12: Farbsensor am IIC-Bussystem

Sie vertiefen Ihre Kenntnisse zum IIC-Busprotokoll und können das IIC-Controller Modul des MC9S08JM60 zur Kommunikation mit anderen Busteilnehmern einsetzen, insbesondere dem Farbsensor.

1. HSV-Farbraum

Der Farbsensor des MC-Cars liefert die Farbinformation als RGB-Werte. Um eine bestimmte Farbe zu detektieren, müsste somit für drei Farbwerte (Rot, Grün, Blau) überprüft werden, ob sie in einen bestimmten Toleranzbereich fallen. Diese Toleranzbereiche sind sowohl abhängig von der Streuung der Sensoren als auch von Umgebungslichteinflüssen. Deshalb ist die Detektieren von Farben in anderen Farbräumen oft einfacher. In dieser Übung wird das HSV-Modell1 verwendet, dass sich als Zylinder repräsentieren lässt, siehe Abbildung 1.

Die 3 verwendeten Parameter besitzen dabei folgende Bedeutung:

- H = Farbton (hue) 0 … 360° 0° = Rot, 120° = Grün, 240° = Blau, Abbildung 2
- S = Farbsättigung (saturation) 0 … 100% 0% = Neutralgrau, 100% reine Farbe
- V = Hellwert (value) 0 … 100% 0% = keine Helligkeit, 100% = höchste Helligkeit

Die Zylinderachse entspricht der zentralen Graulinie mit 0% Farbsättigung und hat per Definition den Farbton H = 0 (Rot). Punkte am Aussenrand des Zylinders haben 100% Farbsättigung, wobei die Helligkeit zunimmt, je weiter oben sich der Punkt im Zylinder befindet.

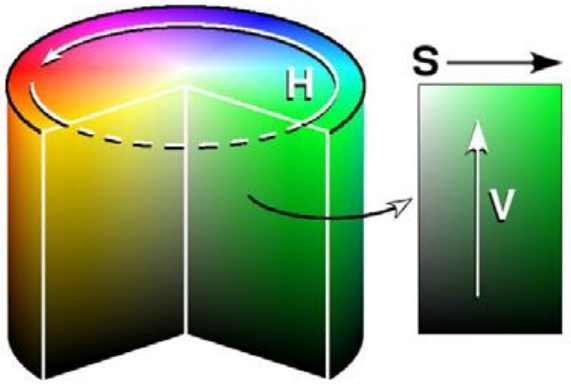

Abbildung 1 – HSV-Farbraum
Quelle: http://de.wikipedia.org/wiki/HSV-Farbraum

Abbildung 2 – Farbton als Winkel 0° … 360°

Der Vorteil des HSV-Modells kommt zum Tragen, wenn man eine bestimmte Farbe unabhängig von deren Helligkeit und Sättigung detektieren möchte. Um beispielsweise Grün zu erkennen, braucht man nur einen einzelnen Wert (H) auf Werte zwischen 100 und 140 zu prüfen. Dieser Vorteil rechtfertigt den Rechenaufwand für die Konvertierung von RGB zu HSV. Diese Umrechnung ist in der Funktion colSensRGBtoHSV() bereits fertig implementiert.

2. Kalibrierung

Um optimale Werte zu erreichen, sollten der Farbsensor und der Liniensensor kalibriert werden.

Die Kalibrierung wird nicht gespeichert, muss also nach jedem Reset/Einschalten durchgeführt werden. Falls das Umgebungslicht stark ändert (z.B. durch Sonneneinstrahlung), kann es auch zwischendurch notwendig sein, die Kalibrierung erneut auszuführen. Für die Kalibrierung geht man wie folgt vor:

1. MC-Car einschalten und Programm laden (falls nicht im Flash)
2. MC-Car auf eine schwarze Fläche stellen und anschliessend den Joystick-Taster kurz nach links betätigen. Hinweis: Unbedingt zuerst Schwarz kalibrieren und erst danach Weiss (wichtig bei der Berechnung der Threshhold-Werte für die Linienerkennung)!
3. MC-Car auf eine weisse Fläche stellen und anschliessend den Joystick-Taster kurz nach rechts betätigen.
4. Kalibrierung fertig, Farben werden zuverlässig erkannt

3. IIC-Farbsensor

Legen Sie in CW ein neues C-Projekt mit Copy/Paste an und nutzen Sie das gegebene File main.c als Hauptdatei. Fügen Sie dem neuen Projekt unter Project_Sources ausserdem die Datei colSens_temp.c zu.

Aktualisieren Sie die Bibliothek MC_Library mit den gegebenen linesens.h/c Dateien.

a) Informieren Sie sich im Datenblatt TCS3414CS.pdf über die prinzipielle Funktionsweise des Farbsensors, seine IIC-Schnittstelle und seine Register (insbesondere Command, Control und ADC Data Register).

b) Analysieren Sie den gegebenen Code und vervollständigen Sie im File colSens_temp.c die Funktionen
   1. colSensInit()
   2. colSensUpdateGain()
   3. colSensReadRawColor()

   unter Verwendung der IIC-Funktionen i2cWriteCmdData() und i2cReadCmdData() aus i2c.c (siehe CW Task-View).

c) Testen Sie Ihre Implementierung in dem Sie den MC-Car über Testfelder.png auf dem Bildschirm ihres Notebooks bewegen und dabei die wechselnden Farben der Front-LEDs beobachten.

d) Implementieren Sie in main.c andere farbabhängige Aktionen (z.B. Soundausgabe bei Rot, Beschleunigung bei Grün) für die Fahrt auf einem Parkour folgender Art:

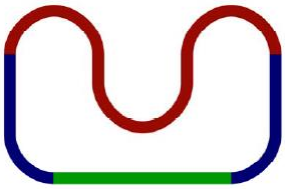

Abbildung 3 - Parkour

colSens.c

```
[…]
#define I2C_COLSENS_ADR                 0x72
[…]
/**
 * Power on and ADC enable through write of appropriate bits in control register
 * Structure of communication protocol:
 * Start Condition | Byte 1 | Akn | Byte 2 | Akn | Byte 3 | Akn | Stop condition
 * contents of the transfered data
 * Byte 1           | Byte 2        | Byte 3
 * I2C Address      | Command       | Data
 * Device Adr+W     | commandReg    | Values for ControlStatusReg
 * 0x72             | 0x80          | 0x03
 *
 * @returns
 *    EC_SUCCESS or an error code if the device has not responded
 */
tError colSensInit(void)
{
  tError error;
  tCommandReg commandReg;
  tControlStatusReg controlStatusReg;

  // set default value
  whiteValue.red = whiteValue.green = whiteValue.blue = 32000;
  blackValue.red = blackValue.green = blackValue.blue = 0;

  // enable "under floor lightning"
  PTCDD_PTCDD2 = 1;
  PTCD_PTCD2 = 1;

  // @ToDo 1: Power on and enable ADC for color sensor TCS3414
  // Power on and enable ADC for color sensor TCS3414
  commandReg.Byte = 0;              // Suppress compiler warning C5651;
  commandReg.Bits.CMD = 1;          // Select command register. Must write as 1.
  commandReg.Bits.Transaction = REG_CMD_TRANS_BYTE_PROT; // use of Byte protocol for I2C
  commandReg.Bits.Address = REG_ADR_CONTROL;             // Select register for following R/W

  controlStatusReg.Byte = 0;        // Suppress compiler warning C5651;
  controlStatusReg.Bits.Power = 1;  // power on
  controlStatusReg.Bits.AdcEn = 1;  // enable ADC

  error = i2cWriteCmdData(I2C_COLSENS_ADR, commandReg.Byte, &controlStatusReg.Byte, 1);
  if (error) return error;
```

```
  // configure Gain Register in color sensor TCS3414 to the most sensitive setting
  gainReg.Bits.Gain = 3;
  gainReg.Bits.Prescaler = 0;
  return colSensUpdateGain();
}
```

[…]

```
/**
 * Sets Prescaler Mode and Gain in the Gain Register of TCS3414
 * (Transaction Type = Byte protocol)
 *
 * @returns
 *    EC_SUCCESS or an error code if the device has not responded
 */
tError colSensUpdateGain(void)
{
  tCommandReg commandReg;

  // @ToDo 2: Configure Gain Register
  commandReg.Byte = 0;       // Suppress compiler warning C5651;
  commandReg.Bits.CMD = 1;   // Select command register.  Must write as 1.
  commandReg.Bits.Transaction = REG_CMD_TRANS_BYTE_PROT; // use of Byte protocol for I2C
  commandReg.Bits.Address = REG_ADR_GAIN;                // select register for following read

  return i2cWriteCmdData(I2C_COLSENS_ADR, commandReg.Byte, &gainReg.Byte, sizeof(tControlGainReg));
}
```

[…]

```
/**
 * Reads the raw 16-bit RGB color values from the sensor TCS3414
 * in a single IIC-Read transfer.
 * (Transaction Type = Block protocol)
 *
 * @param [out] pColor
 *   pColor->red
 *   pColor->green
 *   pColor->blue
 *   pColor->clear
 *
 * @return
 *   EC_SUCCESS if it was successful, an error code otherwise
 */
tError colSensReadRawColor(tColorRawRGB *pColor)
{
  tError error;
  tCommandReg commandReg;

  // @ToDo 3: Read ADC Data Registers
  commandReg.Byte = 0;       // Suppress compiler warning C5651;
  commandReg.Bits.CMD = 1;   // Select command register.  Must write as 1.
  commandReg.Bits.Transaction = REG_CMD_TRANS_BLOCK_PROT; // use of Block protocol for I2C
  commandReg.Bits.Address = REG_ADR_DATA_GREEN;           // select register for following read

  error = i2cReadCmdData(I2C_COLSENS_ADR, commandReg.Byte, (char*)pColor, sizeof(tColorRawRGB));
  if (error) return error;

  // Note the byte ordering in the ADC Channel Data Registers!!
  swapByteOrder(&pColor->red);
  swapByteOrder(&pColor->green);
  swapByteOrder(&pColor->blue);

  return EC_SUCCESS;
}
```

[…]