and P2.7 set to low -> no signal for left (P2.7) and right (2.6) motor
        __delay_cycles(10);         // wait 10 cycles
        P2->OUT = 0xC0;             // P2.6 and P2.7 set to high -> enable signal for left (P2.7) and right (2.6) motor
        P2->OUT = 0x40;             // P2.6 set to high -> enable signal for right (2.6) motor
        __delay_cycles(2);          // wait 2 cycles
        P2->OUT = 0;                // P2.6 set to low -> no signal for right (2.6) motor
        i--;                        // decrement cycle counter
    }
}
```

Line follower implementation



```
else if(readoutleft<readoutright)
{
    i = 1000;                       // set i to 1000 -> number of cycles
    while(i > 0)                    // in this loop the right moving is realized with a PWM signal
    {
        P2->OUT = 0;                // P2.6 and P2.7 set to low -> no signal for left (P2.7) and right (2.6) motor
        __delay_cycles(10);         // wait 30 cycles
        P2->OUT = 0xC0;             // P2.6 and P2.7 set to high -> enable signal for left (P2.7) and right (2.6) motor
        P2->OUT = 0x80;             // P2.7 set to high -> enable signal for left (P2.7) motor
        __delay_cycles(2);          // wait 2 cycles
        P2->OUT = 0;                // P2.7 set to low -> no signal for left motor
        i--;                        // decrement cycle counter
    }
}
else if(readoutleft==readoutright)
{
    i = 1000;                       // set i to 1000 -> number of cycles
    while(i > 0)                    // in this loop the forward moving is realized with a PWM signal
    {
        P2->OUT = 0;                // P2.6 and P2.7 set to low -> no signal for left (P2.7) and right (2.6) motor
        __delay_cycles(5);          // wait 5 cycles
        P2->OUT = 0xC0;             // P2.6 and P2.7 set to high -> enable signal for left (P2.7) and right (2.6) motor
        P2->OUT = 0;                // P2.6 and P2.7 set to low -> no signal for left (P2.7) and right (2.6) motor
        i--;                        // decrement cycle counter
    }
}
```

Final code for following a line on the ground

Source code robot line follower

- You can copy and paste the code to your CCS Cloud project and hit “Run”

```
/* TI_RSLK_LineFollwer
 * T. Lorenzen
 * 08/17/20
 * Code for TI RSLK follwing the black line on the ground
 * with its 8 photo transistors it can distinquish bright/dark surfaces
 * v1.0 08/17/20 Thorsten
 */

#include "msp.h"
#include <stdint.h>
unsigned long i;
unsigned int time=1000;
unsigned long readout=0;
unsigned long readoutleft=0;
unsigned long readoutright=0;

int main(void) {                    // this code checks the line sensor signal and decides to activate the left
motor, the right motor, or both
   volatile uint32_t i;             // It should move forward turn left, turn right and go back to start position

   WDT_A->CTL = WDT_A_CTL_PW |               // Stop WDT
            WDT_A_CTL_HOLD;

   P5->DIR = 0x38;                     // P5.4 and P4.5 motor drive as output and set P5.3 IR LED output driver
   P2->DIR = 0xC0;                     // P2.6 and P3.7 as output set to output driving the motor
   P3->DIR = 0xC0;                     // P3.6 and P3.7 as output to control motor deriver sleep mode
   P5->OUT = 0x00;                      // P5.4 and P5.5 set to low -> set direction to forward for left (P5.4) and
right (5.5) motor
   P3->OUT = 0xC0;                      // P3.6 and P3.7 set to high -> disable sleep mode for left (P3.7) and right
(3.6) motor
   P2->OUT = 0x00;                      // P2.6 and P2.7 set to low -> no signal for left (P2.7) and right (2.6) motor
   P9->DIR = 0x04;                     // P9.2 LED driver control set to output driver
   P9->OUT = 0x00;                      // P9.2 clear LED driver control

   while(1)
   {                                // read line sensor signal after charging the caps and waiting for 1ms.
      P5->DIR = 0x38;                  // set P5.3 (IR LED) as output
      P5->OUT = 0x08;                   // turn on P5.3 (IR LED)
      P9->OUT = 0x04;                   // turn on P9.2 (IR LED)
      P7->DIR = 0xFF;                  // P7.0 to P7.7 (IR Sensors) as output
      P7->OUT = 0xFF;                   // turn on P7 to charge caps
```

Useful links

Order your TI-RSLK MAX
http://www.ti.com/tool/TIRSLK-EVM

Publish your TI-RSLK MAX project
https://texasinstruments.hackster.io/

Use the curriculum for further learning
https://university.ti.com/en/faculty/ti-robotics-system-learning-kit/ti-rslk-max-edition-curriculum

Programming with CCS cloud:
https://dev.ti.com/

Support for TI-RSLK MAX
http://e2e.ti.com/

Follow us on social media
https://twitter.com/txinstruments
http://www.ti.com/facebook
http://www.ti.com/linkedin
https://www.instagram.com/txinstruments/
https://www.youtube.com/texasinstruments