**Masterarbeit**

# EVALUIERUNG VON LOW-POWER KAMERAMODULEN MIT EINEM MIKROCONTROLLER

ausgeführt am


Fachhochschul-Masterstudiengang
Automatisierungstechnik-Wirtschaft

von
**Ing. Roman Ruthofer, BSc**
1610322003

betreut und begutachtet von
Dipl.-Ing. Dieter Lutzmayr

Graz, November 2017 .....

Unterschrift

# EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

.....

Unterschrift

# DANKSAGUNG

An dieser Stelle möchte ich mich bei all jenen bedanken, die mich während der Anfertigung bei dieser Masterarbeit unterstützt haben und mir beigestanden sind.

Daneben gilt ein außerordentlicher Dank meiner Familie und meinen Freunden, die immer mit Verständnis und Zuspruch einen großen Teil zu dieser Arbeit beigetragen haben und ohne deren Unterstützung ein Masterstudium an der FH CAMPUS 02 nicht möglich gewesen wäre.

Ein besonderer Dank geht an meinen Betreuer Herrn Dipl.-Ing. Dieter Lutzmayr, der mich seit der Ideenfindung für diese Arbeit begleitet hat und während der Erstellung immer ein hilfreicher Ansprechpartner für all meine Anliegen war. Dankend hervorzuheben ist auch die großzügige Bereitstellung und Leihe aller benötigen Komponenten durch die FH CAMPUS 02, um diese Arbeit zu ermöglichen.

Nicht zuletzt geht ein Großteil meines Dankes auch an meine Studienkollegen und Lektoren, die mich im Laufe meines Studiums immer hilfreich und bereichernd begleitet haben.

# KURZFASSUNG

Kamerasensoren werden in der heutigen Zeit nicht nur für Fotoapparate verwendet, sondern sind in einer Vielzahl von Smart-Devices, zum Beispiel Smartphones, Tablets und Smartwatches verbaut. Diese Vielzahl an Bildverarbeitungssensoren legt eine Verwendung neben den ersichtlichen Aufgabengebieten wie Foto- und Video-Anwendungen nahe. So kann mithilfe einer entsprechenden Softwareanwendung Bildverarbeitung, Textverarbeitung, das Lesen von Barcodes oder Gesichtserkennung durchgeführt werden.

Ziel der Arbeit ist es den Energieverbrauch von Kameramodulen, die mithilfe eines Mikrocontrollers angesteuert werden, zu senken und energieverbrauchende Faktoren aufzuzeigen. Um dies zu realisieren wurde ein Überblick über den Zusammenhang von Funktion, Auflösung, Framerate und Energieverbrauch von mehreren Kameramodulen erstellt und die Ansteuerungssoftware für die Module adaptiert.

Das Resultat der Arbeit zeigt einen eindeutigen Zusammenhang des Energieverbrauchs der Kameramodule mit den gelieferten Bildraten und einen messbaren aber nicht markanten Zusammenhang zwischen den einzelnen Auflösungen wie QQVGA, QVGA und VGA.

Aufgrund fallender Preise und der steigenden Anzahl an verbauten Kameramodulen werden Bildauswertungen immer häufiger zum Einsatz kommen. Die Einbindung solcher Kameramodule als Low-Power-Applikation mit einer durchgehenden Bildauswertung ist mit heutigen technischen Mitteln durchaus umsetzbar. Es sollte jedoch bedacht werden, dass die Module durch eine Optimierung der Bildrate und Reduzierung der Bildinformation / Auflösung in entsprechend, energiesparende Zustände versetzt werden müssen.

# ABSTRACT

Nowadays, camera sensors are not only used within digital cameras, but also within many other smart devices, such as smartphones, tablets, and smartwatches. The large amount of available image processing sensors suggests that these can be utilized in addition to the usual tasks such as photo and video applications. For example, a suitable software application can be used to perform image processing, word processing, barcode reading or face recognition.

The aim of the work is to reduce the power consumption of several camera modules using a microcontroller and to point out their energy consuming factors. To this end, an evaluation of the relationship between function, resolution, frame rate and energy consumption of several camera modules was carried out including the adaption of the control software.

The result of the work shows a clear correlation between the energy consumption of the camera modules with the delivered frame rates and a measurable but not significant correlation between the individual resolutions such as QQVGA, QVGA and VGA.

Due to falling prices and the increasing number of used camera modules, image analysis will become more and more important. The integration of such camera modules as a low-power application with continuous image evaluation can be implemented with today's technical means without problems. However, it should be considered that the modules should be put into corresponding energy-saving states by optimizing the frame rate and reducing the image information / resolution.

# INHALTSVERZEICHNIS

1 Einleitung ... 1
1.1 Motivation ... 1
1.2 Ziele ... 1
2 Theoretische Grundlagen ... 2
2.1 Kamerasensoren ... 2
2.1.1 Grundlagen ... 2
2.1.2 CCD-Sensor ... 4
2.1.3 CMOS-Sensor ... 6
2.1.4 Shutter Type ... 8
2.1.5 Bildauflösung und Pixeldichte ... 9
2.1.6 Camera Parallel Interface als Schnittstelle ... 11
2.1.7 Aufbau des Kamerasensormoduls ... 12
2.2 Mikrocontroller ... 14
2.2.1 Grundlegendes ... 14
2.2.2 Allzweckeingabe / -Ausgabe ... 17
3 Software ... 20
3.1 Programmiersprache C ... 20
3.2 Low-Power-Programmieransätze ... 22
3.3 Datenerfassung über das Camera-Parallel-Interface ... 23
4 Auswahl der Komponenten ... 25
4.1 Mikrocontroller ... 25
4.2 Kameraschnittstellen ... 26
4.3 Kameramodule ... 27
5 Hardware Auswahl ... 28
5.1 STM32L496G-Discovery Board ... 28
5.1.1 Übersicht ... 28
5.1.2 Digital Camera Interface ... 30
5.1.3 Direct Memory Access Controller ... 32
5.1.4 IDD – Messung ... 32
5.2 Kameramodule ... 33
5.2.1 OV9655 ... 33
5.2.2 OV2640 ... 34
5.2.3 OV7670 ... 35
6 Auswertung der Kameramodule ... 36
6.1 Beschreibung des Aufbaus ... 36
6.1.1 Hardware Testaufbau ... 36
6.1.2 Hardware Leistungsmessung ... 39
6.1.3 Software Ansteuerung ... 40
6.1.4 Bestimmung der Bildrate ... 44

6.1.5 Bildauswertung ... 45
6.2 Allgemeiner Vergleich ... 49
6.3 Energieverbrauch ... 50
6.3.1 OV2640 Variante 1 ... 50
6.3.2 OV2640 Variante 2 ... 51
6.3.3 OV9655 ... 52
6.3.4 OV7670 ... 53
6.4 Bildvergleich ... 54
6.4.1 QQVGA ... 54
6.4.2 QVGA ... 54
6.4.3 VGA ... 55
7 Beispielapplikation ... 57
7.1 Sensorauswahl ... 57
7.2 Beschreibung der Funktion ... 57
7.3 Beschreibung des Aufbaus ... 59
7.3.1 Hardware ... 59
7.3.2 Software ... 59
7.4 Energieverbrauch ... 63
8 Ergebnisse und Ausblick ... 64
8.1 Bewertung der Sensoren ... 64
8.2 Bewertung der Beispielapplikation ... 65
8.3 Zusammenfassung ... 66
8.4 Ausblick ... 68
Literaturverzeichnis ... 69
Abbildungsverzeichnis ... 72
Abkürzungsverzeichnis ... 77

# 1 EINLEITUNG

## 1.1 Motivation

Kamerasensoren werden in der heutigen Zeit nicht nur für Fotoapparate verwendet, sondern sie sind in einer Vielzahl von Smart-Devices wie zum Beispiel Smartphones, Tablets und Smartwatches als Kameramodule verbaut. Diese Vielzahl an Bildverarbeitungssensoren legt eine Verwendung neben den typischen Aufgabengebieten wie Foto und Video Anwendungen nahe. So kann mithilfe von Software Bildverarbeitung, Textverarbeitung, das Lesen von Barcodes oder Gesichtserkennung ermöglicht werden.

Als großer Nachteil dieser Anwendungen gilt aber der hohe Energieverbrauch, der bei der Aktivierung des Kameramoduls beziehungsweise bei der Bildverarbeitung entsteht. Dies hat zur Folge, dass die Auswertungen im mobilen Bereich zu einer starken Verkürzung der Batterielaufzeit führen. Aus diesem Grund sind Kameraauswertung im mobilen Bereich zwar technisch möglich, werden im Allgemeinen aber vermieden oder so kurz wie möglich gehalten, um den Energieverbrauch zu senken.

Die Motivation dieser Arbeit ist die oben beschriebenen Grenzen und Probleme der Bildverarbeitung besonders im mobilen Bereich durch die Auswahl von geeigneten Komponenten und Softwarelösungen zu verbessern um dieser Technologie zu einem weiteren Fortschritt zu verhelfen.

## 1.2 Ziele

Ziel der Arbeit ist es, einen Überblick über den Zusammenhang von Funktion, Auflösung, Framerate und Energieverbrauch mehrerer Kameramodule mithilfe eines Mikrocontrollers zu erstellen und aufzuzeigen.

Es soll eine Evaluierung von kostengünstigen und energiesparenden Kameramodulen mithilfe eines Mikrocontrollers in Bezug auf Funktion, Auflösung, Framerate und Energieverbrauch erstellt werden. Der Betrieb im Snapshot-Moduls (Einzelbilder) und bei niedrigen Bildraten soll dabei berücksichtigt werden.

Bezugnehmend auf das Ergebnis der Evaluierung soll ein passender Kamerasensor ausgewählt und im Sinne von Ultra-Low-Power mit einem Mikrocontroller in einer Beispielapplikation angesteuert und ausgewertet werden.

# 2 THEORETISCHE GRUNDLAGEN

## 2.1 Kamerasensoren

### 2.1.1 Grundlagen

Kamerasensoren, wie sie in Fotoapparaten, Kameras und Mobiltelefonen Anwendung finden, sind halbleiterbasierende Bauelemente, die nach dem Wirkungsprinzip des inneren photoelektrischen Effektes arbeiten. Der innere photoelektrische Effekt beschreibt das Anheben von Elektronen in einem Halbleiter vom Valenzband in das Leitungsband durch Absorption der Energie von eingestrahlten Photonen. Da Elektronen in einem Atom nur diskrete Energiezustände aufweisen können, muss die Energie des Photons mindestens die Energie der Energiebandlücke $E_g$ erreichen, damit das Elektron ausreichend angeregt wird, um in das Leitungsband zu springen. Elektronen, die so in das Valenzband des Halbleiters gehoben werden, erzeugen ein Defektelektron (Elektronenloch), welches sich gleich wie das Elektron vom Leitungsband frei im Halbleiter bewegen kann. Dieser Elektronenfluss kann als elektrischer Strom oder Ladung gemessen werden und verhält sich proportional zur Energie (Wellenlänge) des einfallenden Lichts.[1]

Um das Elektron in das Valenzband anzuheben, gilt hierbei:

$$Eph = \frac{h * c}{\lambda} > Eg \qquad (2\text{-}1)$$

$E_{ph}$/J = Energie des Photons
$h$/Js = Planck'sches Wirkungsquantum, 6,626*10^-34 Js
$c$/ms$^{-1}$ = Lichtgeschwindigkeit
$\lambda$/m = Wellenlänge des Photons
$E_g$/J = Energie der Bandlücke

Ist die eingestrahlte Energie des Photons auf den Halbleiter jedoch zu groß, wird das Elektron im Leitungsband angehoben, wobei die Überschussenergie dabei vom Gitter absorbiert wird. Dieser Vorgang wird als Thermalisierung bezeichnet, es entsteht wärme. Falls die Energie des Photons aber ein Elektron über das Leitungsband anhebt spricht man nun vom äußeren photoelektrischen Effekt, wobei das Elektron emittiert wird: Dieser Effekt findet in Kamerasensoren keine Anwendung.[2]

[1] Vgl. D. Durini/D. Arutinov (2014), Kapitel 1.4; 3.2.

[2] Vgl. ebd., Kapitel 1.4; 3.2.

Abbildung 1 zeigt den inneren photoelektrischen Effekt. Die absorbierte Energie des eintreffenden Photons hebt das Elektron vom Valenzband in das Leitungsband. Es ist auch ersichtlich, dass das Elektron im Leitungsband überschüssige Energie verlieren kann, diese Thermalisierungs-Verluste entstehen durch kinetische Energie des Elektrons, welche über der kinetischen Energie des Gitters liegt. Diese kinetische Energie wird innerhalb kürzester Zeit an das Halbleitergitter übergeben, dies resultiert in einer Erwärmung des Halbleiters.[3]

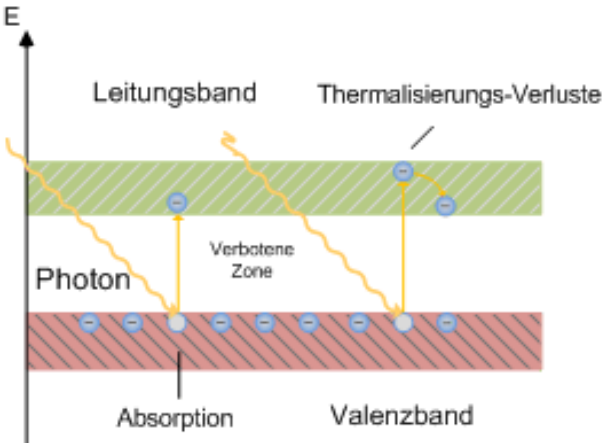


Abbildung 1: Innerer photoelektrischer Effekt (leicht modifiziert); Quelle: https://photovoltaiksolarstrom.com/photovoltaiklexikon/leitungsband-und-valenzband/ [Datum 06.08.2017]

Abbildung 2 beschreibt den äußeren photoelektrischen Effekt. Die Energie „E“ des Photons, welche sich aus dem Planck'schen Wirkungsquantum „h“ und der Frequenz „f“ des Lichts bildet, führt beim Zusammenstoß mit einem Elektron zu dessen Emission. Damit das Elektron emittiert wird, muss die Energie des Photons das Elektron über das Valenzband heben, gleichzeitig wird auch ein Defektelektron gebildet.[4]

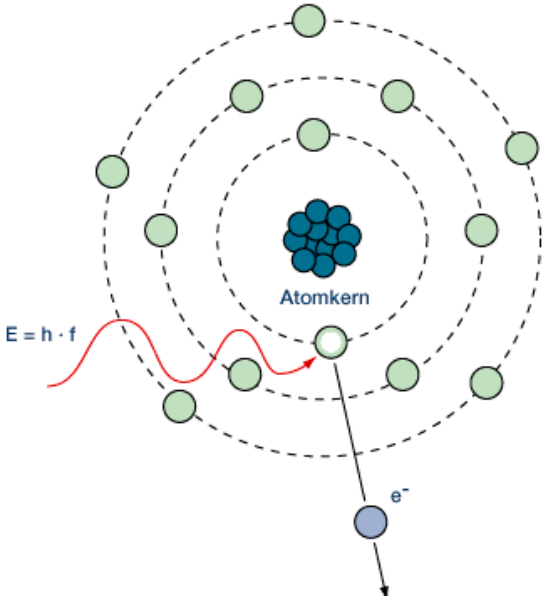


Abbildung 2: Äußerer photoelektrischer Effekt (leicht modifiziert); Quelle: http://www.gironimo.org/wissenschaft/photoelektrische-effekt.html [Datum 06.08.2017]

[3] Vgl. Neges (2003), Online-Quelle [06.08.2017], Kapitel 1.

[4] Vgl. D. Durini/D. Arutinov (2014), Kapitel 1.4.

### 2.1.2 CCD-Sensor

Der Sensortyp CCD steht für *„Charged Coupled Device“* und wurde um 1970 als einer der ersten Kamerasensoren von der Firma Bell Telephone Laboratories vorgestellt. Ab 1980 war die CCD-Technologie der am weit verbreitetste halbleiterbasierte Sensortyp und fand vor allem in der Forschung und Industrie ihre Anwendung. Wie der Name *„Charged Coupled Device“* schon verrät, handelt es sich beim CCD-Aufbau um die Aneinanderreihung von geladenen Bereichen. Jeder geladene Bereich ist ein Photoelement, dass durch einstrahlende Photonen geladen, bzw. in der die Ladung aufintegriert wird. Beim Aufbau unterscheidet man zwischen vier grundlegenden Varianten, dem full frame CCD, frame transfer CCD, interline transfer CCD und dem orthogonal transfer CCD, wobei alle CCD-Varianten gemeinsam haben, dass jede Ladung hintereinander und nicht gleichzeitig von einem Analog-Digitalwandler mit Verstärker ausgelesen wird. Es sind auch Kombinationen der vier Typen möglich. Nach Belichtung des Sensors werden alle Ladungen durch ein serielles Register geschoben und ausgelesen, der CCD-Sensor kann erst wieder belichtet werden, wenn seine Ladungen geleert worden sind, da die vorherige Lichtmessung ansonst verfälscht werden würde. Das Wirkungsprinzip des Sensors lässt sich sehr gut stellvertretend anhand des interline transfer CCD beschreiben, welches in Abbildung 3 (c) ersichtlich ist. Bei dieser Type werden nach Belichtung des Photoelements (Photosites) die angehäuften Ladungen der Elemente in ein Shift-Register verschoben, welches wiederum in einem vorgegebenen Takt seine Ladungen nacheinander im Shift-Register nach unten schiebt und anschließend in ein serielles Register schreibt. Es ist hierbei wichtig, dass bis der vollständige Schiebevorgang vom Shift-Register abgeschlossen ist, keine neuen Ladungen vom Photoelement in das Shift-Register übertragen werden, dies würde ansonsten den Ladungswert im Register ungültig erhöhen. Im seriellen Register werden die Ladungen einzeln zu einem Analog-Digitalwandler mit Verstärker geschoben, der den Messwert anschließend als Grauwert interpretiert. Die Rückrechnung von einem Messwert am AD-Wandler zum ursprünglichen Photoelement erfolgt über den Takt in einer anschließenden Signalauswertung durch einen Mikrocontroller. Aus den oben genannten Zusammenhängen ergibt sich, dass die Taktraten der Register um ein Vielfaches höher sind als die Taktraten, mit der die Register vom Sensor beschrieben werden.[5]

Der Aufbau des full frame CCD-Sensors Abbildung 3 (a) kommt ohne Shift-Register aus. Dies funktioniert dadurch, dass die Ladungen von einem Photoelement in das darunterliegende Register geschoben werden. Der Sensor darf durch diesen Aufbau während der gesamten Verschiebung der Ladungen nicht belichtet werden und braucht somit einen Blendenverschluss, der erst nach dem Auslesen des letzten Elements geöffnet werden darf. Der full frame CCD-Sensor ist die einfachste und günstigste Variante aller CCD – Typen. Der frame transfer CCD-Sensor Abbildung 3 (b) funktioniert ähnlich wie der full frame CCD jedoch benötigt er für jedes aktive Photoelement ein anderes Photoelement, das durch eine Schutzschicht nicht belichtet werden kann (Opaque mask). Eine Blende ist bei diesem Sensor nicht nötig,

[5] Vgl. Lesser (2014), Kapitel 3.

und es werden sehr hohe Leseraten erreicht, jedoch ergibt sich aufgrund seines Wirkungsprinzips eine hohe Anzahl an verschwendeten Photoelementen und somit eine geringere Auflösung des Endbildes.[6]

Der orthogonal transfer CCD-Sensor Abbildung 3 (d) beinhaltet einen sehr speziellen Sensoraufbau, der vor allem bei Teleskopen (Astronomie) Anwendung findet. Er ermöglicht es, die Ladungen für jedes einzelne Photoelement in das obige, darunterliegende oder benachbarte Register zu schieben, um so eine lokale Unschärfe im Bild auszugleichen. Relevanz hat dies bei der Beobachtung von Himmelsbildern, wo atmosphärische Effekte zu unterschiedlicher und sich permanent ändernder Brechung von Licht führen. Am Verbrauchermarkt wird diesem Sensortyp keine Relevanz beigestellt und er findet somit sehr selten Anwendung.[7]

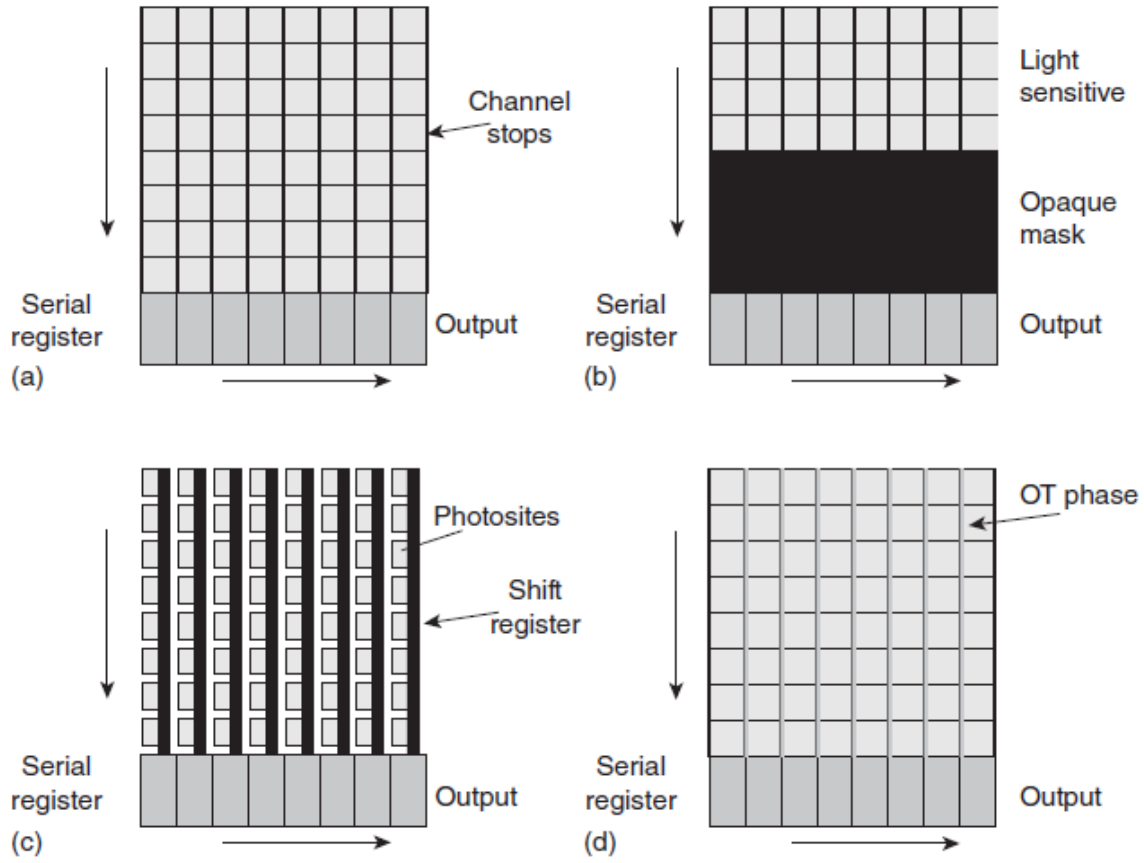


Abbildung 3: (a) full frame CCD,(b) frame transfer CCD, (c) interline transfer CCD, (d) orthogonal transfer CCD.; Quelle: Lesser (2014) Kapitel 3 [Datum 06.08.2017]

[6] Vgl. ebd., Kapitel 3.

[7] Vgl. Lesser (2014) Kapitel 3.

### 2.1.3 CMOS-Sensor

Der elektronische Halbleiterbaustein CMOS steht für „Complementary Metal-Oxide-Semiconductor“ und wurde um 1963 von Frank Wanlass erfunden. Ein CMOS besteht mindestens aus zwei MOSFETS, jedoch können die verwendeten MOSFETS zueinander eine negierte Schaltlogik aufweisen. Unter dieser Bedingung werden die MOSFETS dann als PMOS und NMOS bezeichnet. Bei PMOS spricht man von einem P-Kanal MOSFET, bei einem NMOS von einem N-Kanal MOSFET. Diese unterschiedlichen Eigenschaften werden durch Dotierungen im Halbleitermaterial erreicht. Wenn bei einem CMOS-Inverter-Baustein keine Spannung am Eingang (IN) anliegt, ist der PMOS durchgeschaltet und verbindet Vdd mit dem Ausgang (OUT), der NMOS hat keinen Durchgang. Beim Anliegen einer ausreichend hohen Spannung am Eingang (IN) dreht sich nun das Verhalten dahingehend, dass der PMOS die Verbindung zwischen Vdd und dem Ausgang sperrt, jedoch der NMOS durchschaltet und der Ausgang nun auf Masse (Vss) liegt. MOSFETS haben den Vorteil, dass sie sich sehr gut skalieren lassen und somit hohe Bausteindichten auf einem Waver möglich sind, zudem zeichnen sich MOSFETS mittlerweile durch einen sehr gut beherrschbaren Herstellungsprozess aus.[8]

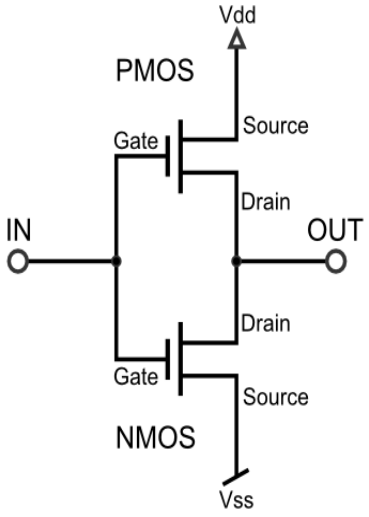


Abbildung 4: CMOS-Inverter Schaltlogik (leicht modifiziert); Quelle: https://upload.wikimedia.org/wikipedia/commons/5/51/CMOS_inverter_model_E.PNG [Datum 06.08.2017]

Die bei einem CMOS-Bildsensor eingesetzte Kombination aus MOSFETS wird als „APS - Active Pixel Sensor“ bezeichnet, wie in Abbildung 5 ersichtlich beinhaltet er eine Verschaltung aus mehreren NMOS-Bausteinen. Mithilfe des Reset NMOS (Mrst) kann bei Aktivierung durch ein Resetsignal (RST) eine aufgebaute Ladung an der Photodiode (PD) durch Anlegen der Spannung (VAA_PIX) vom Photoelement abgezogen werden. Der NMOS Msf agiert in der Schaltung als Pufferverstärker, je nach anliegender Ladung am Gate, generiert durch die Photodiode (PD), fließt ein Strom zwischen Drain und Source. Durch das Schalten von Select (SEL) kann nun die anliegende Ladung an Msf über den NMOS Msel gelesen werden, ohne die Ladung von dem Photoelement abzuziehen. APS-Elemente können hierbei im Aufbau voneinander abweichen, sie beruhen jedoch alle auf dem gleichen Wirkungsprinzip. Die Güte eines APS-Sensors beruht auf der Genauigkeit der MOSFETS, im Speziellen des Msf-Bausteines, welcher auch die Empfindlichkeit des Sensors auf den Lichteinfall bestimmt. [9]

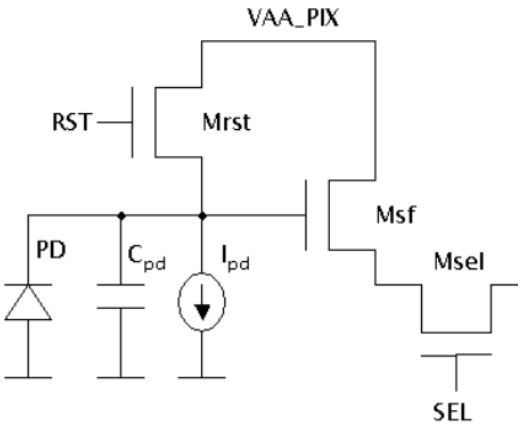


Abbildung 5: Active Pixel Sensor (leicht modifiziert); Quelle: http://www.ee.uidaho.edu/ee/analog/suatay/papers/CMOS_APS_4836_41.pdf [Datum 06.08.2017]

[8] Vgl. Baker (2010), Online-Quelle [14.08.2017].

[9] Vgl. Lesser/Fossum/AY (2002), Online-Quelle [14.8.2017].

Ein CMOS-Kamerasensor besteht nicht nur aus dem APS und einem RGB-Filter, sondern auch aus Leiterbahnen für die Ansteuerung. Bei der Platzierung dieser Leiterbahnen unterscheidet man zwischen einem „Frontseitenbeleuchteten CMOS-Sensor“, bei dem die Leiterbahnen an der Oberseite des Sensors sitzen und einem „Rückseitenbeleuchteten CMOS-Sensor“, bei dem die Photoelemente direkt an der Oberfläche sind. Wie in Abbildung 6 ersichtlich, ergibt sich dadurch ein erheblicher Unterschied des Weges des einfallenden Lichts. So muss bei einem frontseitenbeleuchteten Sensor die ganze Leiterbahnenschicht durchquert werden. Dies hat natürlich eine Minderung der einfallenden Lichtmenge auf die Sensorfläche zur Folge, da viele Photonen von den Leiterbahnen reflektiert werden, jedoch kann das Licht auch unterschiedlich im Medium gebrochen werden und somit das Bildsignal verwischt werden. Beim rückseitenbeleuchteten CMOS-Sensor existieren all diese Probleme nicht, jedoch bedarf es eines sehr schwierigen Herstellungsprozesses, wobei der Waver bei der Herstellung umgedreht werden muss, und somit eine beidseitige Waverbearbeitung vorliegt.[10]

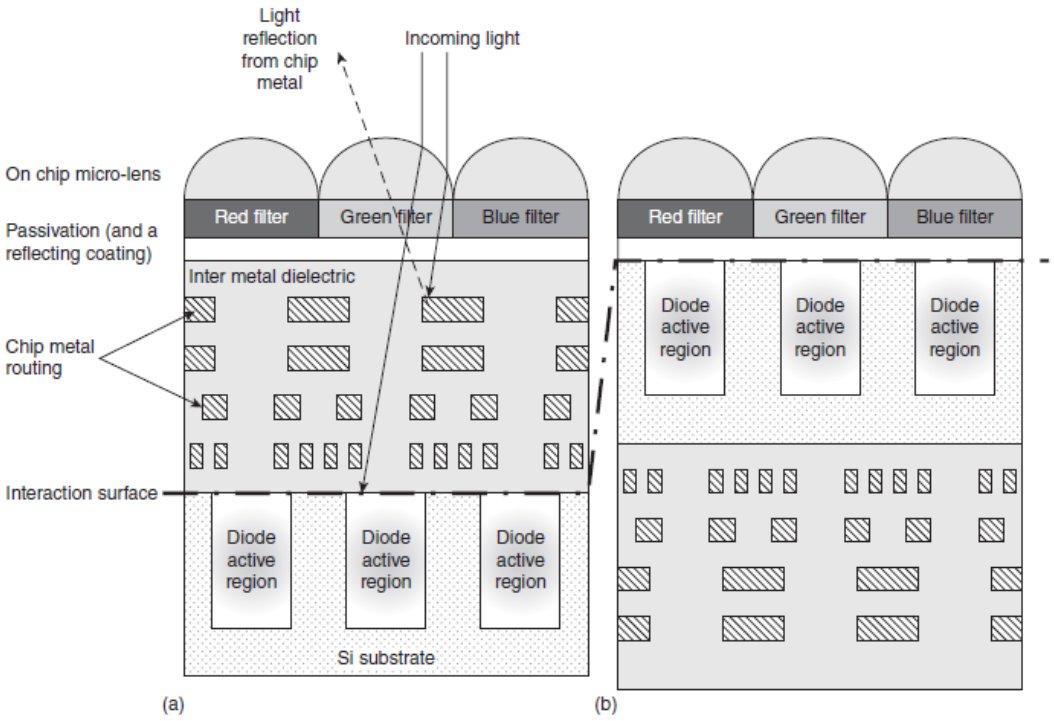


Abbildung 6: (a) Frontseitenbeleuchteter CMOS-Sensor (b) Rückseitenbeleuchteter CMOS-Sensor; Quelle: Lahav/Fenigstein/Strum (2014) Kapitel 4

Die wesentlichen Vorteile eines CMOS-Sensors gegenüber eines CCD-Sensors besteht aus der Möglichkeit die Sensorpixel gleichzeitig auszulesen was in einer höheren möglichen Bildwiderholrate resultiert, bei gleicher Auflösung. Zusätzlich muss nicht der gesamte Bildbereich gelesen werden, sondern es können im Bedarfsfall nur Teil-Bereiche des Sensors adressiert werden. Als Nachteil gegenüber dem CCD-Sensor ist besonders das Rauschen, bemerkbar bei geringer Lichtintensität, hervorzugeben. Dies wird durch die separaten Verstärker pro Pixel verursacht, welche technologisch bedingt, Abweichungen bei der Verstärkung verursachen.[11]

[10] Vgl. Lahav/Fenigstein/Strum (2014).

[11] Vgl. Stemmer Imaging (2018), Online-Quelle [13.1.2018].

### 2.1.4 Shutter Type

Beim Shutter Type unterscheidet man zwischen einem Rolling Shutter und einem Global Shutter, beide beziehen sich darauf, wie die Bildinformation aus den Photoelementen ausgelesen wird. Aufgrund seines Wirkungsprinzips findet man bei einem CCD-Sensor nur den Global Shutter vor, bei einem CMOS Sensor meist, aber nicht immer, den Rolling Shutter. Bei einem Global Shutter wird die Bildinformation aller Photoelemente gleichzeitig ausgelesen. Es wird damit garantiert, dass jeder Bildpunkt zum gleichen Zeitpunkt und bei selber Integralzeit aufgenommen wird. Das gleichzeitige Auslesen wird durch eine Zwischenspeicherung des vollständigen Bildes oder durch Schließen der Belichtungsblende möglich, sodass der Mikrokontroller Zeit hat, ein unverändertes Bild zu verarbeiten. Ein wesentlicher Nachteil von Global Shutter-Sensoren ist jedoch die langsame Bildupdaterate. Im Gegensatz dazu steht der Rolling Shutter. Hier werden die Bildpunkte nacheinander, reihenweise ausgelesen. Beim Auslesen wird Zeile für Zeile zwischengespeichert und anschließend verarbeitet. Bildinformationen von anderen Zeilen können sich in dieser Zeit aber weiter verändern. Dies hat zur Folge, dass jede Zeile, die so gelesen wird, einen anderen Zeitpunkt repräsentiert und man damit bei einem Rolling Shutter Bild von keiner Momentaufnahme mehr sprechen kann. Es ist jedoch zu beachten, dass der Rolling Shutter Effekt nur auftritt, wenn das Bild sich innerhalb der Zeit, welche die Kamera braucht um all ihre Zeilen auslesen zu können, nennenswert verändert. Der Grund für die Verwendung eines Rolling Shutters liegt in dem Wirkungsprinzip des CMOS-Sensors, bei dem meist zeilenweise die APS ausgelesen werden. Der CMOS-Sensor wiederum findet sehr häufig eine Verwendung aufgrund seiner einfachen und kostengünstigen Fertigung im Vergleich zum CCD-Sensor. Aufgrund der weiten Verbreitung der CMOS-Technologie ist bei einem Großteil aller Bildsensoren für Verbraucher mit einem Rolling Shutter-Effekt zu rechnen, im Gegensatz dazu werden in der Industrie CMOS-Sensoren mit einem Global Shutter bevorzugt eingesetzt.[12]

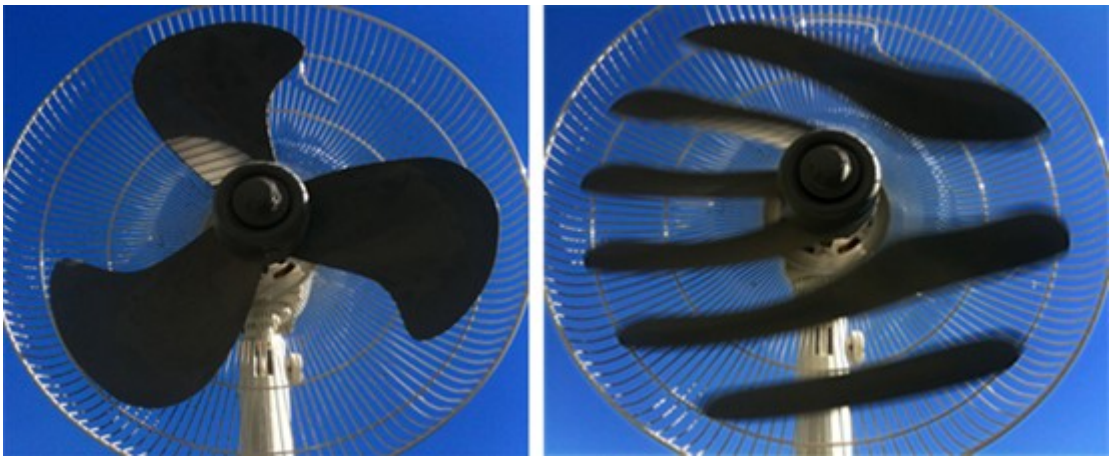

Abbildung 7: Links Global Shutter, Rechts Rolling Shutter; Quelle http://lfa.mobivap.uva.es/~fradelg/phd/figures/global-shutter.png [Datum 06.08.2017]

[12] Vgl. QIMAGING (2014), Online-Quelle [14.08.2017].

### 2.1.5 Bildauflösung und Pixeldichte

Die Bildauflösung ist ein wichtiger Begriff im Bereich der Bildverarbeitung und Darstellung, welcher eine Aussage über die Anzahl an dargestellten Bildpunkten in einem Bild hat. Dafür gibt es die Pixeldichte eines Bildes, sie wird in DPI – Dots per Inch oder PPI – Pixel per Inch angegeben, wobei PPI im digitalen Bereich von Kameras und Bildschirmen eine übliche Angabe ist, DPI hingegen in der Druckbranche als Bezeichnung für die Anzahl an realen Punkten pro Inch auf einem Papier Verwendung findet. Die Pixeldichte eines Bildes trifft keine Aussage darüber, welche Bildgröße oder Anzahl an Pixel im Gesamten vorliegt, sie lässt auch ohne weitere Angabe keine Rückschlüsse zu, ob die Anzahl an Pixel in einer Darstellung ausreichend gewählt worden sind. Die Pixeldichte eines Kamerabildes in DPI ergibt sich einerseits durch die Anzahl an Photoelementen auf dem Kamerasensor, und andererseits durch die gewählte Darstellungsgröße des Bildes auf dem Wiedergabemedium. Bei Kamerasensoren bewegt man sich bei der Pixelanzahl, nach Stand der Technik 2017, in Bereichen von 0,3 MP bis 36 MP. Dies bedeutet, dass für ein Kamerabild $3*10^5$ bis $36*10^6$ Pixel als maximale Bildinformation zu Verfügung steht. Bei der Aufteilung der Pixel in ein Höhe-zu-Breite-Verhältnis spricht man von Bildgrößen wie QHD, VGA, PAL oder NTSC. Um ein Bild in einer VGA-Auflösung darstellen zu können, braucht man zumindest 640 x 480 Pixel, was somit ab einem 0.3 MP Kamerasensor möglich wäre. Setzt man die Bildgröße in Verhältnis zu der Leinwandgröße, auf der die Bildinformation wiedergegeben wird, erhält man wiederum die DPI des Bildes. Bei Darstellung eines VGA aufgelösten Bildes mit 640x480 Bildpunkten erhält man auf einer 640x480 Inch großen Leinwand somit 1 DPI.[13,14]

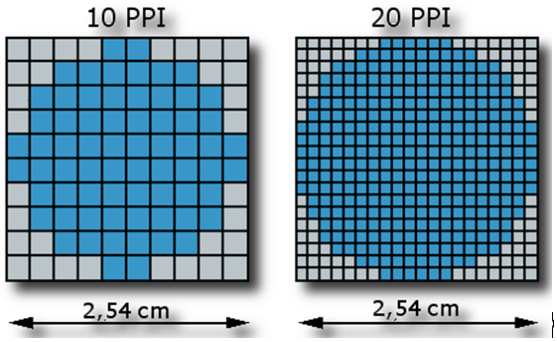


Abbildung 8: Veränderung der PPI bei gleichbleibender Bildgröße; Quelle: http://www.fmedda.com/de/article/dpi_ppi [Datum 06.08.2017]

Bilder, die bei anschließender Darstellung eine höhere Gesamtanzahl an Pixel als der Kamerasensor aufweisen, beinhalten Pixel, die nachträglich durch Interpolation zweiter Bildpunkte in der Bildverarbeitung entstehen und anschließend eingefügt werden. Jedoch haben auch Bilder, die von einem RGB-Kamerasensor aufgenommen worden sind und die gleiche Anzahl an Pixel wie der Sensor besitzen, interpolierte Bildpunkte, dies lässt sich durch die vorgeschaltete RGB-Filter meist als Bayer-Muster erklären. Das Bayer-Muster, Abbildung 9, besteht aus 4 Pixel, wovon ein rotes, ein blaues und zwei grüne in einer fest vordefinierten Arrangierung verwendet werden. Aufgrund dieses Bayer-Musters kann anschließend bei der Bildauswertung eine Umwandlung in einen RGB-Farbwert stattfinden. Die

[13] Vgl. Neuhaus (2004-2017), Online-Quelle [15.08.2017].

[14] Vgl. Foto-Mosaik-Edda (2017), Online-Quelle [15.08.2017].

Umrechnung der Farbfilterlichtwerte zu einem RGB-Wert kann hierbei auf verschiedenen Wege erfolgen. Eine einfache Methode, um dies zu realisieren ist die direkte Zuweisung des roten Farbfilterlichtwertes ($R_{RGB}$ = R), des blauen Wertes ($B_{RGB}$ = B), und bei der Zuweisung des grünen Wertes wird arithmetisch über beide Pixel gemittelt ($R_{RGB}$ = (G1+G2)/2). Diese Methode hat jedoch zur Folge, dass die resultierende Auflösung um die Hälfte kleiner ist als die ursprüngliche, oder in anderen Worten, dass die resultierende Pixelanzahl um das Vierfache kleiner ist als der Schwarz-weiß-Sensor ohne Filter wiedergeben würde. Für einen 4MP-RGB-Sensor mit Bayer-Muster würde also effektiv ein 1-MP-Bild in Farbe zur Verfügung stehen. Um dies zu umgehen oder zu kompensieren, wenden die Hersteller von Kamerasensoren einen Interpolationsalgorithmus an, der anhand von benachbarten Pixel die fehlenden Bildpunkte mit Grauwert und Farbwert errechnet. Die genauen Algorithmen sind zumeist nicht offengelegt und herstellerspezifisch, bewirken aber eine Bildunschärfe, die wiederum korrigiert werden muss. Die Berechnung der fehlenden Bildpunkte findet sehr oft direkt am Kamerasensor/Chip statt, so dass von außen die Bearbeitung nicht ersichtlich ist. Üblicherweise werden lineare, kubische Bildbearbeitungsalgorithmen und der „Nearest-Neighbor-Algorithmus“ sowie Spline-Interpolationen angewandt, um den korrekten Farbwert und die Intensität der interpolierten Pixel zu berechnen. Abbildung 9 zeigt das Bayer-Muster welches als Filter vor dem Bildsensor angebracht ist. Abbildung 10 zeigt links das Bayer-Muster-Bild, welches von den Bildsensoren an die Bildverarbeitung weitergegeben wird, und rechts das Ergebnis der Bildverarbeitung.[15,16,17]

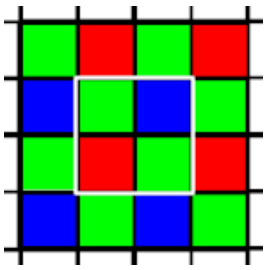

Abbildung 9: Bayer-Muster; Quelle: https://www.uni-muenster.de/ZIV/Lehre/MM_HWK/V001S02.htm [Datum 06.08.2017]

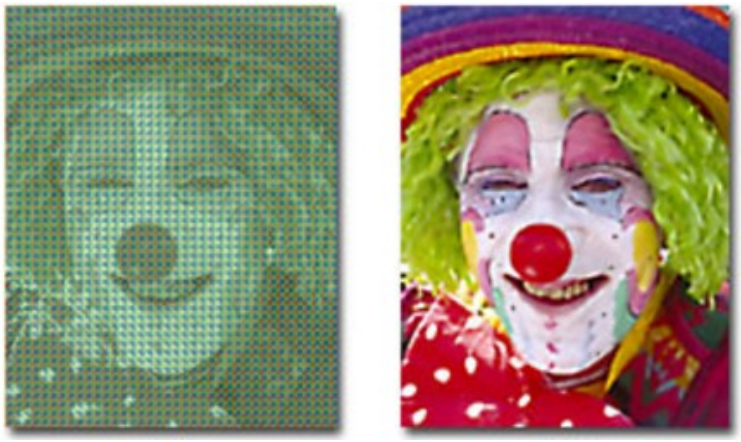

Abbildung 10: Bayer-Muster Rekonstruktion; Quelle: http://www.olympusmicro.com/primer/digitalimaging/cmosimagesensors.html [Datum 03.09.2017]

[15] Vgl. Universität Münster, Online-Quelle [15.08.2017].

[16] Vgl. TU Clausthal (2010), Online-Quelle [15.08.2017].

[17] Vgl. Olympusmicro (2012), Online-Quelle [03.09.2017]

### 2.1.6 Camera Parallel Interface als Schnittstelle

Das Camera-Parallel-Interface – CPI ist eine 8 bis 12 Bit breite Datenleitung, welche die Bildinformation des Bildsensors überträgt. Um diese Bildinformation auswerten zu können, ist neben dem CPI auch eine Steuerleitung, unter anderem I2C, zwingend erforderlich. Die Daten von dem CPI werden üblicherweise in einem 1-2 Byte-Format an den verarbeitenden Mikrocontroller weitergegeben, gängige Datenformate hierbei sind RGB565 (16Bit), RGB555 (15Bit), YUV422 (8Bit) und YCbCr422 (8Bit). Das Camera-Parallel-Interface trifft keine Aussage darüber, in welchem Datenformat die Bildinformationen übertragen wird, dies kann über das I2C Interface definiert oder dem Datenblatt des Sensors entnommen werden. Um die Daten am CPI Bus richtig interpretieren zu können, bedarf es auch zusätzlicher Informationssignale wie Vertical SYNC (VSYNC), Horizontal Reference (HREF) und der Pixel Clock (PCLK), diese Signale werden vom Mikroprozessor im Fotosensor zur Verfügung gestellt. [18]

| Mode | Byte | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|
| RGB565 | First | R4 | R3 | R2 | R1 | R0 | G5 | G4 | G3 |
| | Second | G2 | G1 | G0 | B4 | B2 | B2 | B1 | B0 |
| RGB555 | First | X | R4 | R3 | R2 | R1 | R0 | G4 | G3 |
| | Second | G2 | G1 | G0 | B4 | B2 | B2 | B1 | B0 |

Abbildung 11:CPI mit RGB565 oder RGB555; Quelle: FTDI (2015), Online-Quelle [20.08.2017]

Abbildung 11 zeigt, wie Daten im RGB565- oder RGB555-Format via CPI übertragen werden. Für die Übertragung jedes Bildpunkts, z.B. R0, G0, B0, werden hierbei zwei Takte der Pixel Clock (PCLK) benötigt. Je nach verwendetem Datenformat können unterschiedliche Farbtiefen wiedergegeben werden, bei RGB565 ergeben sich aus den 16 Bit 65.536 mögliche Farbwerte, bei RGB555 aus 15 Bit nur 32.768. Bei Verwendung von 1-Byte-Formaten wie YUV422 kann pro Takt ein vollständiger Bildpunkt übertragen werden, jedoch können nur 256 Farbwerte aufgelöst werden.[19]

Neben dem CPI findet auch das Mobile-Industry-Processor-Interface - MIPI Verwendung, dieser Standard wurde von der MIPI-Alliance gegründet. Mögliche Protokolle der MIPI-Schnittstelle sind CSI-1, CSI-2 und CSI-3, wobei alle im Gegensatz zu CPI eine serielle Schnittstelle verwenden, welche vom Aufbau eine Ähnlichkeit mit SPI hat. MIPI findet vor allem in modernen Mobiltelefonen mit hochauflösenden Kamerasensoren und hohen Datenraten seine Verwendung, da der Standard aber nicht offengelegt wird und Informationen nur für Mitglieder der Allianz zu Verfügung stehen, findet der Standard fast keine Verwendung außerhalb der Großindustrie.[20]

[18] Vgl. FTDI (2015), Online-Quelle [20.08.2017].

[19] Vgl. ebd.

[20] Vgl. ebd.

### 2.1.7 Aufbau des Kamerasensormoduls

Ein digitales Kamerasensormodul kann üblicherweise in drei Teilbereiche gegliedert werden, dem Bildsensor CMOS oder CCD, der Kameralinse und der Einbindungsschnittstelle CPI oder MIPI. Dieses Modul kann von den Sensorherstellern, abgesehen von der Linse, als System-on-a-Chip - SoC-Lösung bezogen werden. Der vereinfachte Aufbau eines gesamten Kamerasensormoduls kann in untenstehender Abbildung 12 betrachtet werden. Es besteht aus einem lichtundurchlässigen Gehäuse, um den Sensor vor unerwünschten Lichtstrahlen zu schützen, einem CCD- oder CMOS-Fotosensor für die Bilderfassung, mehreren Linsen, um das Licht fokussiert auf den Sensor zu bündeln, und aus einer Blende, welche durch Öffnen und Schließen den Strahlengang beschneiden und somit die Schärfentiefe vergrößern oder den Sensor verdunkeln kann. Die Linse in Kombination mit dem Verschluss wird als Objektiv verstanden, wobei abhängig von der Fotosensortechnologie auch starre Verschlüsse mit einer fest eingestellten Blende, verbaut sein können.[21,22]

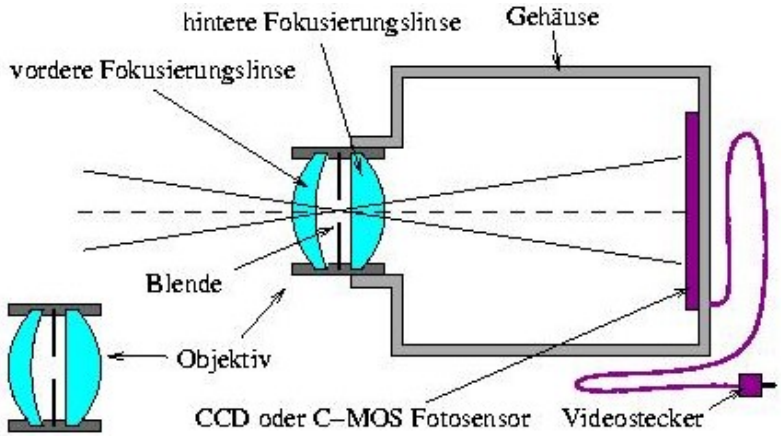


Abbildung 12: Vereinfachter Aufbau einer Kamera; Quelle: http://maurer-tech.com/Debian/v4l/DieKamera.jpg [03.09.2017]

Abbildung 13 zeigt beispielhaft das Funktionsblockdiagramm eines OmniVision CMOS-Sensors OVM7692, der Hersteller verweist hier neben der Linse auf folgende Einteilung, dem „image sensor core, dem „image sensor processor“ und dem „image output interface“. Neben den genannten Einteilungsbereichen findet man am Sensor eine 8-Bit-CPI-(D [7:0]) und MIPI-(MCP, MCN, MDP, MDN) Schnittstelle für die Übertragung des Bildmaterials, sowie ein I2C-Interface (SIOC, SIOD). Das I2C-Interface wird für die Konfiguration der Bildbearbeitung im „image sensor processor“ benötigt, je nach Einstellung können das Datenformat (RGB, YUV, YCbCr) und die Auflösung eingestellt und viele weitere Einstellungen (z.B. Weißabgleich, Bildrate) vorgenommen werden. Die „timing generator and system control logic“ liefert wichtige Taktraten für die Interpretation des Videosignals, dabei spielt Vertical SYNC (VSYNC), Horizontal Reference (HREF) und Pixel Clock (PCLK) eine Rolle. PWDN ist eine Zusatzfunktion und steht für „power down“, sie ermöglicht einen Standbybetrieb. XVCLK ist der System-Clock-Eingang für den OVM7692. Dieser wird extern durch einen Mikrocontroller oder Oszillator konstant vorgegeben, aus ihm leitet das Kamerasensormodul seine internen Taktraten ab. Der „image sensor

[21] Vgl. FTDI (2015), Online-Quelle [20.08.2017].

[22] Vgl. Westphalen (2016).

core“ besteht aus dem CMOS-Bildsensor, einem verstellbaren Verstärker, einem Analog-Digitalwandler und anschließender Bild/Kontrastbearbeitung. Zu beachten ist, dass die Daten die Bildinformation des Bayer-Musters ohne Interpolation repräsentieren. Anschließend werden die Informationen im „image sensor processor“ je nach Einstellung via I2C im Digital Signal Processor – DSP verarbeitet. Die Verarbeitung beinhaltet unter anderem die Interpolation des Bayer-Musters auf die ursprüngliche Auflösung, Filterung und eine Bildkorrektur, anschließend wird das Bild im „formatter“ auf das eingestellte Bildformat, z.B. YUV422, konvertiert und dem „image output interface“ übergeben. In Letzterem findet noch eine Zwischenspeicherung des Bildsignals und die Ausgabe über CPI oder MIPI statt. Je nach Type und Hersteller unterscheiden sich die Funktionen der Kamerasensormodule, der dem OVM7692 entnommene Grundaufbau bleibt aber immer gleich. Eine spezielle Zusatzfunktion des OVM7692 ist z.B. die „50/60 Hz auto detection“, welche ein Bildflimmern, ausgelöst durch eine getaktete Lichtquelle oder Bildwiedergabe, erkennen und kompensieren soll. [23,24]

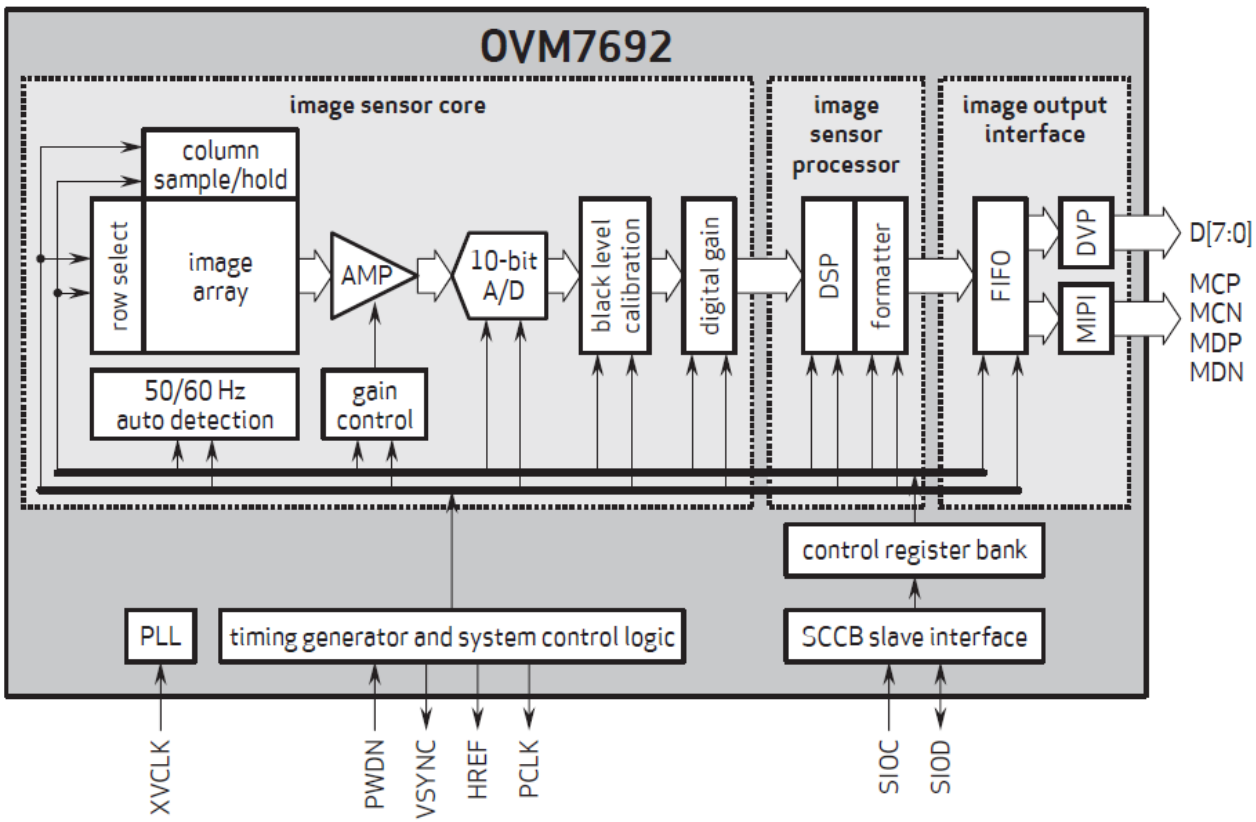


Abbildung 13: OmniVision OVM7692 Funktionsblockdiagramm; Quelle: OVM7692 (2013)

[23] Vgl. FTDI (2015), Online-Quelle [20.08.2017].

[24] Vgl. OV7670 (2005).

## 2.2 Mikrocontroller

### 2.2.1 Grundlegendes

Ein Mikrocontroller ist eine System-on-a-Chip-Lösung, die wesentliche Steuerungs- und Kommunikationselemente in sich vereint, und speziell für ihn geschriebene Programme mithilfe seiner Komponenten in einer Endlosschleife ausführt. Mikrocontroller werden je nach Anwendung speziell ausgewählt und angepasst, und sollten so weit wie möglich alle benötigen Bausteine für die gewählte Applikation bereitstellen. Die meisten µ-Controller bestehen zumindest aus einer CPU, festen und flüchtigen Speicherelementen, externen Kommunikationsschnittstellen, Zähler und Oszillatoren. Diese Bausteine sind über Daten- und Steuerleitung fest miteinander verknüpft. Das Herzstück eines Mikrocontrollers bildet dabei immer seine CPU, diese kann sich separat auf der Platine befinden oder als MCU verbunden mit anderen Bausteinen als Chip vorhanden sein. Moderne Mikrocontroller werden als MCU mit einer Vielzahl an Funktionen angeboten. Befinden sich alle Elemente des Mikrocontrollers auf einem einzelnen Chip, spricht man von einem System-on-Chip.[25]

Der typische Aufbau eines µC kann in Abbildung 14, anhand eines Cortex-M4 betrachtet werden, ein µC besteht aus folgenden Komponenten:

- CPU – sie bildet die Zentrale Recheneinheit des µC und besteht aus einem Speicherbaustein (Puffer), einem Steuerwerk und einem Rechenwerk (ALU), welches für die arithmetischen Berechnungen und logische Operationen zuständig ist. Grundlegend kann man CPUs nach ihrer Designphilosophie unterscheiden. Ist ein schneller und energiesparender Prozessor erwünscht, greift man oft zu einer RISC- (Reduced Instruction Set Computer) Architektur mit wenigen Befehlsätzen. Bei Bedarf eines komplexen Prozessors, welcher eine Vielzahl von Befehlen versteht, spricht man von CISC (Complex Instruction Set Computer). Neben dem Design spielt auch die Datenbreite des Prozessorkerns eine wichtige Rolle. Heutige CPUs haben hier 4 bis 64 Bit, einfache Prozessoren, wie der Intel 8051, haben nur eine geringe Datenbreite von 8 Bit. Die Datenbreite bezieht sich auf die Größe der internen parallelen Datenpfade des Prozessors und hat somit einen wesentlichen Einfluss auf dessen Leistung. Eine weitere wichtige Kennzahl einer CPU ist ihre Taktrate, mit der die Berechnungen und Operationen ausgeführt werden, generell gilt, dass eine höhere Taktzahl zu mehr Leistung führt. Bei µC liegt diese Taktrate üblicherweise im MHz Bereich.[26]
- Oszillator – er generiert den Systemtakt für den Mikrocontroller. Aus ihm werden alle anderen Takte, die intern benötigt werden, abgeleitet. Die Schwingfrequenz des Oszillators ist meist um einiges langsamer als die eigentliche Taktrate der CPU, da der Grundtakt durch Multiplikatoren vervielfacht wird. Der Oszillator besteht üblicherweise aus einem sehr genauen Schwingquarz und wird nicht in der MCU verbaut, sondern extern so nah wie möglich an der MCU platziert.

[25] Vgl. Metz (2014), Online-Quelle [27.08.2017].

[26] Vgl. ebd.

- Interrupt-Steuerung – sie ist ein wesentlicher Bestandteil eines Mikrocontrollers und erlaubt Programmunterbrechungen, ausgelöst durch externe Steuersignale – Interrupts. Bei einem Interrupt werden alle wesentliche aktuellen Systemwerte zwischengespeichert und die definierte Interruptroutine ausgeführt. Nach Beendigung der Interruptroutine kann der µC durch die Zwischenspeicherung wieder im vorherigen Programmschritt einsteigen. Interrupts bieten die Möglichkeit, eine ständige Zustandsabfrage (Polling) zu umgehen und so erst Rechenleistung aufzuwenden, wenn es nötig ist.[27]

- ROM – steht für Read Only Memory und ist eine veraltete Bezeichnung für den Festspeicher eines Mikrocontrollers, da moderne ROM-Speicher auch mehrfach im Betrieb beschrieben werden können, auf dem sich seine Programmdaten befinden. Der Festwertspeicher definiert sich dadurch, dass bei Ausfall der Versorgungspannung seine gespeicherte Information erhalten bleibt, somit wird er als Permanentspeicher in einem µC genutzt, ROM hat verglichen mit dem RAM eine langsame Lese- und Schreibgeschwindigkeit. Üblicherweise besteht ein ROM heutzutage aus einem Flash oder EEPROM und hat auf dem µC eine Speichergröße im kB- bis MB-Bereich.[28]

- RAM – er bildet den flüchtigen Programmspeicher des µC und steht für Random-Access Memory. RAM kann als Flip-Flop-Transistorvariante oder mittels Kondensatoren realisiert werden. Flip-Flop-Speicher brauchen permanent Spannung, um ihre Information zu halten, die Kondensatorausführung muss nur zyklisch neu beschrieben werden. Im RAM befinden sich Daten, die während des Betriebs des Mikrocontrollers anfallen, wie Sensorwerte oder berechnete Größen. Bei einem Spannungsverlust gehen all diese Daten verloren. RAM hat den Vorteil, dass es im Vergleich zu ROM sehr schnelle Lese- und Schreibraten hat, zudem kann man ihn unbegrenzt oft beschreiben. Verglichen mit ROM ist die Speichergröße von RAM, auf Mikrocontrollern, meist sehr klein und im kB-Bereich.[29]

- Zeitgeber & Zähler – sind wichtige Komponenten, damit der µC Ereignisse zählen und seine Anwendungen in einen Zeitbezug setzten kann. Sie können als externe Komponenten eingebunden werden, finden sich aber sehr oft selbst auf dem Mikrocontroller. Zeitgeber und Zähler sind am µC zwei fest miteinander verknüpfte Komponenten, da die Zeit sehr oft über das Zählen von Systemtakten abgeleitet wird. Typische Anwendungen sind Interrupts, die nach Erreichen eines gewissen Zeitwerts ausgeführt werden, um den µC in einen anderen Zustand zu versetzen oder eine Programmroutine auszulösen. Zeitgeber und Zähler eignen sich nicht für die Generierung von absoluten Zeitstempeln, da nach Spannungsverlust der aktuelle Zeitwert verloren geht.[30]

[27] Vgl. Metz (2014), Online-Quelle [27.08.2017].

[28] Vgl. ebd.

[29] Vgl. ebd.

[30] Vgl. ebd.

- Systembus – in einem modernen ARM-Mikrocontroller wird der Bus als AMBA - Advanced Microcontroller Bus Architecture ausgeführt. Dieser beinhaltet einen APB – Advanced Peripheral Bus und einen AHB – Advanced High Performance Bus. Der AHB ist ein Hochleistungsbus für Anbindungen, die eine schnelle Verbindung zur MCU benötigen. Im Gegensatz dazu steht der APB, ein leistungsschonender, kostengünstiger Bus mit geringem Datendurchsatz, der für langsame Anbindungen wie UART in einem ARM Cortex Verwendung findet. Der AHB ist dabei direkt an die MCU angebunden. Ein Zugriff auf den APB durch Cortex M3/4 erfolgt über eine AHB-APB Bridge die dabei als AHB-Slave agiert. Um Daten ohne großen Overhead und ohne Bearbeitung von dem APB-Subsystem z.B. in den RAM zu schreiben, wird optional ein Direct-Memory-Access-Controller verwendet, der direkt am AHB und APB hängt.[31]
- PMU – ist die Power Management Unit, die in einem Mikrocontroller zum Einsatz kommen kann. Dabei hat sie ihre eigene CPU, Software und Speicher unabhängig vom µC verbaut. Die PMU ist spannungsversorgungsunabhängig und wird durchgehend von einer Reservebatterie (z.B. Knopfbatterie) gespeist. Zu ihren Aufgaben gehören die Steuerung von Ein/Aus-, Sleep- und Deep-Sleep-Modi, die Abschaltung von ungenützter Peripherie, die Strom-Spannungsüberwachung der Versorgung und die Zurverfügungstellung einer Real-Time-Clock für den Mikrocontroller.[32]
- Schnittstellen – sie ermöglichen es dem µC, mit externen Komponenten zu kommunizieren, Datenwerte zu erfassen oder auszugeben. Grundlegend unterscheidet man zwischen digitalen Schnittstellen und analogen Schnittstellen am µC. Digitale Schnittstellen können für die Erfassung oder das Schreiben von einfachen Bits als I/O-Interface ausgeführt sein, oder als GPIO-General Purpose Input Output mehrere Funktionen, wie einen Kommunikationsbus oder Interrupteingang vereinen. Analoge Schnittstellen verfügen je nach Konfiguration über einen A/D- oder D/A-Wandler, da die CPU selbst nur digitale Größen verarbeiten kann.[33]

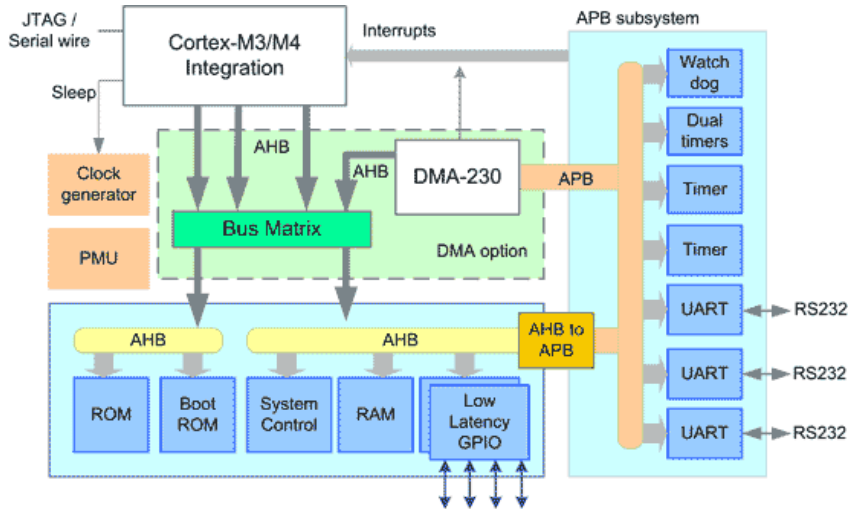


Abbildung 14: Aufbau eines ARM Cortex M3/4; Quelle: https://www.arm.com/assets/images/M3_4_system.png [Datum 27.08.2017]

[31] Vgl. Dönicke (2009), Online-Quelle [13.09.2017].

[32] Vgl. Christensson (2007), Online-Quelle [13.09.2017].

[33] Vgl. Metz (2014), Online-Quelle [27.08.2017].

### 2.2.2 Allzweckeingabe / -Ausgabe

Wie in Kapitel 2.2 bereits kurz beschrieben, werden die Schnittstellen am Mikrocontroller dazu verwendet, um mit externen Komponenten und Schaltungen Daten auszutauschen. Ein wesentlicher Bestandteil dieser Schnittstellen sind die digitalen Allzweckeingabe- und Ausgabe-Schaltkreise auf einem Mikrocontroller, welche auch als GPIO - General Purpose Input/Output bezeichnet werden. GPIOs können auf fast allen Mikrocontrollern vorgefunden werden und sind als einfache Kontakte (Pins) ausgeführt, siehe Abbildung 15. Diese Pins kann man als Eingang oder Ausgang im Programm des Mikrocontrollers definieren, was ihnen ermöglicht, entweder das Anliegen einer Spannung zu erkennen oder ein Steuersignal auszugeben. Bei der Definition des GPIO im Programmcode ist es wichtig, auch einen hochohmigem Pull-up oder Pull-down-Widerstand einzustellen, welcher dann durch die Driver-Bausteine vorgeschaltet wird. Diese Widerstände verhindern einen ungültigen Spannungszustand am Pin und definieren dessen Logik im unbeschalteten Zustand. Mit der Beschaltung als Pull-up wird im Falle keiner Beschaltung am Pin das Potential auf $V_{DD}$ gezogen, bei einer Pull-down Beschaltung ist dies umgekehrt und der Wert geht zu $V_{SS}$, falls er nicht vom Pin-Signal überschrieben wird. Zusätzlich zu einem Pull-up oder Pull-down-Widerstand verfügt der GPIO meist noch über Schutzbausteine wie Dioden oder ggf. selbstheilende Sicherungen, die gegen Kurzschluss und Überlast schützen. Bei der Beschaltung des Pins als Ausgang wird über das „Output Data Register“ der Wert für den Ausgang vom Programm vorgegeben. Je nach Registerwert wird mittels der MOSFETs das Ausgangssignal auf $V_{DD}$ oder $V_{SS}$ gelegt. Zu beachten ist hierbei, dass pro Register zumindest 8 Bits/Pins angesteuert werden. Bei der Beschaltung des Pins als Eingang kann der Wert vom Pin im „Input Data Register“ gelesen werden. Die MOSFETS des Ausgangsdrivers sind hierbei gesperrt. Der TTL Schmitt Trigger im Eingangsdriver dient der Signalaufbereitung und garantiert ein logisch gültiges Signal, welches in den „Input Data Register“ geschrieben wird. Wichtig beim Verwenden von GPIOs ist, den Bereich des Spannungspegels für eine logische 0 und eine logische 1 im Datenblatt nachzulesen. Üblicherweise wird eine logische 0 zwischen 0 V bis 0.8 V erkannt und eine logische 1 zwischen 2 V und 5 V. Der Bereich zwischen 0.8 V und 2 V ist ungültig und nicht erlaubt, er würde in einem nicht definierten logischen Signal resultieren.[34]

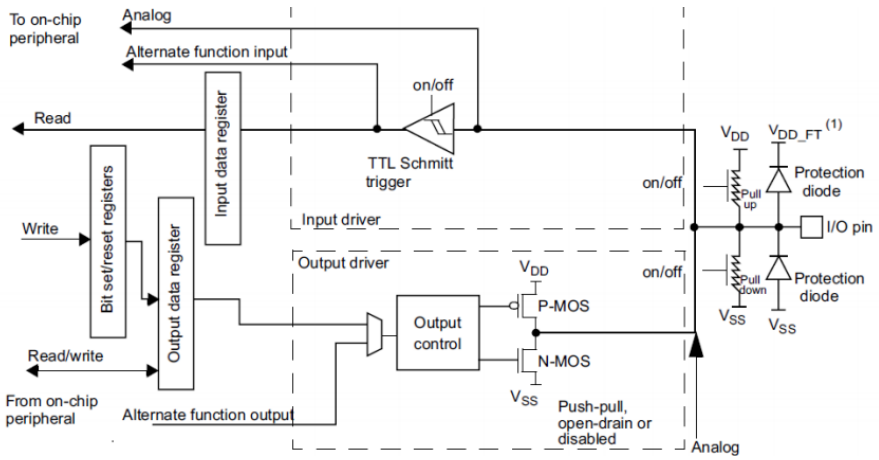


Abbildung 15: STM32F40x GPIO Aufbau; Quelle: https://www.fmf.uni-lj.si/~ponikvar/STM32F407%20project/Ch4%20-%20The%20use%20of%20ports.pdf, Seite 1 [Datum 01.09.2017]

[34] Vgl. Ponikvar (2014), Online-Quelle [1.09.2017].

Neben der Funktion als einfaches I/O haben viele GPIO-Pins eine zusätzliche Funktion. So können bestimmte Pins als Interrupt-Eingang verwendet oder mehrere Pins zusammen zu einer Kommunikationsschnittstelle vereint werden. Eine bekannte und häufig ausgeführte Schnittstelle ist I2C - Inter-Integrated Circuit für welche 2 GPIO Pins, SCL - Serial Clock und SDA - Serial Data benötigt werden. I2C ist ein „Open-Drain" Multimasterbus welcher es dem Mikrocontroller (Master) ermöglicht, mehrere Teilnehmer (Slaves) anzusteuern. Es können jedoch auch mehrere Master im Bussystem vorhanden sein. „Open-Drain" bezieht sich auf die Schaltung der Spannung am Bus, Teilnehmer können das Spannungspotential nur im Takt der „Serial Clock" gegen Masse ($V_{SS}$) ziehen oder bei Sperren des FETs das Potential freigeben, sodass es automatisch mittels $R_{pu}$ gegen Plus ($V_{DD}$) gezogen wird. I2C erlaubt auch eine bidirektionale Übertragung von Daten zwischen Master und Slaves. Dies wird dadurch gesteuert, dass Slaves am Bus nur kommunizieren dürfen, wenn sie vom Master dazu aufgefordert werden. Bei einem Multimastersystem erkennt der Master durch Überwachen der Leitung, wenn sein Befehl von einem anderen Master überschrieben worden ist. Wichtig für die Kommunikation mittels I2C ist auch, dass jeder Teilnehmer eine einzigartige Adresse im Bus fest zugewiesen hat; vordefinierte Adressen der Teilnehmer können den Datenblättern der Bausteine entnommen werden. Die Datenübertragung am Bus selbst erfolgt durch ein Startsignal und eine Lese- Schreibadresse, die vom Master vorgegeben wird. Anschließend folgt eine einfache Datenübertragung durch Ziehen der SDA-Leitung im Takt von SCL zu 0 oder 1. Nach Beendigung der Datenübertragung wird vom Datenempfänger eine Bestätigung (ACK) oder ein Abbruch (NACK) nach Erhalt der Daten gesendet. Die Datenübertragung ist abgeschlossen sobald der Master ein Stoppsignal sendet.[35]

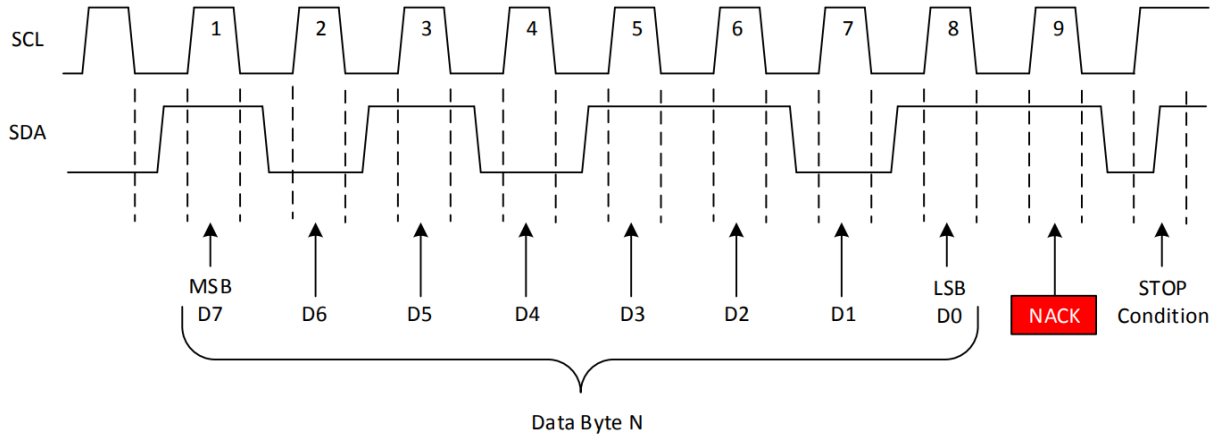


Abbildung 16: I2C Datenübertragung mit Abbruch; Quelle: http://www.ti.com/lit/an/slva704/slva704.pdf, Seite 6 [Datum 02.09.2017]

Eine weitere wichtige und häufige Schnittstelle, die mittels zweier GPIO Pins realisiert werden kann, ist das UART-Universal Asynchronous Receiver Transmitter Interface welches es ermöglicht, eine einfache serielle Kommunikation mit externen Bausteinen aufzubauen. UART besteht aus einer Empfangs- (RxD) und eine Sendeleitung (TxD), welche gleichzeitig Daten ohne Taktsignal übertragen können. Damit dies funktioniert, müssen beide Teilnehmer den gleichen Massenbezug aufweisen und die Datenrate sowie die Kommunikationsregeln bei beiden Teilnehmern bekannt sein. Die Datenrate wird bei UART in Baud, Symbole/Character pro Sekunde, angegeben. Übliche

[35] Vgl. Texas Instruments (2015), Online-Quelle [02.09.2018].

Geschwindigkeiten befinden sich in der Größenordnung zwischen 2400 und 115.200 Baud. Neben der Übertragungsgeschwindigkeit müssen noch weitere Parameter für eine erfolgreiche Kommunikation zwischen den Teilnehmern definiert werden. Dazu gehört die Anzahl an Datenbits, die übertragen werden. Um den gesamten ASCII-Code zu übertragen, reichen 7 Bits, für den erweiterten ASCII Bereich werden 8 Bits benötigt. Die Parität der Übertragung kann deaktiviert oder zwischen gerade und ungerade eingestellt werden. Sie ermöglicht eine sehr einfache Erkennung von Übertragungsfehlern, wobei die Anzahl an übertragenen Einsen gezählt und je nach Auswahl der Konfiguration das Paritybit auf null oder eins gesetzt wird, um auf eine gerade oder ungerade Anzahl an positiven Flanken zu kommen. Das Stopbit in der Übertragung bestimmt, wie viele Bits zwischen zwei Übertragungen verstreichen müssen, damit eine neue Übertragung beginnen kann. Üblich sind 1 oder 2 Bits. Die letzte UART-Einstellungsmöglichkeit ist die Reihenfolge, mit welcher die Bytes vom Speicher auf die Leitung gelegt werden. Möglich ist, zuerst das LSB (least significant bit) oder MSB (most significant bit) auszugeben. ASCII Zeichen werden üblicherweise mit LSB zuerst übertragen.[36]

Abbildung 17 zeigt die Übertragung des ASCII-Zeichens „W“, welches 0x57 in hexadezimal oder binär „1010111“ entspricht. Es ist ersichtlich, dass nach dem Startbit das niedrigste Datenbit von „W“ als Erstes übertragen wird. Nach den 8 Datenbits wird eine negative Parität aufgrund von fünf (ungerade) high und der dementsprechenden Einstellung auf „ungerade“ ausgeben. Der Abstand zur nächsten Übertragung beträgt 2 Stoppbits.

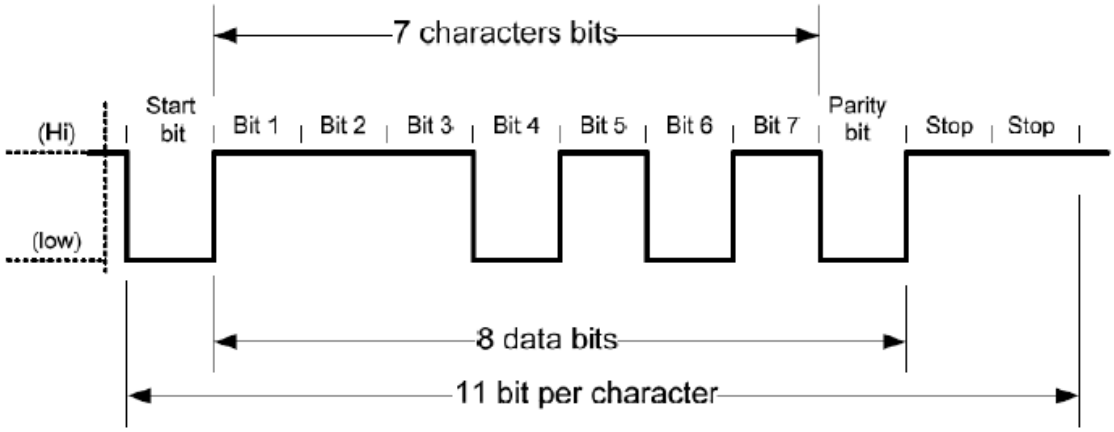


Abbildung 17: UART Übertragung (Modifiziert); Quelle: https://learningmsp430.files.wordpress.com/2014/01/w_example.png [Datum 02.09.2017]

[36] Vgl. Simply Embedded (2017), Online-Quelle [2017.09.02].

# 3 SOFTWARE

## 3.1 Programmiersprache C

Die Programmiersprache C ist eine „GPL-General Purpose Programming Language“ und wurde 1969 bis 1973 in den Bell Laboratories von Dennis Ritchie erfunden. Aufgrund ihrer frühen Standardisierung in der ISO/IEC 9899-1900 gilt sie als eine der wichtigsten und grundlegendsten Programmiersprachen weltweit. Der Bedarf an dieser universellen Programmiersprache ergab sich durch die Entwicklung des Unix-Betriebssystems, welches anfänglich noch in Assembler, B und BCPL programmiert wurde. C entwickelte sich aus der Programmiersprache B, welche wiederum eine Ableitung der Programmiersprache „Basic Combined Programming Language-BCPL“ ist. GPLs sind Programmiersprachen, die für nichtnumerische Probleme entwickelt wurden und eine Abstrahierung der Maschinensprache, z.B. Assembler, auf einem höheren Level ermöglichen. Der Unterschied zwischen B, BCPL, und C sind nicht nur die Befehle, sondern auch die Einführung von Datentypen wie INT oder CHAR und die nötige Konvertierung von C in Assembler durch einen Compiler.[37]

Um einen lauffähigen Computercode in C zu erhalten, muss der Anwender zuerst C-Quellcode erstellen. Der Quellcode beinhaltet die abstrahierten Programm- und Rechenabläufe, die ein Prozessor auszuführen hat. Dieser Quellcode kann wiederum in drei Teilbereiche, die Präprozessordirektive, das Hauptprogramm und die Dokumentation gegliedert werden. In der Präprozessordirektive werden Anweisungen hinzugefügt, die beim Kompilieren zu beachten sind, so kann auf vorhandenem Programm-Bibliotheken (Header oder Dynamic Link Libraries) via „#include“ verwiesen werden, es können aber auch Makros oder Symbole via „#define“ vordefiniert werden. C beinhaltet eine Vielzahl an möglichen Präprozessordirektiven. Diese Befehle können an ihrer Positionierung am Anfang des Quellcodes und durch das vorangehende Rautezeichen „#“ erkannt werden. Das Hauptprogramm beinhaltet das Schlüsselwort „main()“ und befindet sich typischerweise in der Datei „main.c“. Nach einem Programmstart und der Initialisierung des Codes wird immer der Inhalt vom main()-Verzeichnis als Erstes ausgeführt. Aus diesem Grund wird es auch als Einstiegspunkt im Code bezeichnet. Das Main-Verzeichnis beinhaltet die auszuführenden Programm-Befehle und Variablendeklarationen, es ist auch der Aufruf von anderen Funktionen, und somit eine weitere Programmverzweigung möglich. Auch die Dokumentation des Quellcodes ist ein wesentlicher Bestandsteil des Programmcodes. Dies dient der Nachvollziehbarkeit des Erstellten, hat aber keinen direkten Einfluss auf die Funktion des Codes. Die Dokumentation im Quellcode muss via den Zeichen „//“ (C++) für eine Zeile oder „*/“ „/*“ für Bereiche erwirkt werden, da der Compiler ansonsten nicht zwischen Dokumentation und Befehl unterscheiden kann.[38,39]

Um den in einer Hochsprache geschriebenen Quellcode dem Computer lesbar zu machen, wird der Quellcode mithilfe eines Compilers in eine Maschinensprache übersetzt. Die Maschinensprache ist eine sehr einfache, reduzierte und auf Performance ausgelegte, für den Computer verständliche

[37] Vgl. Erlenkötter (1990).

[38] Vgl. ebd.

[39] Vgl. Schröder (2016), Online-Quelle [25.08.2017].

Programmiersprache und verfügt nur über grundlegende Operanden. Durch das Herunterbrechen der Hochsprache auf einfache Rechenoperationen ergibt sich auch, dass der Compiler eine Hochsprache in verschiedene Maschinencodes übersetzten kann und somit der Code auf verschiedenen Architekturen ausgeführt werden kann. Zusätzlich muss er auch den übersetzten Quellcode auf Fehler überprüfen und gegebenenfalls den Programmstart unterbrechen oder Korrekturen vornehmen. Ein weiterer wichtiger Punkt ist, dass der Compiler auch für die Performance-Optimierung des Programmcodes, im Besonderen bei Schleifen, zuständig ist. Diese Optimierung ist ein wesentliches Bewertungskriterium für einen guten Compiler. Nach der Übersetzung des Quellcodes in die Maschinensprache kommt noch der Linker zur Anwendung. Der Linker hat die Aufgabe, den übersetzten Code mithilfe von Referenzbibliotheken in die richtige Reihenfolge zu bringen und einen für die Maschine direkt ausführbaren Code zu erstellen. Üblicherweise werden Linker, Compiler und Textbearbeitungsprogramme-IDEs für den Quellcode als ein Softwarepaket angeboten, was dem Anwender die Programmierung und Arbeit mit einem Tool ermöglicht. [40,41]

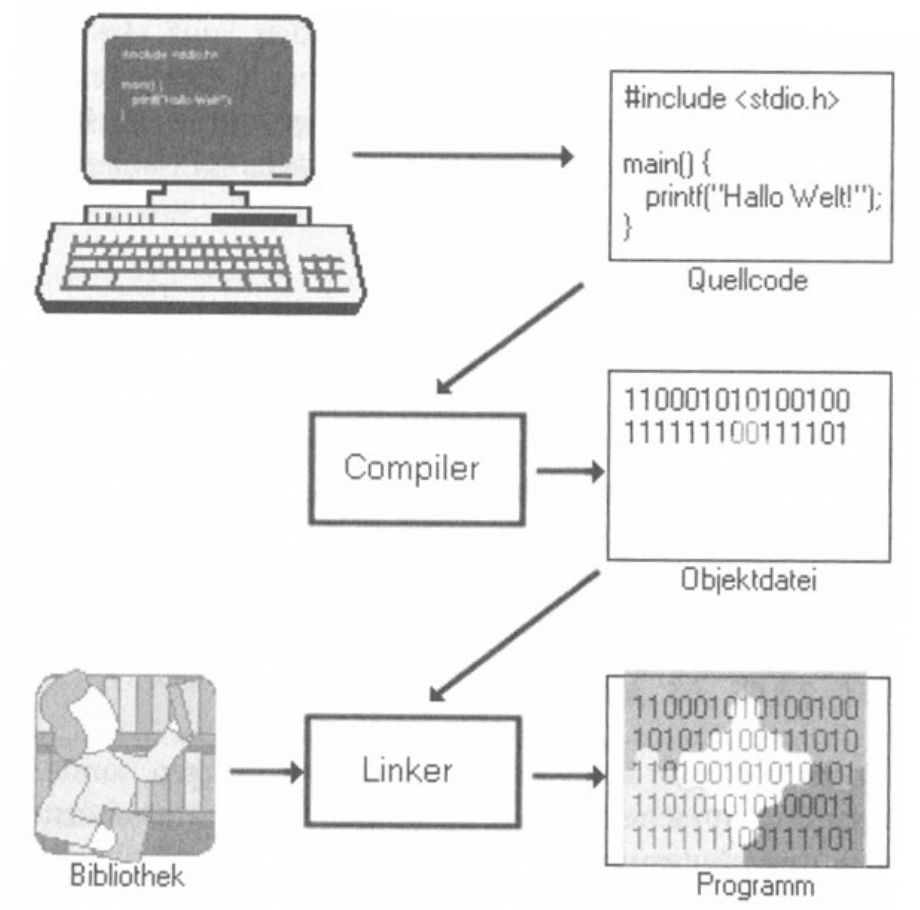


Abbildung 18: Vom Quellcode zum Programm; Quelle: Erlenkötter (1990) S.13

[40] Vgl. Erlenkötter (1990).

[41] Vgl. Mogensen (2000 – 2010), Online-Quelle [25.087.2017].

## 3.2 Low-Power-Programmieransätze

Um eine Low-Power-Applikation mit einem Mikrocontroller zu realisieren, ist es nicht ausreichend, nur eine energiesparende Hardware zu verwenden, sondern sämtliche Funktionen, Ausgänge und Operationen der MCU müssen dementsprechend entworfen und mittels Software energiesparend angesteuert werden. Wesentlich für eine energiesparende Operation ist, dass externe Schaltungen oder Bereiche eines Mikrocontrollers, die nicht benötigt werden, ausgeschaltet oder deaktiviert sind. Vor allem LEDs die den MCU Status ausgeben und fest am Board integriert sind, sollten hier beachtet werden, da sie den Energieverbrauch wesentlich erhöhen. Es ist auch möglich und empfehlenswert, Schaltungen, die kurzfristig nicht benötigt werden wie EEPROMS oder Sensoren, mittels Software und z.B. einem MOSFET für kurze Zeit zu deaktivieren. Auch I/O-Schnittstellen sollten dabei so konfiguriert werden, dass sie möglichst wenig Energie verbrauchen. Im Zweifelsfall wäre eine Konfiguration als unbelegter Ausgang oder geerdeter Eingang zu verwenden. Eine Eruierung des Energieverbrauchs in den unterschiedlichen Betriebsmodi der MCU, z.B. Sleep, Initialize, Run bietet des Weiteren die Möglichkeit abzuschätzen, wie oft und wie lange die MCU in Sleep zu setzten ist, um den vorgegebenen Leistungsverbrauch nicht zu überschreiten. Zusammen mit den möglichen Energiequellen, die die Schaltung versorgen, und deren Kapazitäten erhält man ein Leistungsbudget des Mikrocontrollers. Der Energieverbrauch in den Betriebszuständen der MCU ist auch von dessen Taktrate abhängig, generell gilt, dass ein höherer Takt zu einem größeren Energieverbrauch führt. Aus diesem Grund sollte man die langsamst mögliche Taktrate für seine Applikation wählen, oder den Clockspeed dynamisch im Code anpassen. Im RUN-Betrieb sind Programmschleifen zu vermeiden und Interrupts dem Polling zu bevorzugen, falls die MCU nicht ausgelastet ist oder auf externe Komponenten warten muss, sollte sie zumindest in den Sleep Modus versetzt werden. Falls es die Applikation und die MCU zulässt, kann man das System auch in einen Deep-Sleep-Modus setzten. Dies ist der Zustand mit dem niedrigsten Energieverbrauch, hierbei werden alle wesentlichen Energieverbraucher ausgeschaltet, jedoch kann es zu Informationsverlusten kommen, da zum Beispiel der SRAM geleert wird. Des Weiteren wird bei Reaktivierung, Zeit und Energie für den Mikrocontrollerstart verschwendet, somit rentiert sich ein Deep-Sleep-Modus erst nach einer gewissen Ruhezeit. Für Rechenoperationen gilt, dass Bitverschiebungen und Masken wesentlich schneller und effizienter von der MCU ausgeführt werden als komplexe Rechenvorgänge, eine Programmierung mit solchen Bitoperationen bedarf zwar einer größeren Anstrengung, resultiert aber in einem niedrigerem Energieberdarf.[42]

[42] Vgl. Microchip Technology Inc. (2009), Online-Quelle [26.08.2017].

## 3.3 Datenerfassung über das Camera-Parallel-Interface

Um die Daten, die mittels dem CPI auf den Mikrocontroller geschrieben werden, richtig zu interpretieren, und zu einem Bild zusammenzufügen, muss der Mikrocontroller die jeweilige Bildposition aller gelesenen Pixel wissen. Um diese Information bereitzustellen, wird neben der Datenleitung eine Vertikal-SYNC (VSYNC), eine Horizontal-Reference (HREF) und eine Pixel-Clock (PCLK) als Taktgeber vom Kameramodul zum Mikrocontroller übertragen. Dabei zu beachten ist, dass das CPI-Interface und die Taktleitungen unidirektional sind und so eine Kommunikation mit dem Kameramodul nur über andere Steuerleitungen wie I2C möglich ist. Wie in Abbildung 19 beschrieben, werden Bilddaten bei jeder steigenden Flanke der PCLK übertragen, je nach Breite des CPI und übertragendem Bildformat benötigt man für einen vollständigen Pixel ein oder zwei PCLK-Takte. Der HREF-Takt oder Interrupt wird bei der Übertragung so lange auf high gesetzt, bis eine ganze Zeile vollständig übertragen wurde. Zwischen zwei Zeilen geht HREF auf Low. Bei Übertragung eines neuen Frames wird VSYNC kurz auf high gesetzt, um dem Mikrocontroller ein neues Bild anzukündigen. Neben dem horizontalen Timing gibt es auch ein vertikales Verfahren, wo das Signal HSYNC übertragen wird. Das HSYNC Signal dient der horizontalen Synchronisation und wird bei einer neuen Spalte getriggert. Ein VREF-Signal ist bei Kameramodulen eher unüblich, würde aber ein High ausgeben, solange eine Bildlinie geschrieben wird. Die Anzahl der Pixel in einem Sensor und auch die Verteilung in horizontale und vertikale Pixel ist je nach Sensortype unterschiedlich. Eine Implementation mit Hilfe der horizontalen und vertikalen Takte ermöglicht aber eine flexible Einbindung verschiedener Kamerasensoren.[43]

Ein weiterer wichtiger Punkt der Datenerfassung ist die Framerate oder Bildrate der zu übertragenden Daten. Sie findet besonders bei Videosignalen eine Relevanz. Die Bildrate ergibt sich aus der Zeitdauer, die benötigt wird, um die gesamte Information aller Bildpunkte zu übertragen, zusätzlich werden pro Bild noch vorangehende und nachfolgende Daten außerhalb des Bilderreiches übertragen. Daten außerhalb des Bildbereiches werden auch als “front porch” und “back porch” bezeichnet. Sie bestehen aus inaktiven Pixeln und bilden einen Overhead zum eigentlichen Bildmaterial, sie ergeben sich dadurch, dass die Fläche des Bildsensors nicht ganz ausgenutzt wird. Der Sensor aus Abbildung 19 hat eine Auflösung von 656x492 (0.3 MP) Pixel, davon sind aber nur 640x480 aktiv. Bei der Datenübertragung werden aber alle Pixel und somit 322.752 Pixel übertragen. In RGB555 oder RGB565 ergibt dies mit einer 8Bit CPI 2 benötigte Byte pro Pixel und somit in Summe 645.504 Byte an Daten pro Frame. Da pro PCLK Takt nur ein Byte übertragen wird benötigt es 645.504 PCLK-Takte pro Frame. Bei einer theoretischen Taktrate von 25 Mhz ergibt dies 38.7 frames per second – fps. Die reale Taktrate liegt aber meist ein bisschen unter der theoretisch berechneten, da die Verarbeitungszeit des Mikrocontrollers berücksichtigt werden muss.[44]

$$Bildfrequenz = \frac{1}{\frac{Bilddaten}{Taktrate}} \qquad (3\text{-}1)$$

Bildfrequenz/fps = Updaterate des Bildes
Bilddaten/Byte = Übertragende Bildinformation pro Frame
Taktrate/Hz = Taktfrequenz von PCLK

[43] Vgl. FTDI (2015), Online-Quelle [20.08.2017]

[44] Vgl. ebd.

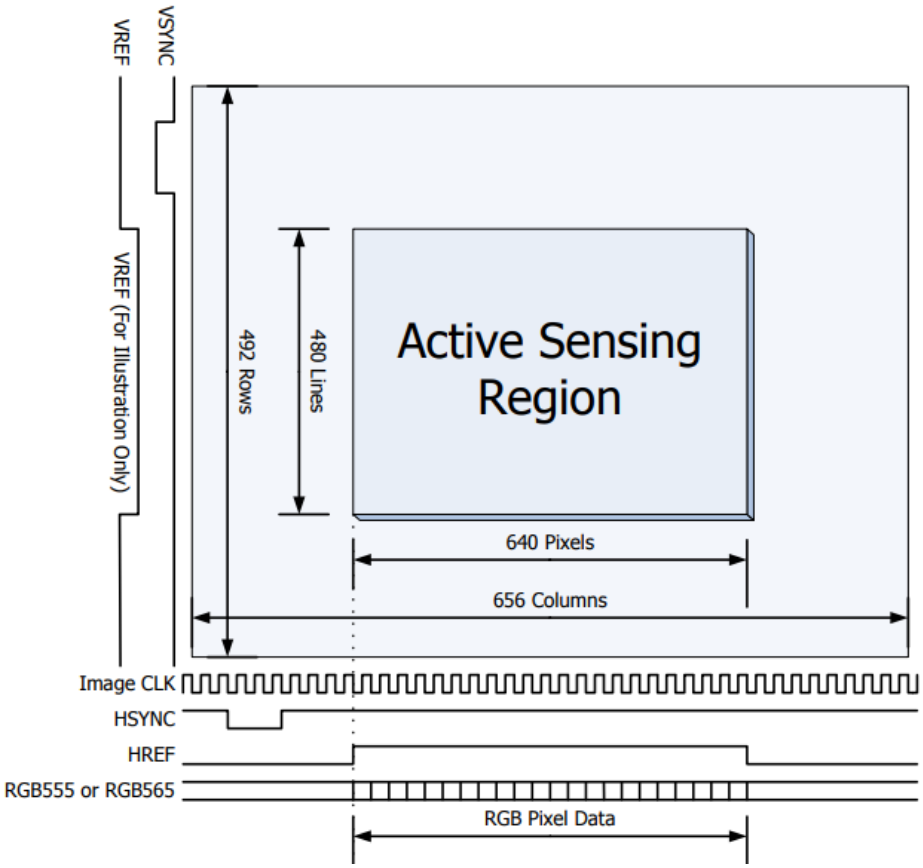


Abbildung 19: Taktraten des Bildsignals; Quelle: FTDI (2015), Online-Quelle [20.08.2017]

# 4 AUSWAHL DER KOMPONENTEN

## 4.1 Mikrocontroller

Die Auswahl des richtigen Mikrocontrollers für den praktischen Teil der Arbeit erfolgt unter den geforderten Gesichtspunkten, eine energiesparende und kostengünstige Kameraapplikation zu erstellen und die generierten Daten auszuwerten. Vielversprechende Kandidaten für diese Anwendung sind hierbei die Arduino-, STM32-, LPC- und die Atmel-Produktreihe, die allesamt preiswerte Ultra-Low-Power Mikrocontroller in ihrem Portfolio anbieten. Arduino bezieht hierbei seine MCUs von anderen Herstellern wie Atmel und Intel und stellt diese als Mikrocontroller mit einer Arduino-IDE zur Verfügung. Das Ziel von Arduino ist es, hierbei die Programmierung und Ansteuerung so einfach wie möglich für den Anwender zu gestalten, was zu einer weiten Verbreitung und einer umfangreichen Wissensdatenbank im Internet geführt hat. In die engere Auswahl im Portfolio von Arduino ist der „Nano“ gekommen, der nur für wenige Euros erwerbbar und mit einer Atmega 328 MCU ausgestattet ist. Aufgrund der geringen Rechenleistung und RAM-Größe ist eine Bildverarbeitung/Bearbeitung aber nur bei geringen Auflösungen der Kameramodule möglich. Die beiden Hersteller NXP (LPC) und STMicroelectronics (STM32) bieten ein sehr ähnliches, günstiges Portfolio im Bereich der Cortex MCUs an, jedoch wurde aufgrund ihrer Low-Power-Eigenschaft und des Direct Memory Access controllers (DMA) die MCU STM32L496AGI6 für diese Arbeit gewählt. Zusätzlich gibt es die Möglichkeit, den STM32L496AGI6 als Entwicklerboard (32L496GDISCOVERY) mit einem LCD für Tests des Kamerabildes und einem Camera Module Interfaces (DCMI) zu beziehen. Der STM32L4x ist eine Cortex M4-MCU die über ausreichend Rechenleistung verfügt, um hochauflösende Bilder zu verarbeiten und bei Bedarf auszuwerten.

Neben den bekannten Plattformen wird auch die OpenMV Cam auf ihr Potential für diese Arbeit untersucht. Das erfolgreiche Kickstarterprojekt bietet auf seiner Website ein Mikrocontrollerboard mit einer STM32F7x MCU und einem OV7725 Kameramodul an, wobei der Sensor bzw. die Linse gegen andere Modelle ausgetauscht werden können. Die Stärken der OpenMV Cam liegen in der Echtzeitauswertung von Bildinformationen, welche mithilfe einer OpenMV Python-IDE sehr einfach programmiert und auf dem Mikrocontroller implementiert werden kann. Das OpenMV-Modul wird aufgrund des hohen Energieverbrauches der Cortex-M7 STM32F7x MCU und des eingeschränkten Zugriffs der IDE auf die MCU für diese Arbeit als ungeeignet betrachtet.[45]

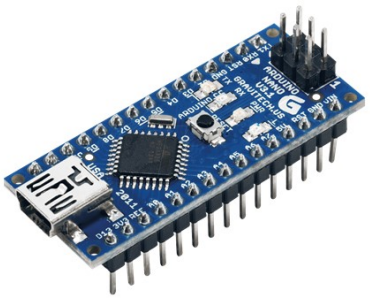

Abbildung 20: Arduino Nano; Quelle: http://www.etechpk.net/wp-content/uploads/2016/02/ARDUINO_NANO_03.png [Datum 11.09.2017]

Abbildung 21: OpenMV Kameramodul; Quelle: https://cdn.sparkfun.com//assets/parts/1/2/1/1/7/14186-01a.jpg [Datum 11.09.2017]

[45] Vgl. Open MV (2017), Online-Quelle [11.09.2017].

## 4.2 Kameraschnittstellen

In der Industrie gibt es mehrere mögliche Schnittstellen, wie ein Kameramodul seine Bildinformationen weitergeben kann. Grundlegend unterscheidet man zwischen digitalen und analogen Video-Schnittstellen. Eine gebräuchliche, ältere analoge Schnittstelle ist das „Composite Video“, welches besonders in Fernsehern und Videokameras seine Anwendung gefunden hat. Kameramodule die über einen „Composite Video“ Ausgang verfügen, bieten meist nicht die Möglichkeit, dass sie z.B. via I2C angesteuert werden können. Zudem befindet sich die Auflösung des Videosignals üblicherweise nur im Bereich von PAL (768x576) oder VGA (640x480). Bei der Auswertung und Verarbeitung des analogen Videosignals mit einem Mikrocontroller ergibt sich das Problem, dass das Videosignal erst mittels eines sehr schnellen A/D-Wandlers in ein lesbares Digitalsignal umgewandelt werden muss. Dieses Verfahren ist leistungs- und ressourcenbeanspruchend, weswegen in der Praxis digitale Signale für eine µC-Verarbeitung bevorzugt werden. Bei digitalen Videosignalen von einem Kameramodul unterscheidet man zwischen einem seriellen (MIPI) oder parallelen Dateninterface (CPI), welches vom „Image Sensor Processor“ direkt zur Verfügung gestellt wird. Kameramodule, welche auf dem MIPI-Standard beruhen, findet man in großen Mengen und zu sehr günstigen Preisen am Markt, da man auf die Kameramodule von modernen Smartphones zugreifen kann, welche fast alle das MIPI-Protokoll verwenden. Jedoch ist das Protokoll nur für Mitglieder der MIPI Allianz offengelegt. CPI hingegen wird nur von wenigen Anbietern am Markt wie OmniVision oder ON Semiconductor angeboten, bietet dafür aber eine ausführliche Schnittstellenbeschreibung. Neben MIPI und CPI gibt es auch noch weniger gebräuchliche Anbindungen via UART oder SPI, wobei in allen Fällen der „Image Sensor Processor“ MIPI oder CPI ausgibt und ein zweiter Prozessor am Kameramodul das Videosignal entsprechend umwandelt.

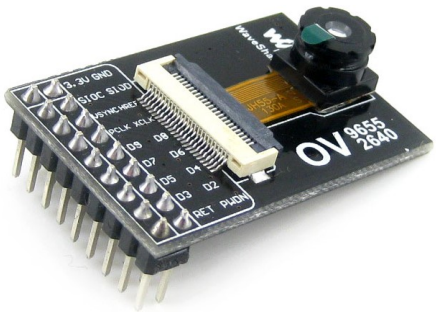


Abbildung 22: CPI Kameramodul; Quelle: http://www.waveshare.com/img/devkit/accBoard/OV9655-Camera-Board/OV9655-Camera-Board-2.jpg [Datum 11.09.2017]

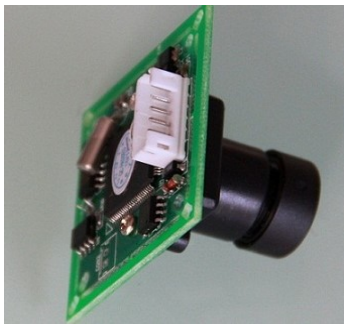

Abbildung 23: UART Kameramodul; Quelle: http://www.si-cube.com/UpFiles/Article/201106/201162155413577.jpg [Datum 11.09.2017]

Abbildung 24: Composite Video Kameramodul; Quelle: https://cdn.sparkfun.com//assets/parts/1/9/9/2/08773-03-L.jpg [Datum 11.09.2017]

Abbildung 25: MIPI Kameramodul; Quelle: https://www.e-consystems.com/images/denebola/Denebola-CX3-RDK-camera-Module.jpg [Datum 11.09.2017]

## 4.3 Kameramodule

Aufgrund der in Kapitel 4.2 beschriebenen Limitierungen der Kameraschnittstellen wurde für die Komponentenauswahl MIPI und das analoge „Composite Video-Signal“ außer Acht gelassen. Die nach dem Ausschlussverfahren verbleibende CPI- und UART- sowie SPI-Anbindung wurden für die Suche von Kameramodulen herangezogen. Dabei wurde ein besonderes Augenmerk auf den Energieverbrauch der Module, deren maximale Framerate/Auflösung und deren Ansteuerungsmöglichkeit (Standby/Sleep) gelegt. Beim Vergleich von Datenblättern der Kamerasensoren mit einer integrierten Signalumwandlung zu UART/SPI erschien hierbei ein erhöhter typischer Energiebedarf bei Dauerbetrieb (~200 mW) im Vergleich zu CPI Modulen mit demselben Sensor (~60 - 120 mW), weswegen für die engere Auswahl schlussendlich vermehrt CPI herangezogen wurde.

Für die Masterarbeit kamen folgende Sensoren, siehe Tabelle 1, aufgrund ihrer Verfügbarkeit und der geforderten Eigenschaften in die engere Auswahl:

Tabelle 1: Sensorauswahl

| Sensor | Auflösung/MP | Energieverbrauch/mW lt. Datenblatt | Schnittstelle | Sensortype | Preis/Euro vom 11.09.2017 |
|---|---|---|---|---|---|
| OV9650 | 1.3 | 50 | CPI | CMOS | 4 |
| OV7670 | 0.3 | 60 | CPI | CMOS | 4 |
| OV5640 | 5 | 140[46] | CPI | CMOS | 18 |
| MT9M001 | 1.3 | 363 | CPI | CMOS | 17 |
| OV7725 | 0.3 | 120 | CPI | CMOS | 8 |
| OV9655 | 1.3 | 90 | CPI | CMOS | 8 |
| OV2640 | 2 | 140 | CPI | CMOS | 22 |

Eine untergeordnete aber trotzdem relevante Eigenschaft der Sensoren ist die Breite des aufgenommenen Lichtspektrums, so ist im Speziellen der Infrarotbereich für die spätere Anwendung interessant. Nicht alle der in Tabelle 1 genannten Kameramodule bieten diesen Bereich aufgrund ihrer Infrarotfilter, die IR-Filter werden dabei auf der Kameralinse oder direkt über den Sensoren angebracht, an. Somit wurde für die endgültige Auswahl auch in Betracht gezogen ob, der Filter bei dem jeweiligen Modul zugänglich ist und falls vorhanden, ohne Probleme entfernt werden kann.

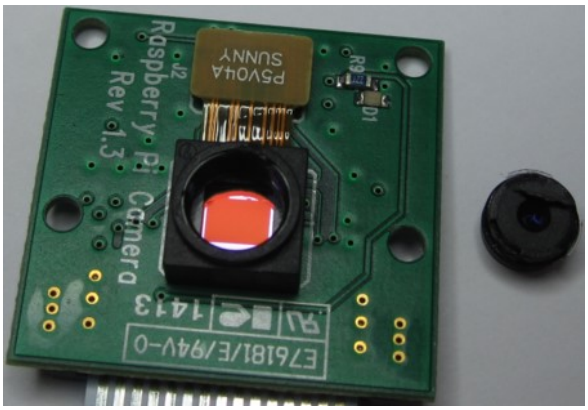


Abbildung 26: IR-Filter über dem CMOS Sensor; Quelle: http://wiki.raspberrytorte.com/index.php?title=File:RemoveLensFromRPiCamModule.png [Datum 12.09.2017]

[46] Hersteller verweist auf 140mA beim Energiebedarf, Annahme 140mW ist gemeint.

# 5 HARDWARE AUSWAHL

## 5.1 STM32L496G-Discovery Board

### 5.1.1 Übersicht

Das STM32L496G-Discovery Board ist eine Evaluierungs-Platine mit einer STM32L496AGI6 Mikrocontrollereinheit. Die Architektur der 32-Bit-MCU entspricht dem eines ARM Cortex M4, mit einem 1 MB großem Flashspeicher und 320 kB RAM. Neben den üblichen GPIOs bietet das Discovery Board einen 1.54 Zoll großen TFT Farb-LCD, der mit 240 x 240 Pixel auflöst und berührungsempfindlich ist. Das Board verfügt auch über einen Audio-Stereoausgang mit SAI Audio CODEC, einem analogen Mikrofon-Eingang und zwei digitale MEMS-Mikrophone. Für die dauerhafte Speicherung von Daten kann auf einen integrierten microSD™-Anschluss inklusive Karte zurückgegriffen werden. Für die Einbindung eines 8 - 14 Bit-CPI-Kameramoduls ist ein Flachbandkabelanschluss vorgesehen, welcher es der Kamera mithilfe eines Direct-Memory-Access (DMA) Steuerbausteins ermöglicht, direkt in den RAM der MCU zu schreiben. Als schnelle Ein- und Ausgabe am Board können zudem ein fest verbauter XY-Joystick und Reset/Eingabeknöpfe verwendet werden.[47]

Ziel des STM32L496G-Discovery Boards ist es, eine vollständige Entwicklungsplattform zu bieten, mit deren Hilfe Anwendungen umgesetzt und demonstriert werden können. Dabei liegt ein klarer Fokus auf Ultra-Low-Power Anwendungen, die jedoch auf ausreichend Rechenleistung zugreifen können, falls diese benötigt wird. Programmcodes und Speicherzustände des µC können mithilfe des ST-LINK/V2-1 Debuggers in Echtzeit analysiert werden. Die STM32-Serie unterstützt hierbei eine Vielzahl an IDEs, die wohl bekannteste darunter ist die µVision ARM-MDK von Keil. Leistungsmessungen des µC können mithilfe der IDD-Schnittstelle am Board einfach durchgeführt werden, des Weiteren kann der Mikrocontroller seinen Leistungsbedarf für die unterschiedlichen Betriebsmodi berechnen und ausgeben.[48]


Abbildung 27: Vorderseite 32L496G-DISCOVERY (Modifiziert); Quelle: http://www.mikrozone.sk/obrazky/newspost_images/12.27.19.jpg [Datum 12.09.2017]

[47] Vgl. STMicroelectronics (2017), Online-Quelle [12.09.2017].

[48] Vgl. ebd.

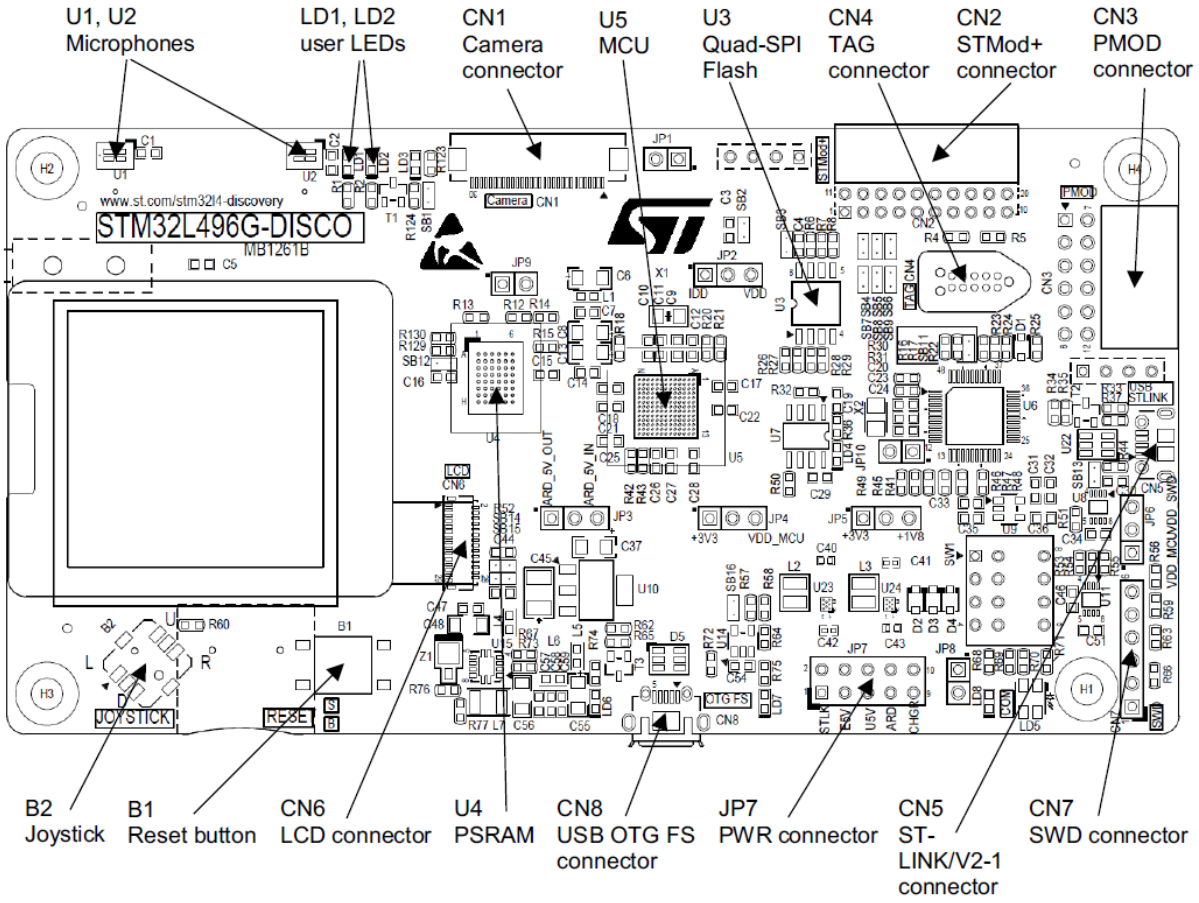


Abbildung 28: 32L496G-DISCOVERY Übersicht Vorderseite; Quelle: UM2160 User manual (2017) S.11

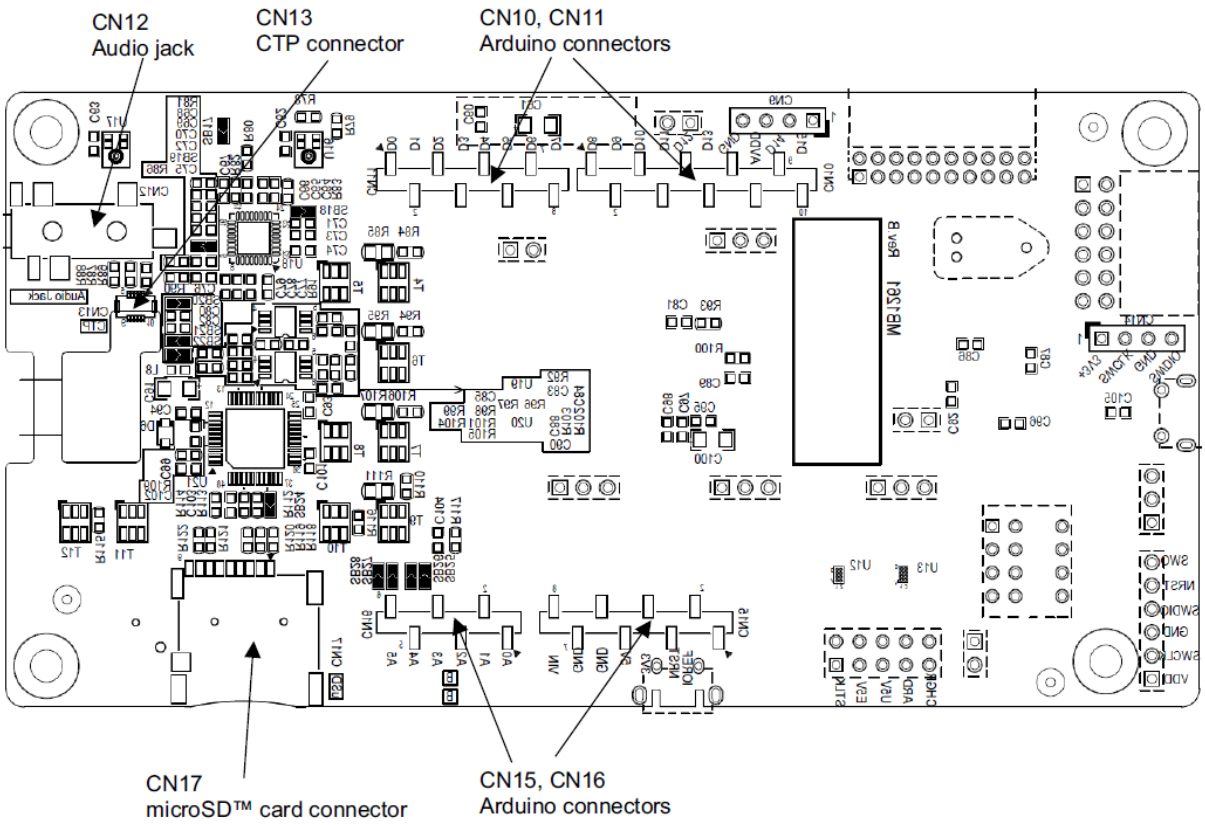


Abbildung 29: 32L496G-DISCOVERY Übersicht Rückseite; Quelle: UM2160 User manual (2017) S.12

### 5.1.2 Digital Camera Interface

Die Kameraschnittstelle DCMI-CN1 am Mikrocontroller bietet die Möglichkeit, ein bis zu 8-Bit breites Camera-Port-Interface einzubinden. Die Schnittstelle ermöglicht zusätzlich zu I2C und üblichen Taktleitungen auch ein Taktsignal für den System-Clock-Eingang an das Kameramodul zu schicken. Somit ist es nicht nötig, am Kameramodul einen externen Oszillator anzuschließen. Für den Betrieb der Kamera muss der µC mindestens mit 3.3 V versorgt werden.[49]

Das Einlesen der Videodaten über DCMI kann hierbei kontinuierlich in den Betriebszuständen Run, Sleep und Low-Power oder getriggert für einen Frame erfolgen. Die Betriebszustände Standby oder Shutdown bewirken eine Deaktivierung der Schnittelle mit Datenverlust. Es gibt auch die Möglichkeit, die Bildinformation direkt am DMCI zu schneiden und somit nicht benötigte Bildinformation zu entfernen, bevor sie in den RAM-Speicher mittels DMA-Baustein geschrieben wird. Wie in Abbildung 30 exemplarisch gezeigt, wird je nach Tiefe des Datenbusses die Bildinformation unterschiedlich im Speicher abgelegt. Bei einem 8-Bit-CPI werden 4 Datenblöcke zu je 8-Bits in eine Adresse zu 32-Bit geschrieben, bei 10- 12 und 14-Bit-CPI Anbindungen werden nur 2 Datenblöcke zu je 10 bis 14 Bit auf die Adresse gelegt. Die Information wird, sobald die 4-Byte der Adresse vollständig beschrieben worden sind, mittels DMA-Controller in den FIFO-Speicher übertragen. Zusätzlich besteht die Möglichkeit, bei HREF- bzw. VSYNC-Fehlern in der Übertragung, einem Überschreiben der Daten vom DCMI oder der Beendigung der Datenübertragung, einen Interrupt für den µC zu generieren.[50]

8-Bit

| Byte address | 31:24 | 23:16 | 15:8 | 7:0 |
|---|---|---|---|---|
| 0 | Dn+3[7:0] | Dn+2[7:0] | Dn+1[7:0] | Dn[7:0] |
| 4 | Dn+7[7:0] | Dn+6[7:0] | Dn+5[7:0] | Dn+4[7:0] |

10-Bit

| Byte address | 31:26 | 25:16 | 15:10 | 9:0 |
|---|---|---|---|---|
| 0 | 0 | Dn+1[9:0] | 0 | Dn[9:0] |
| 4 | 0 | Dn+3[9:0] | 0 | Dn+2[9:0] |

12-Bit

| Byte address | 31:28 | 27:16 | 15:12 | 11:0 |
|---|---|---|---|---|
| 0 | 0 | Dn+1[11:0] | 0 | Dn[11:0] |
| 4 | 0 | Dn+3[11:0] | 0 | Dn+2[11:0] |

14-Bit

| Byte address | 31:30 | 29:16 | 15:14 | 13:0 |
|---|---|---|---|---|
| 0 | 0 | Dn+1[13:0] | 0 | Dn[13:0] |
| 4 | 0 | Dn+3[13:0] | 0 | Dn+2[13:0] |

Abbildung 30: DCMI FIFO Ablage; Quelle: STM32L4DCMI S.5

[49] Vgl. UM2160 User manual (2017) S. 29.

[50] Vgl. STM32L4DCMI.

Die Schnittstellenbeschreibung, bzw. der Anschluss der Kameraschnittstelle kann aus der Abbildung 31 und der Abbildung 32 entnommen werden.

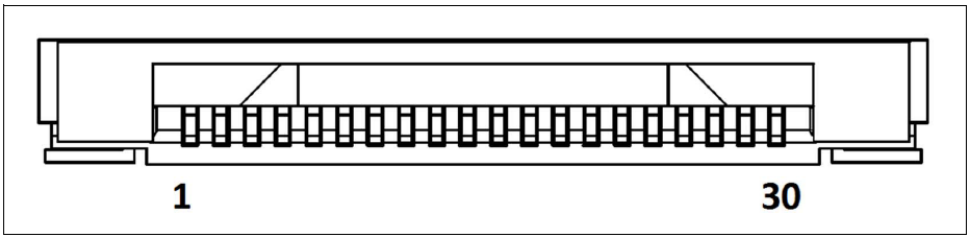


Abbildung 31: CN1 Anschluss Vorderseite; Quelle: UM2160 User manual (2017) S.30

| Pin number | Description | Pin number | Description |
|---|---|---|---|
| 1 | GND | 16 | GND |
| 2 | NC | 17 | DCMI_HSYNC (PH8) |
| 3 | NC | 18 | NC |
| 4 | DCMI_D0 (PH9) | 19 | DCMI_VSYNC (PI5) |
| 5 | DCMI_D1 (PH10) | 20 | VDD |
| 6 | DCMI_D2 (PH11) | 21 | Camera_CLK (MCU PA8) |
| 7 | DCMI_D3 (PH12) | 22 | NC |
| 8 | DCMI_D4 (PH14) | 23 | GND |
| 9 | DCMI_D5 (PI4) | 24 | NC |
| 10 | DCMI_D6 (PE5) | 25 | DCMI_PWR_EN (MFX_GP6) |
| 11 | DCMI_D7 (PI7) | 26 | RESET# |
| 12 | NC | 27 | DCMI_SDA (PB14) |
| 13 | NC | 28 | DCMI_SCL (PH4) |
| 14 | GND | 29 | GND |
| 15 | DCMI_PIXCK (PH5) | 30 | VDD |

Abbildung 32: Schnittstellenbeschreibung CN1; Quelle: UM2160 User manual (2017) S.30[51]

[51] Eine detaillierte Beschreibung der Pins kann dem Handbuch UM2160 User manual (2017) Seite 30 entnommen werden.

### 5.1.3 Direct Memory Access Controller

Der Direkt Memory Access Controller im µC hat Zugriff auf den AHB und APB und ermöglicht einen Peripherie-zu-Speicher-, Speicher-zu-Peripherie-, Peripherie-zu-Peripherie- und Speicher-zu-Speicher-Datenfluss ohne dass die MCU für diese Schreibvorgänge benötigt wird. Im STM32L496x befinden sich zwei DMA-Bausteine, die unabhängig voneinander Aufgaben übernehmen können. Bei einer erfolgreichen oder abgebrochenen Übertragung kann der DMA einen Interrupt generieren, dies gilt auch, wenn die Übertragungshalbzeit erreicht worden ist. Mittels DMA kann eine Datenbreite von 8Bit/16Bit und 32Bit übertragen werden, wobei die Pointer für die Start- und Zieladresse frei konfigurierbar sind. Übertragungen zwischen zwei Speichern werden automatisch, sobald der DMA gültig konfiguriert wurde, ohne Hardware-Request ausgeführt. Bei peripheren Operationen wird ein peripherer Request benötigt, der mittels Software getriggert wird. Die Bausteine DMA1 und DMA2 sind in den Betriebsmodi Run, Sleep, Low-power-run und Low-Power-sleep aktiv, im Stoppmodus wird der Controller pausiert und das aktuelle Speicherbild bleibt erhalten. Bei Power-down oder in Standby wird der gesamte DMA Speicher geleert und der Baustein muss neu initialisiert werden.[52]

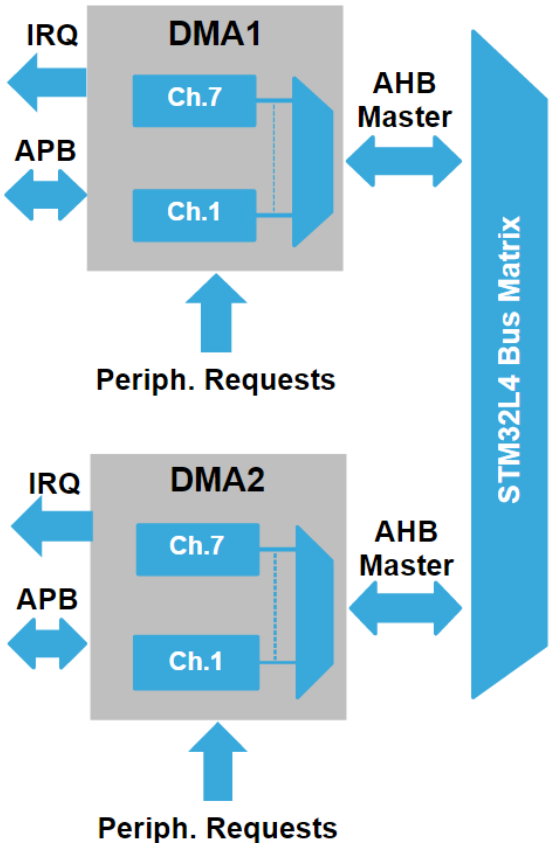


Abbildung 33: DMA; Quelle: STM32L4DMA S.2

### 5.1.4 IDD – Messung

IDD ist eine Hardware- und Software-Implementierung, die es dem Anwender ermöglicht, an der MCU den aktuellen Energieverbrauch zu messen. Dabei unterscheidet man zwischen den IDD-Pins am Board, die es erlauben ein Amperemeter, unter Ausschließung des ST-LINK V2, vor den µC zu schalten und der integrierten IDD-Messung, die mittels Software und einem verbauten IDD-Strommesskreis ermöglicht wird. Die Konfiguration, ob IDD verwendet wird, kann über den Jumper JP2 gesetzt werden, bei Weglassen des Jumpers muss ein Amperemeter zwischengeschaltet werden, damit der µC spannungsversorgt wird. Ein Nachteil der integrierten IDD-Messung mittels Software ist, dass Rechenleistung und Energie der MCU für die Berechnung verbraucht werden und somit das Ergebnis verfälscht wird, jedoch ergibt sich mittels der Software eine einfache Möglichkeit, den Verbrauch in den unterschiedlichen Betriebsmodi zu eruieren.[53]

[52] Vgl. STM32L4DMA.

[53] Vgl. UM2160 User manual (2017) S. 19.

## 5.2 Kameramodule

### 5.2.1 OV9655

Das Kameramodul OV9655 ist ein von OmniVision hergestelltes, weitläufig verbreitetes Kameramodul aus dem Jahre 2006. Es verfügt über ein Image Sensor Array, Timing Generator, Analog Signal Processor, A/D Converters, Digital Signal Processor (DSP), Output Formatter, Scaling Image Output, Strobe Mode, Digital Video Port (CPI) und dem SCCB Interface ($I^2C$). Die für die $I^2C$-Ansteuerung und Konfiguration relevanten Daten müssen dem Datenblatt entnommen werden, die eindeutige $I^2C$-ID des OV9655 ist 0x96. $I^2C$ Befehle für die Kameraregister können über die Adresse 0x60 geschrieben und über die Adresse 0x61 gelesen werden.[54]

Tabelle 2: Wichtige OV9655 Kennzahlen aus dem Datenblatt; Quelle: OV9655 (2006)

| | |
|---|---|
| Bezeichnung: | OV9655 CMOS SXGA |
| Pixel: | 1.3 MP |
| Aktive Auflösung des Sensors: | 1280 x 1024 |
| Unterstütze Auflösung: | SXGA, VGA, CIF und Downscaling |
| 8-Bit Ausgabeformate: | GRB 4:2:2, RGB565/555, YUV (4:2:2), YCbCr (4:2:2) |
| 10-Bit Ausgabeformate: | Raw RGB |
| Maximale Framerate bei SXGA: | 15 fps |
| Typischer Energieverbrauch: | 90 mW |
| Standbystrom: | < 20 µA |

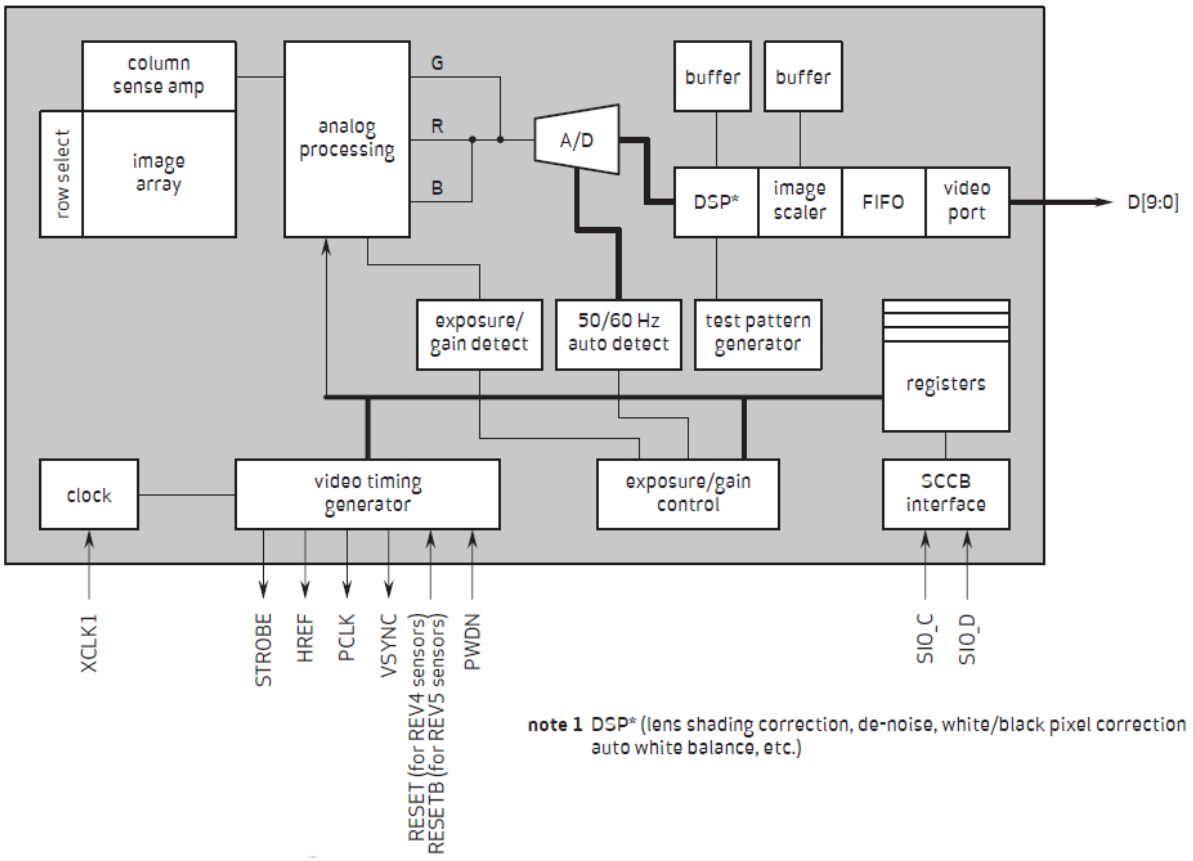


Abbildung 34: Blockschaltbild OV9655; Quelle: OV9655 (2006)

[54] Vgl. OV9655 (2006).

### 5.2.2 OV2640

Das Kameramodul OV2640 ist ein von OmniVision hergestelltes Kameramodul aus dem Jahre 2006. Es verfügt über ein Image Sensor Array, Timing Generator, Analog Signal Processor (AMP), A/D Converters, Digital Signal Processing (DSP), Compression Engine, Output Formatter und einem FIFO Speicher mit Digital Video Port (CPI) und SCCB Interface ($I^2C$) Anschluss. Die für die $I^2C$-Ansteuerung und Konfiguration relevanten Daten müssen dem Datenblatt entnommen werden, die eindeutige $I^2C$ ID des OV2640 ist 0x26 und 0x41. $I^2C$ Befehle für die Kameraregister können über die Adresse 0x60 geschrieben und über die Adresse 0x61 gelesen werden. Das Kameramodul OV2640 bietet als Sonderfunktion die Möglichkeit Bilddaten im JPEG-Format auszugeben. Dieser Modus führt zu einem höheren Energiebedarf des Sensors, ermöglicht aber komprimiertes und somit speicherschonendes Bildmaterial.[55]

Tabelle 3: Wichtige OV2640 Kennzahlen aus dem Datenblatt; Quelle: OV2640 (2006)

| | |
|---|---|
| Bezeichnung: | OV2640 Color CMOS UXGA |
| Pixel: | 2 MP |
| Aktive Auflösung des Sensors: | 1600 x 1200 |
| Unterstütze Auflösung: | UXGA, SXGA, SVGA, VGA, QVGA, QQVGA |
| 8-Bit Ausgabeformate: | GRB 4:2:2, RGB565/555, YUV (4:2:2), YCbCr (4:2:2) |
| 10-Bit Ausgabeformate: | Raw RGB |
| Maximale Framerate bei UXGA: | 15 fps |
| Typischer Energieverbrauch: | 125 mW |
| Standby Energieverbrauch: | 600 µA |

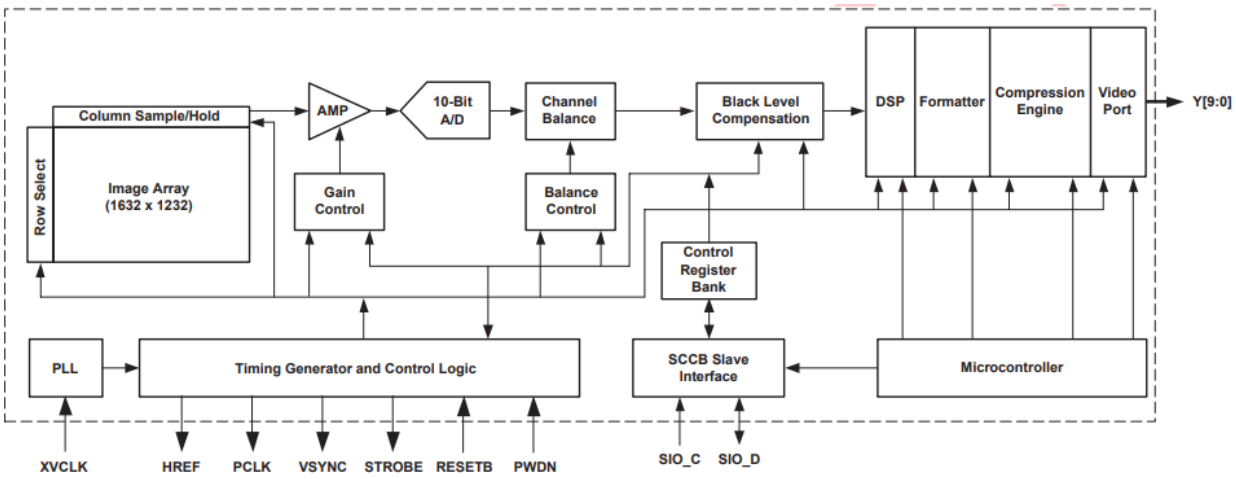


Abbildung 35: Blockschaltbild OV2640; Quelle: OV2640 (2006)

[55] Vgl. OV2640 (2006).

### 5.2.3 OV7670

Das Kameramodul OV7670 ist ein älteres von OmniVision hergestelltes Kameramodul aus dem Jahre 2005. Es verfügt über ein Image Sensor Array, Timing Generator, Analog Signal Processor, A/D Converters, Test Pattern Generator, Digital Signal Processor (DSP), Image Scaler, LED Strobe Flash Control Ouput, Digital Video Port (CPI) und dem SCCB Interface ($I^2C$). Die für die $I^2C$ Ansteuerung und Konfiguration relevanten Daten müssen dem Datenblatt entnommen werden, die eindeutige $I^2C$ ID des OV7670 ist 0x76. $I^2C$ Befehle für die Kameraregister können über die Adresse 0x42 geschrieben und über die Adresse 0x43 gelesen werden.[56]

Tabelle 4: Wichtige OV7670 Kennzahlen aus dem Datenblatt; Quelle: OV7670 (2005)

| | |
|---|---|
| Bezeichnung: | OV7670 CMOS VGA |
| Pixel: | 0.3 MP |
| Aktive Auflösung des Sensors: | 640 x 480 |
| Unterstütze Auflösung: | VGA, CIF |
| 8-Bit Ausgabeformate: | GRB 4:2:2, RGB565/555, YUV (4:2:2), YCbCr (4:2:2) |
| Maximale Framerate bei VGA: | 30 fps |
| Typischer Energieverbrauch: | 60 mW |
| Standby Strom: | < 20 µA |

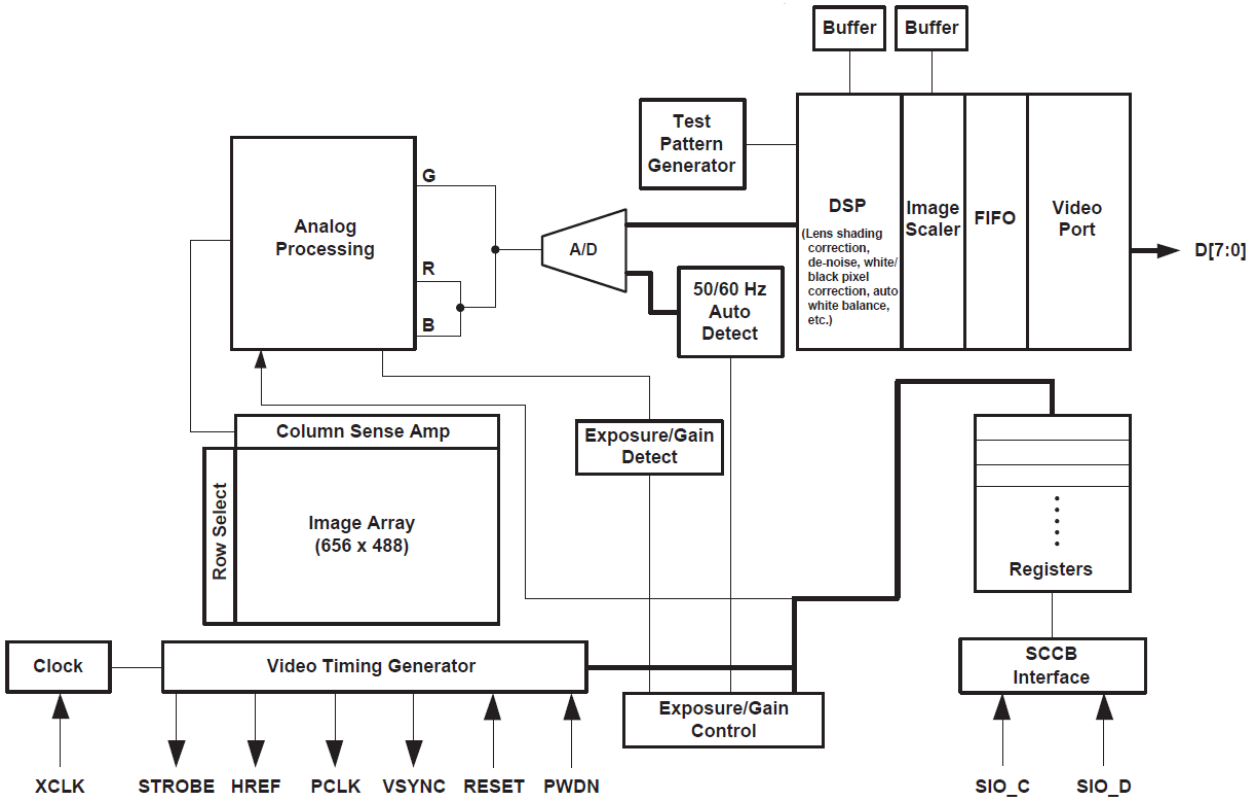


Abbildung 36: Blockschaltbild OV5640; Quelle: OV7670 (2005)

[56] Vgl. OV7670 (2005).

# 6 AUSWERTUNG DER KAMERAMODULE

## 6.1 Beschreibung des Aufbaus

### 6.1.1 Hardware Testaufbau

Um die Kameramodule an den Mikrocontroller anzubinden wurde die CN1 Schnittstelle am µC verwendet. Die Schnittstelle ist über ein Flachbandkabel – FFC mit einem FPC zu DIP Adapter verbunden, der einen Zugriff auf die einzelnen Pins des Kamerainterfaces ermöglicht. Das Kameramodul selbst ist mithilfe von Break-Out-Drähten mit dem DIP-Sockel des Adapters verbunden, um eine flexible Einbindung verschiedener Kameramodule zu ermöglichen. Bei der Einbindung ist zu beachten, dass die CN1-Schnittstelle des STM32L496G-Discovery Boards entgegen der Beschreibung im Handbuch in der aktuellen Version nur ein 8-Bit-CPI unterstützt und somit Kameramodule mit einer breiteren Datenleitung wie der 10-Bit MT9M001-Sensor über diesen Weg nicht eingebunden werden können. Zudem erwies sich der lange Übertragungsweg der Strippen, insbesondere für die Taktleitungen, als äußerst störungsanfällig.

Der Aufbau ist in Abbildung 37 und Abbildung 38 beispielhaft mit dem Kameramodul OV9655 dargestellt, wobei der FPC- zu DIP-Adapter aus Gründen der Veranschaulichung zweimal abgebildet wurde. Die schematische Darstellung des gesamten Aufbaus wird in Abbildung 39 gezeigt, wobei die Verdrahtung beim OV2640 vom dargestellten abweicht, da die Pin-Positionen beim OV2640 unterschiedlich ausgeführt sind. Zudem hat das Kameraboard des OV2640 einen internen Taktgeber verbaut und benötigt somit keinen externen Grundtakt (Camera_CLK).

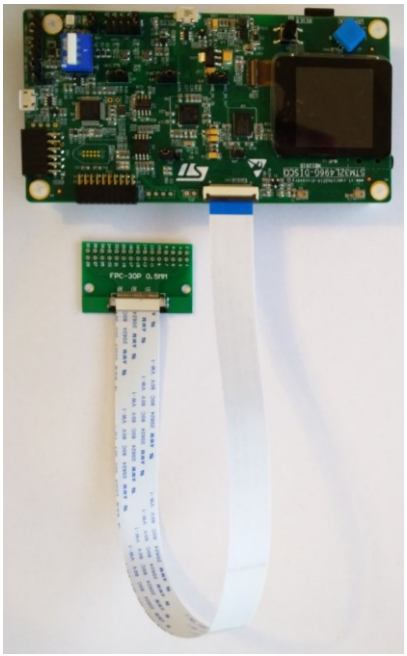

Abbildung 37: Discovery Board verbunden mit einem FPC- zu DIP-Adapter, Quelle: Eigene Darstellung.

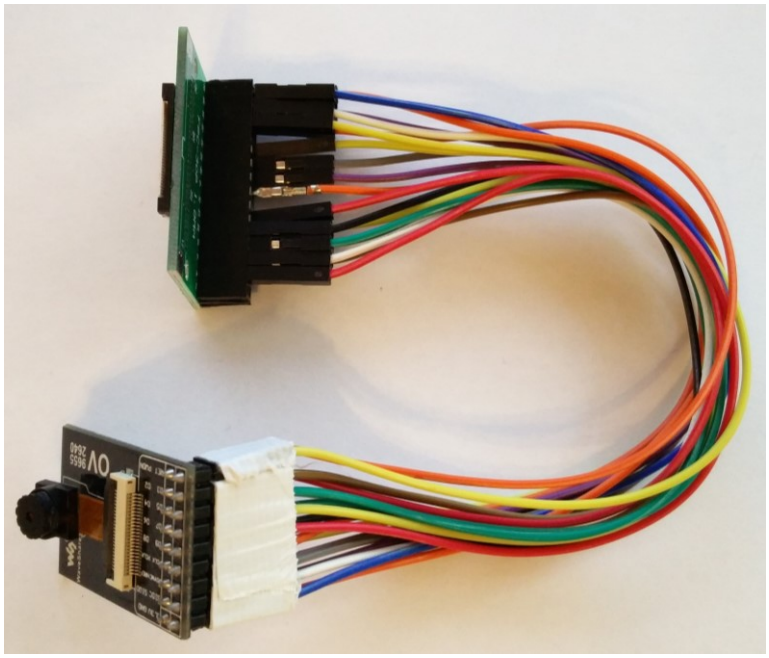

Abbildung 38: FPC- zu DIP-Adapter verbunden mit dem OV9655 Kameramodul, Quelle: Eigene Darstellung.

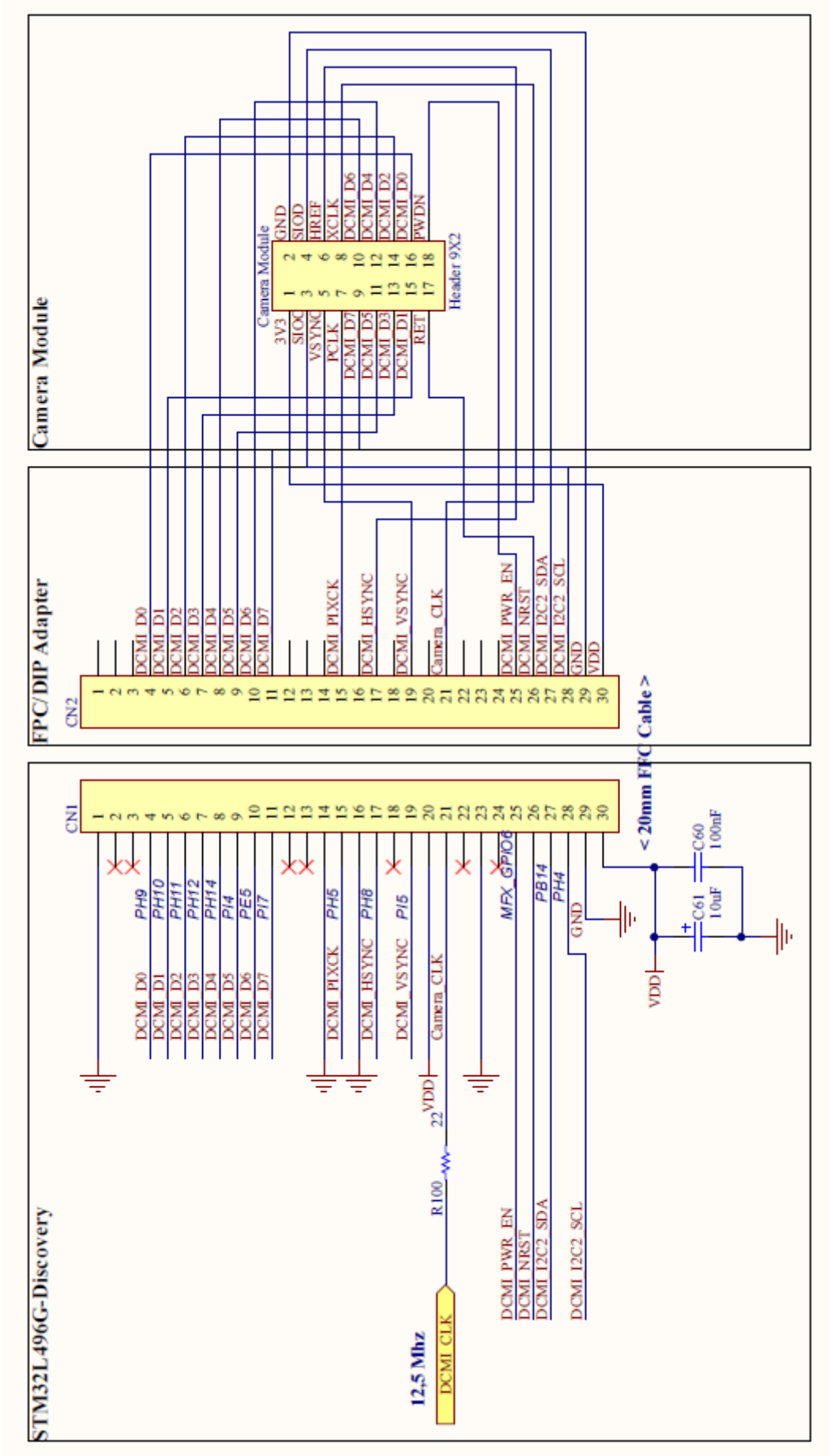


Abbildung 39: Schematische Darstellung der Kameramodulintegration - Allgemein, Quelle: Eigene Darstellung.

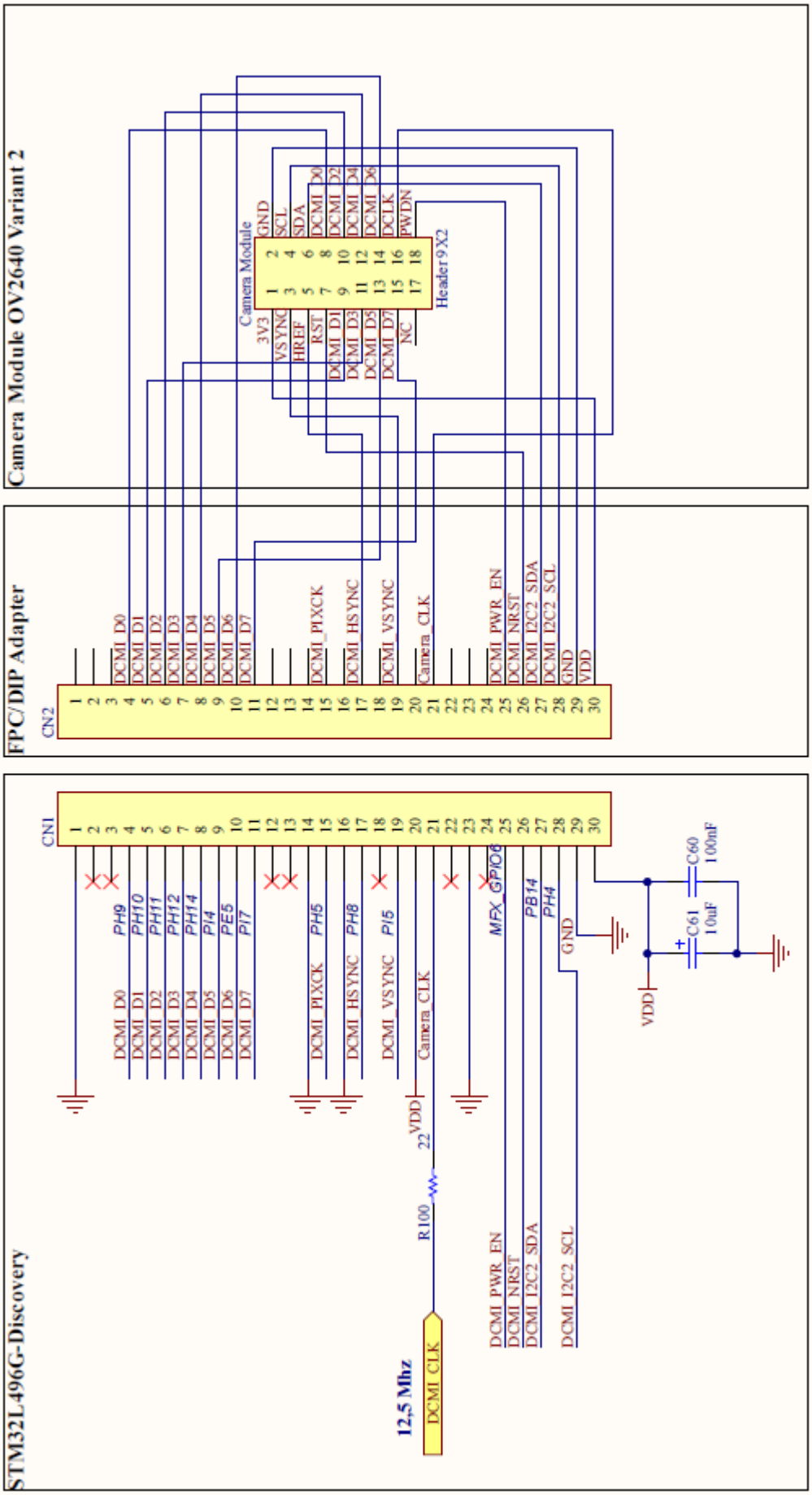


Abbildung 40: Schematische Darstellung der Kameramodulintegration – OV2640 Variante 2, Quelle: Eigene Darstellung.

### 6.1.2 Hardware Leistungsmessung

Die Leistungsmessung am µC-Board wird einerseits über die IDD- (JP2) Schnittstelle des STM32L496G-Discovery und andererseits an der Spannungsversorgung des Kameramoduls selbst vorgenommen. Der Grund für die doppelte Messung ergab sich durch die Limitierung der IDD Schnittstelle, welche nur eine Leistungsmessung des STM32L496AG Mikrocontrollers erlaubt, jedoch nicht periphere Einbindungen wie ein Display oder ein Kameramodul erfasst. Dieses Verhalten der IDD-Schnittstelle kann aus der spärlichen Dokumentation im Handbuch nicht entnommen werden, jedoch ergaben mehrfache Messungen und eine anschließende Recherche, dass die IDD-Leistungsmessung lastunabhängig ist und nur in Abhängigkeit der MCU-Auslastung bzw. dessen Betriebszustandes (Run/Sleep/Deep Sleep) steht. Die Leistungsmessung an der IDD-Schnittstelle wird über Messung der IDD-Spannung und des Stromes durch die Schnittstelle mit einem Multimeter realisiert. Die Messung an der Versorgungsleitung des Kameramodules, im Spannungsbereich, wird über Einbringen eines Messshunts mithilfe eines Oszilloskops durchgeführt, wobei die Spannungsversorgung des Kameramodules und der Spannungsabfall am Shunt gemessen wird. Der Grund für die Verwendung des Messshunts (ohmsche Last) ergibt sich durch die Möglichkeit einer genaueren Messung mithilfe eines Oszilloskops sowie die zeitliche Betrachtung des Energieverbrauches, jedoch muss aufgrund der hohen Ungenauigkeit des Shuntwiderstandes das Ergebnis mit dem Oszilloskop differenziert betrachtet werden. Andere Verbraucher wie das Display oder LEDs, welche nicht über den STM32L496AG spannungsversorgt werden, werden in der Leitungsmessung nicht berücksichtigt da aufgrund des oben beschriebenen Aufbaus diese Komponenten für die Erhebung des Energieverbrauchs keine Verfälschung der Messung verursachen. Der Energieverbrauch der einzelnen Kameramodule wird mit dem Shunt im Leitungs-Pfad berechnet. Eine Rückrechnung des Energieverbrauchs ohne Shunt wird nicht vorgenommen, da die Kameramodule nur bedingt als konstante ohmsche Last gesehen werden können.

Tabelle 5: Messmitteltabelle

| **Messmittel** | **Type** | **Messunsicherheitsangabe für verwendete Bereiche** | **Abschätzung der Messunsicherheit für verwendete Bereiche** |
|---|---|---|---|
| Multimeter | Voltcraft VC 850 | Strommessung: 1,6 % + 8 Counts<br>Spannungsmessung: 1,0 % + 8 Counts | ± (1,6 % + 0,08) mA<br>± (1,0 % + 8) mV |
| Oszilloskop | Fluke 190-104 Scopemeter | Spannungsmessung: 3 % + 10 Counts<br>Vertikale Messung: 2.1 % + 0.04 range/div<br>Horizontale Messung: 100 ppm + 0.04 div | ± (3 % + 1) mV<br>-<br>- |
| Shunt | 1,9 Ohm | 1,0 % + 4 Counts[57] | ± 0,42 Ohm |

[57] Messunsicherheit des für die Messung verwendeten Voltcraft VC 850 Mutimeters für den Lastbereich 0,1 bis 600 Ohm.

### 6.1.3 Software Ansteuerung

Die für die Arbeit erstellte Kameraansteuerung wird auf Basis der STM32L4 Softwarebibliothek und dem beinhalteten Beispiel „DCMI/DCMI_CaptureMode“ [58] entworfen. Die von STMicroelectronics zu Verfügung gestellte Applikation beinhaltet eine DMA-Beispielanwendung für ein OV9655-Kameramodul in QVGA-Auflösung mit anschließender Bildskalierung auf das im 32L496GDISCOVERY verbaute Display. Die in Abbildung 41 gezeigte Projektstruktur beinhaltet die für die Arbeit überarbeitete Projektstruktur, die grob in vier Bereiche unterteil werden kann. Der Ordner „STM32L496G-Discovery“ beinhaltet die Ansteuerung und Auswahl der unterschiedlichen Kameramodule sowie des DMA-Controllers und des LCD. Der Ordner „User“ beinhalt die Main-Funktion des Projektes wo die Initialisierung der Ein- und Ausgänge, der Systemtakte und die Auswahl der Kameraauflösung vorgenommen wird. Der Bereich „Components“ beinhaltet sämtliche Treiber für die Kameramodule und des LCD. Diese Treiber werden unter anderem über „*discovery_camera.c“ und „*discovery_lcd.c“ geladen. Die letzten beiden Verzeichnisse „CMSIS“ und „*Hal_Driver“ beinhalten sämtliche grundlegenden Treiber, die für die Ansteuerung des Mikrocontrollers benötigt werden. Zu diesen gehört ein „Hardware Abstraction Layer“, und Low-Layer-APIs, welche zum Beispiel eine einfache Ansteuerung der I2C-, UART- oder SPI-Schnittstellen ermöglicht. Für die Programmierung in C wurde Keil µVision als Entwicklungsumgebung gewählt, welches sich durch die weite Verbreitung und die nahtlose Unterstützung sämtlicher STM32-Mikrocontroller auszeichnet. Aufgrund der Verwendung einer studentischen Version der Software wird jedoch der Download des kompilierten Codes auf den Flash-ROM mit 32 kB limitiert. Dadurch ist eine vermehrte Nutzung von Makros im Programmcode nötig, welche je nach Verwendung des Kameramodul-Typs unterschiedliche Teilbereiche des Codes kompilieren lässt. Durch diese Speicherlimitierung kann eine automatische Erkennung der Kameramodule im Code nicht integriert werden, und die Auswahl der Module erfolgt über Kompilerdirektiven.

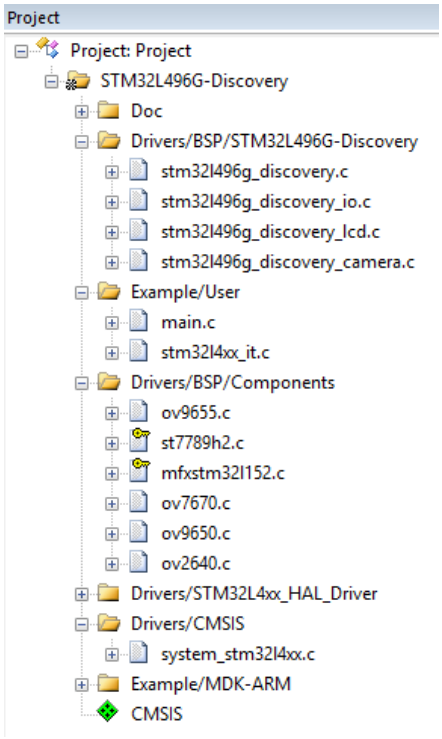


Abbildung 41: Projektstruktur, Quelle: Eigene Darstellung.


Abbildung 42: Sensorauswahl, Quelle: Eigene Darstellung.

[58] STMicroelectronics (2017), Online-Quelle [29.10.2017].

Neben der Limitierung des ROM-Speichers durch die Entwicklungsumgebung wird der SRAM durch den verwendeten Mikrocontroller auf 320 kB begrenzt. Dies führt dazu, dass VGA-Bilder mit 640x480 Pixel bei RGB565 mit einer Speichergröße von 614 kB nicht auf dem SRAM des µC abgelegt werden können. Um dieses Problem zu umgehen, wird der Bildpuffer auf die Größe eines RG565 QVGA-Bildes gesetzt, was 153 kB entspricht und das eingehende VGA-Bild in vier QVGA-Bilder unterteilt. Die umgesetzte Verkleinerung des VGA-Bildes ermöglicht so eine Ablage des QVGA-Bildes im SRAM und somit eine Bearbeitung und Auswertung des Bildmaterials. Bei der angewandten Methode ist zu beachten, dass 4 Frames des Kameramoduls benötigt werden, um ein vollständiges VGA-Bild zu analysieren. Eine weitere Limitierung des Mikrocontrollers ergibt sich durch den DMA-Controller, welcher im STM32L496AG auf 262 kB an Bilddaten pro Frame beschränkt ist, eine mögliche Verwendung eines zweiten DMA-Controllers oder eines Zwischenspeichers, um den Durchsatz zu erhöhen, ist der STM32F4 Produktfamilie vorbehalten und bei einem STM32L4 nicht möglich.

```
#ifdef VGA // Scroll though the VGA Picture (640x480) with 4x 320x240
    if(i == 0x20)
    { HAL_DCMI_Stop(&hDcmiHandler);
      HAL_Delay(250);
      BSP_CAMERA_SAMPLE(CAMERA_R640x480,0,240);
      hal_status = HAL_DCMI_Start_DMA(&hDcmiHandler, DCMI_MODE_CONTINUOUS,  (uint32_t)pBuffer,(320*240)/2 );
    }
    else if (i == 0x40)
    { HAL_DCMI_Stop(&hDcmiHandler);
      HAL_Delay(250);
      BSP_CAMERA_SAMPLE(CAMERA_R640x480,640,0);
      hal_status = HAL_DCMI_Start_DMA(&hDcmiHandler, DCMI_MODE_CONTINUOUS,  (uint32_t)pBuffer,(320*240)/2 );
    }
    else if (i == 0x60)
    { HAL_DCMI_Stop(&hDcmiHandler);
      HAL_Delay(250);
      BSP_CAMERA_SAMPLE(CAMERA_R640x480,640,240);
      hal_status = HAL_DCMI_Start_DMA(&hDcmiHandler, DCMI_MODE_CONTINUOUS,  (uint32_t)pBuffer,(320*240)/2 );
    }
    else if (i == 0x80)
    { HAL_DCMI_Stop(&hDcmiHandler);
      HAL_Delay(250);
      BSP_CAMERA_SAMPLE(CAMERA_R640x480,0,0);
      hal_status = HAL_DCMI_Start_DMA(&hDcmiHandler, DCMI_MODE_CONTINUOUS,  (uint32_t)pBuffer,(320*240)/2 );
      i = 0x00;
    }
#endif
```

Abbildung 43: Zerschneiden eines VGA Bildes zu QVGA, Quelle: Eigene Darstellung.

Die Ansteuerung des Kameramodules erfolgt, je nach Setzen der Kompilerdirektiven, über den Aufruf einer entsprechenden Funktion im Driver-File für die Kamerainitialisierung. Dabei wird das Kameramodul zuerst softwareseitig resettet und alle Einstellungen im Modul verworfen. Anschließend wird via $I^2C$ alle wesentlichen Kameraregister entsprechend den getroffenen Auflösungseinstellungen gesetzt und der DMA-Controller für die Bildübertragung aktiviert.

Zu beachten ist, dass die Registereinstellungen der Kameramodule den jeweiligen Handbüchern der Sensoren entnommen werden müssen und die Einstellungen aufgrund ihrer großen Abweichungen nicht für andere Sensortypen adaptiert werden können und somit individuell ausgearbeitet werden müssen.

```
820  void ov2640_Init(uint16_t DeviceAddr, uint32_t resolution)
821  {
822    uint32_t index;
823
824    /* Initialize I2C */
825    CAMERA_IO_Init();
826
827    /* Prepare the camera to be configured by resetting all its registers */
828    CAMERA_IO_Write(DeviceAddr, 0xFF, 0x01);
829    CAMERA_IO_Write(DeviceAddr, OV2640_SENSOR_COM7, 0x80);
830    CAMERA_Delay(200);
831
832    /* Initialize OV2640 */
833    switch (resolution)
834    {
835    case CAMERA_R160x120:
836      {
837        for(index=0; index<(sizeof(OV2640_QQVGA)/2); index++)
838        {
839          CAMERA_IO_Write(DeviceAddr, OV2640_QQVGA[index][0], OV2640_QQVGA[index][1]);
840          CAMERA_Delay(2);
841        }
842        break;
843      }
844    case CAMERA_R320x240:
845      {
853    case CAMERA_R480x272:
854      {
858    case CAMERA_R640x480:
859      {
867    default:
868      {
871    }
872  }
```

Abbildung 44: Initialisierung des OV2640, Quelle: Eigene Darstellung.

```
 99  const unsigned char ov9655_yuv_sxga[][2]= {
100  { 0x12, 0x80 },{ 0x00, 0x00 },{ 0x01, 0x80 },{ 0x02, 0x80 },{ 0x03, 0x1b },{
101  0x04, 0x03 }, { 0x0e, 0x61 },{ 0x0f, 0x42 },{ 0x11, 0x00 },{ 0x12, 0x02 },{
102  0x13, 0xe7 },{ 0x14, 0x3a },{ 0x16, 0x24 }, { 0x17, 0x1d },{ 0x18, 0xbd },{
103  0x19, 0x01 },{ 0x1a, 0x81 }, { 0x1e, 0x04 }, { 0x24, 0x3c }, { 0x25, 0x36
104  },{ 0x26, 0x72 }, { 0x27, 0x08 }, { 0x28, 0x08 },{ 0x29, 0x15 },{ 0x2a, 0x00
105  },{ 0x2b, 0x00 },{ 0x2c, 0x08 },{ 0x32, 0xff },{ 0x33, 0x00 },{ 0x34, 0x3d
106  },{ 0x35, 0x00 },{ 0x36, 0xf8 },{ 0x38, 0x72 },{ 0x39, 0x57 }, { 0x3a, 0x0c
107  },{ 0x3b, 0x04 },{ 0x3d, 0x99 }, { 0x3e, 0x0c },{ 0x3f, 0xc1 },{ 0x40, 0xd0
108  },{ 0x41, 0x00 },{ 0x42, 0xc0 },{ 0x43, 0x0a },{ 0x44, 0xf0 },{ 0x45, 0x46
109  },{ 0x46, 0x62 }, { 0x47, 0x2a }, { 0x48, 0x3c },{ 0x4a, 0xfc },{ 0x4b, 0xfc
110  },{ 0x4c, 0x7f },{ 0x4d, 0x7f },{ 0x4e, 0x7f },{ 0x52, 0x28 },{ 0x53, 0x88
111  },{ 0x54, 0xb0 },{ 0x4f, 0x98 },{ 0x50, 0x98 },{ 0x51, 0x00 },{ 0x58, 0x1a
112  },{ 0x58, 0x1a },{ 0x59, 0x85 },{ 0x5a, 0xa9 },{ 0x5b, 0x64 },{ 0x5c, 0x84
113  },{ 0x5d, 0x53 },{ 0x5e, 0x0e }, { 0x5f, 0xf0 }, { 0x60, 0xf0 }, { 0x61,
114  0xf0 },{ 0x62, 0x00 }, { 0x63, 0x00 }, { 0x64, 0x02 },{ 0x65, 0x16 },{ 0x66,
115  0x01 },{ 0x69, 0x02 },{ 0x6b, 0x5a }, { 0x6c, 0x04 }, { 0x6d, 0x55 }, {
116  0x6e, 0x00 },{ 0x6f, 0x9d },{ 0x70, 0x21 }, { 0x71, 0x78 },{ 0x72, 0x00 },{
117  0x73, 0x01 },{ 0x74, 0x3a },{ 0x75, 0x35 },{ 0x76, 0x01 },{ 0x77, 0x02 },{
118  0x7a, 0x12 },{ 0x7b, 0x08 }, { 0x7c, 0x15 }, { 0x7d, 0x24 },{ 0x7e, 0x45 },{
119  0x7f, 0x55 },{ 0x80, 0x6a },{ 0x81, 0x78 },{ 0x82, 0x87 },{ 0x83, 0x96 },{
120  0x84, 0xa3 },{ 0x85, 0xb4 }, { 0x86, 0xc3 },{ 0x87, 0xd6 },{ 0x88, 0xe6 },
121  { 0x89, 0xf2 },{ 0x8a, 0x03 }, { 0x8c, 0x0d }, { 0x90, 0x7d }, { 0x91, 0x7b
122  }, { 0x9d, 0x03 },{ 0x9e, 0x04 }, { 0x9f, 0x7a }, { 0xa0, 0x79 }, { 0xa1,
123  0x40 }, { 0xa4, 0x50 },{ 0xa5, 0x68 }, { 0xa6, 0x4a }, { 0xa8, 0xc1 },{
124  0xa9, 0xef }, { 0xaa, 0x92 },{ 0xab, 0x04 },{ 0xac, 0x80 },{ 0xad, 0x80 },{
125  0xae, 0x80 },{ 0xaf, 0x80 },{ 0xb2, 0xf2 },{ 0xb3, 0x20 },{ 0xb4, 0x20 },{
126  0xb5, 0x00 },{ 0xb6, 0xaf },{ 0xbb, 0xae },{ 0xbc, 0x7f },{ 0xbd, 0x7f },{
127  0xbe, 0x7f }, { 0xbf, 0x7f },{ 0xc0, 0xe2 },{ 0xc1, 0xc0 },{ 0xc2, 0x01 },
128  { 0xc3, 0x4e }, { 0xc6, 0x05 },{ 0xc7, 0x80 }, { 0xc9, 0xe0 },{ 0xca, 0xe8
129  }, { 0xcb, 0xf0 },{ 0xcc, 0xd8 },{0xcd, 0x93} ,{ 0xFF, 0xFF } };
130
```

Abbildung 45: YUV SXGA Registereinstellungen für den OV9655, Quelle: Eigene Darstellung.

Die für die Darstellung des Kamerabildes am 240x240 Pixel großen LCD benötigte Funktion HAL_DCMI_ConfigCROP wird von der Beispielapplikation von STM32 größtenteils übernommen. Hierbei wird der OV9655 Sensor mit QVGA (320x240 Pixel) betrieben und das erhaltene Kamerabild auf 240x240 Pixel beschnitten. Die Darstellung der Bilder am Display wird von STMicroelectronics dahingehend realisiert, dass die Speicherstelle im SRAM, von der das LCD-Display seine Bildinformation bezieht, vom DMA-Controller als Speicherstelle für die beschnittene Bildinformation fungiert. Somit werden die Bildinformationen via DMA an den LCD-Puffer übertragen.

```
HAL_DCMI_ConfigCROP(phdcmi,
                    40,               /* Crop in the middle of the VGA picture */
                     0,               /* Same height (same number of lines: no need to crop vertically) */
                     (240 * 2) -1,    /* 2 pixels clock needed to capture one pixel */
                     (240 * 1) - 1); /* All 240 lines are captured */
 HAL_DCMI_EnableCROP(phdcmi);
```

Abbildung 46: Beschneidung des QVGA Bildes auf 240x240 für das LCD-Display, Quelle: Eigene Darstellung.

Bei der Konfiguration des OV2640 Sensors war es trotz Setzen der dementsprechenden Register nicht möglich, das HREF und VSYNC in der für den Mikrocontroller korrekten Polarität zu generieren, weswegen bei Verwendung des OV2640 die Auswertelogik im Programmcode abgeändert worden ist.

```
    /*** Configures the DCMI to interface with the camera module ***/
  /* DCMI configuration */
#ifdef OV2640_Sensor
  phdcmi->Init.CaptureRate      = DCMI_CR_ALL_FRAME;
  phdcmi->Init.HSPolarity       = DCMI_HSPOLARITY_LOW;
  phdcmi->Init.SynchroMode      = DCMI_SYNCHRO_HARDWARE;
  phdcmi->Init.VSPolarity       = DCMI_VSPOLARITY_LOW;
  phdcmi->Init.ExtendedDataMode = DCMI_EXTEND_DATA_8B;
  phdcmi->Init.PCKPolarity      = DCMI_PCKPOLARITY_RISING;
  phdcmi->Init.ByteSelectMode   = DCMI_BSM_ALL;
  phdcmi->Init.LineSelectMode   = DCMI_LSM_ALL;
  phdcmi->Instance              = DCMI;
#endif

#ifndef OV2640_Sensor
  phdcmi->Init.CaptureRate      = DCMI_CR_ALL_FRAME;
  phdcmi->Init.HSPolarity       = DCMI_HSPOLARITY_HIGH;
  phdcmi->Init.SynchroMode      = DCMI_SYNCHRO_HARDWARE;
  phdcmi->Init.VSPolarity       = DCMI_VSPOLARITY_HIGH;
  phdcmi->Init.ExtendedDataMode = DCMI_EXTEND_DATA_8B;
  phdcmi->Init.PCKPolarity      = DCMI_PCKPOLARITY_RISING;
  phdcmi->Init.ByteSelectMode   = DCMI_BSM_ALL;
  phdcmi->Init.LineSelectMode   = DCMI_LSM_ALL;
  phdcmi->Instance              = DCMI;
#endif
```

Abbildung 47: Konfiguration des DCMI nach Logik des HREF und VSYNC Signals, Quelle: Eigene Darstellung.

### 6.1.4 Bestimmung der Bildrate

Die Bildrate der von den Kameramodulen erzeugten Bilder kann entweder mittels Mikrocontroller ausgelesen werden, indem nach Übertragung eines vollständigen Bildes ein Interrupt-Counter verwendet wird oder mittels Oszilloskop an den Taktleitungen. Das für die Bestimmung der Framerate wichtige Signal ist VSYNC, welches das Ende einer Bildübertragung an den Mikrocontroller durch eine logische 1 signalisiert. Die Zeit zwischen zwei VSYNC-Signalen ergibt die Periode, mit der das Kameramodul die Bilder erneuert. Aus ihr kann die Bildfrequenz bzw. die Framerate des Bildes abgeleitet werden. Hierbei ist zu beachten, dass die tatsächliche Bildrate in einer Applikation von der Bildrate des Kameramoduls abweichen kann, da aufgrund von Verzögerungen und Bearbeitungszeiten im µC sogenannte Framedrops entstehen können. Da diese Abweichungen jedoch sehr gering sind, wird näherungsweise die Bildrate des Kameramoduls als die Bildrate im Mikrocontroller angenommen. Ein wesentlicher Einflussfaktor auf die Bildrate des Sensors ist der Eingangstakt XCLK des Moduls, welches in Abbildung 50 mit ca. 20 Mhz ersichtlich ist, ein Anheben des XCLK-Taktes führt zu einer kürzeren Bildverarbeitungszeit im Kameramodul. Der Grundtakt XCLK kann wiederum mittels eines Prescalers im Kameraregister vergrößert werden um so softwareseitig die Framerate anzuheben. Diese Methode wird für die Bestimmung des Energieverbrauchs in Abhängigkeit der Framerate verwendet. Zu beachten ist, dass der DMA Baustein im STM32L496AG Mikrocontroller Bildraten über 50 Hz und unter 0,5 Hz nicht stabil verarbeiten kann und somit das Limit für die Bildrate dementsprechend gesetzt wird.

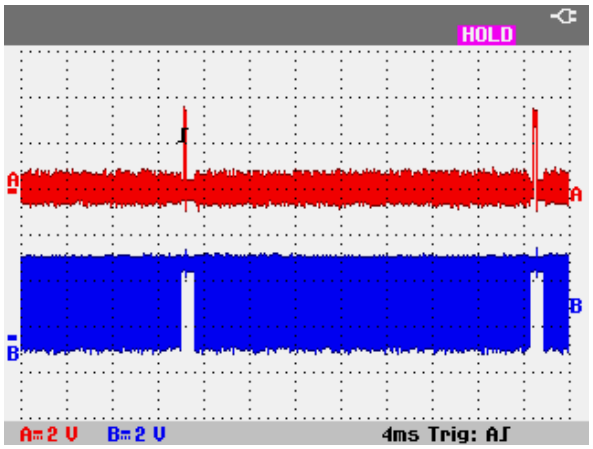


Abbildung 48: Aufnahme von HREF (Blau) und VREF(Rot) für ein Frame, Quelle: Eigene Darstellung.

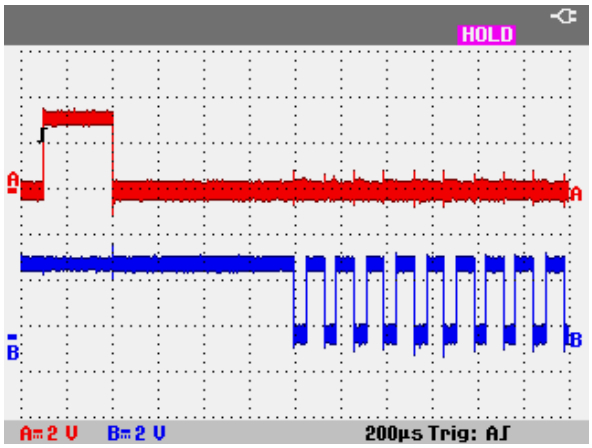


Abbildung 49: Aufnahme von HREF (Blau) und VREF(Rot) im Detail, Quelle: Eigene Darstellung.

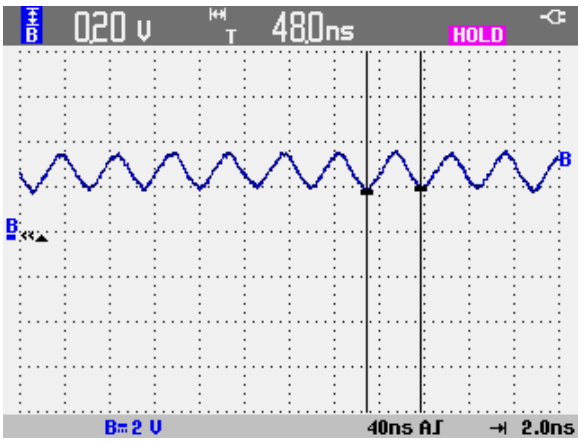


Abbildung 50: XCLK, Quelle: Eigene Darstellung.

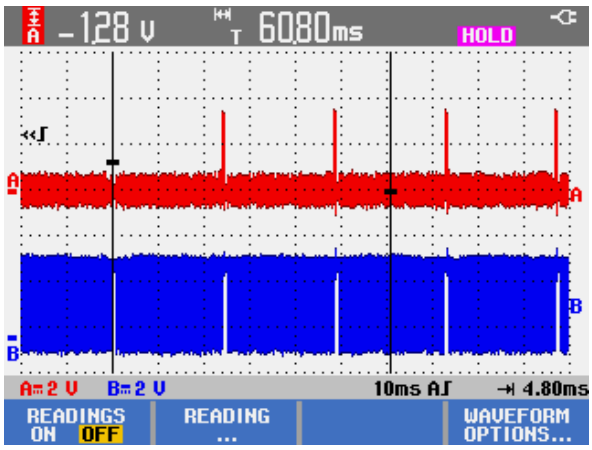


Abbildung 51: Bestimmung der Framerate mithilfe eines Oszilloskops, Quelle: Eigene Darstellung.

### 6.1.5 Bildauswertung

Die Auswertung der Bilddaten erfolgt mithilfe eines 240x240 Pixel Displays am Mikrocontroller, eines Oszilloskops und einer Debug-Software, die speziell in C# erstellt wurde, um das Speicherabbild des µC auszuwerten. Das Display am Mikrocontrollerboard wird vor allem für erste Versuche mit den Kameramodulen verwendet. Es erweist sich als besonders hilfreich, um eine Einschätzung zu bekommen, ob das Kameramodul korrekte Daten schickt und ob der Sensor auf Änderungen des Lichteinfalls reagiert. Aufgrund der Limitierung auf 240x240 Pixel kann das Display jedoch nur begrenzt für VGA-oder QVGA-Auflösungen herangezogen werden, weswegen die weitere Visualisierung am Computer realisiert wird. Abbildung 52 zeigt eine auf 240x240 skalierte Bildaufnahme der Kamera die mithilfe des Displays dargestellt wird. Das in Abbildung 53 ersichtliche Testbild ist ein vom Kameramodul erstelltes Bild, welches vor allem für die Überprüfung der korrekten Kameraparameter, Leitungsstörungen und Einbindung verwendet wird.

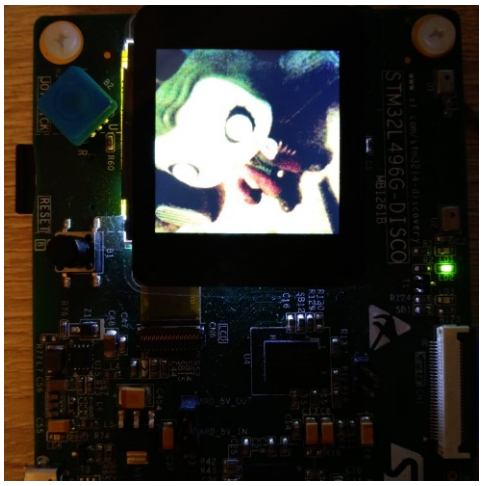


Abbildung 52: Display am µC Board mit Kamerabild, Quelle: Eigene Darstellung.

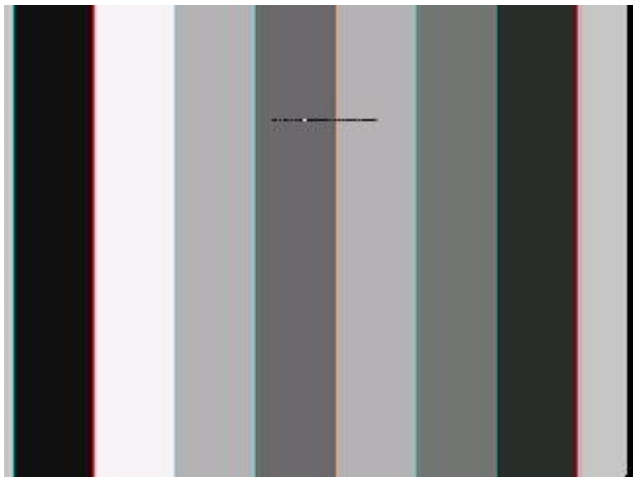

Abbildung 53: Test Pattern eines Kameramoduls, Quelle: Eigene Darstellung.

Für eine Visualisierung der Bilder am Computer müssen die Bilddaten im SRAM des Mikrocontrollers ausgelesen und in einem Dump-File gespeichert werden. Die verwendete Entwicklungsumgebung Keil µVision bietet diese Möglichkeit mithilfe des „SAVE“ Befehls in der Kommandozeile. Um die korrekte Speicheradressen für die Erstellung des Dumps zu finden, wird über die Debug-Funktion von µVision die Speicheranfangsadresse des Bildbuffers im SRAM ausgelesen, die Endadresse des Bildbuffers ergibt sich durch die Bildgröße. Als Beispiel hat ein QVGA-Bild mit einer Auflösung von 320x240 Pixel in Summe 76.800 Pixel in seinem Bild. Aufgrund des verwendeten Bildformats RGB565 mit 2 Byte pro Pixel ergibt sich dadurch eine Gesamtbildgröße von 153,6 kB im SRAM. Die Anfangsadresse des Buffers für das Kamerabild wird von der Entwicklungsumgebung konstant auf „0x200003F8“ gesetzt, somit ergibt sich eine Speicherendadresse für ein QVGA-Bild in hexadezimal von „0x20025bf8“. Die Start- und Endadresse für den Speicherdump wird aufgrund einer einfacheren Auswertung, jedoch auf „0x200003F**0**“ und „0x20025bf**f**“ geändert, was eine gleichbleibende Blockgröße im Speicherdumpfile gewährleistet. Der Kommandozeilenbefehl für das Erstellen des Dumpfiles besteht aus dem Kommando, dem Speicherpfad und der Anfangs- und Endadresse des Speichers. Im Falle eines QVGA-Bildes wird somit in der Entwicklungsumgebung der Befehl „SAVE C:\Users\u13b82\Desktop\dump0.hex 0x200003F0,0x2001C5FF“ verwendet.

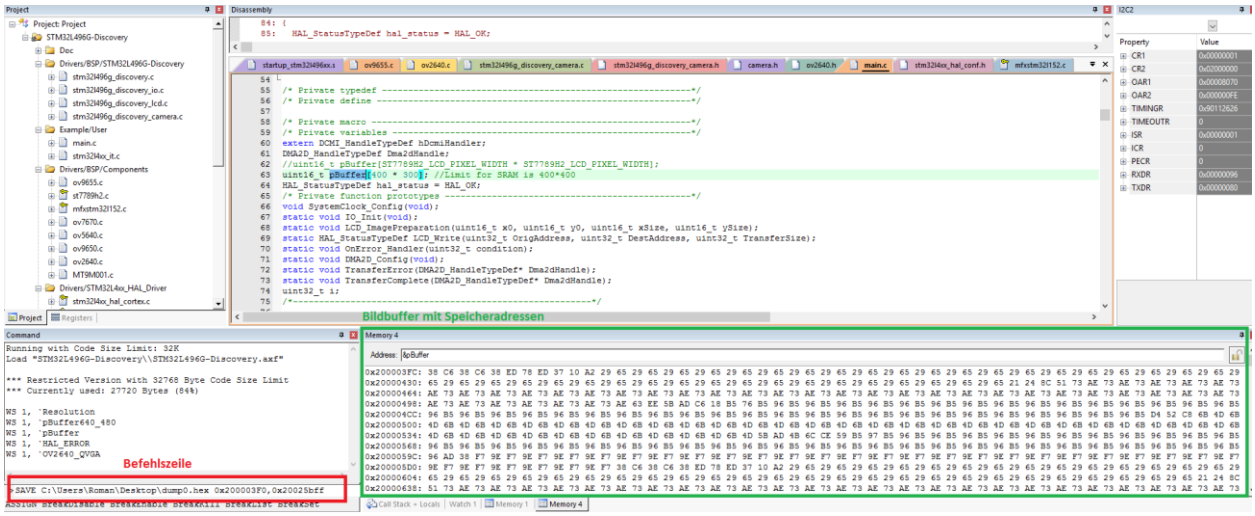


Abbildung 54: Erstellen eines Speicherabbildes in Keil µVision, Quelle: Eigene Darstellung.

Das erstellte Dumpfile wird von Keil µVision im HEX386 Format abgespeichert, welches neben den Speicherdaten auch Zusatzinformationen zu den Daten beinhaltet. So stellt der in Abbildung 55 grün markierte Bereich Informationen über die Anzahl an Bytes pro Linie, die Speicheradresse und den Speichertyp da. Der rote Bereich beinhaltet die Bilddaten und der blaue Bereich ist die Checksumme der Daten. [59]

Das Speicherabbild eines QVGA-Bildes ergibt so ein Textdokument mit 15006 Zeilen an Bildinformation.

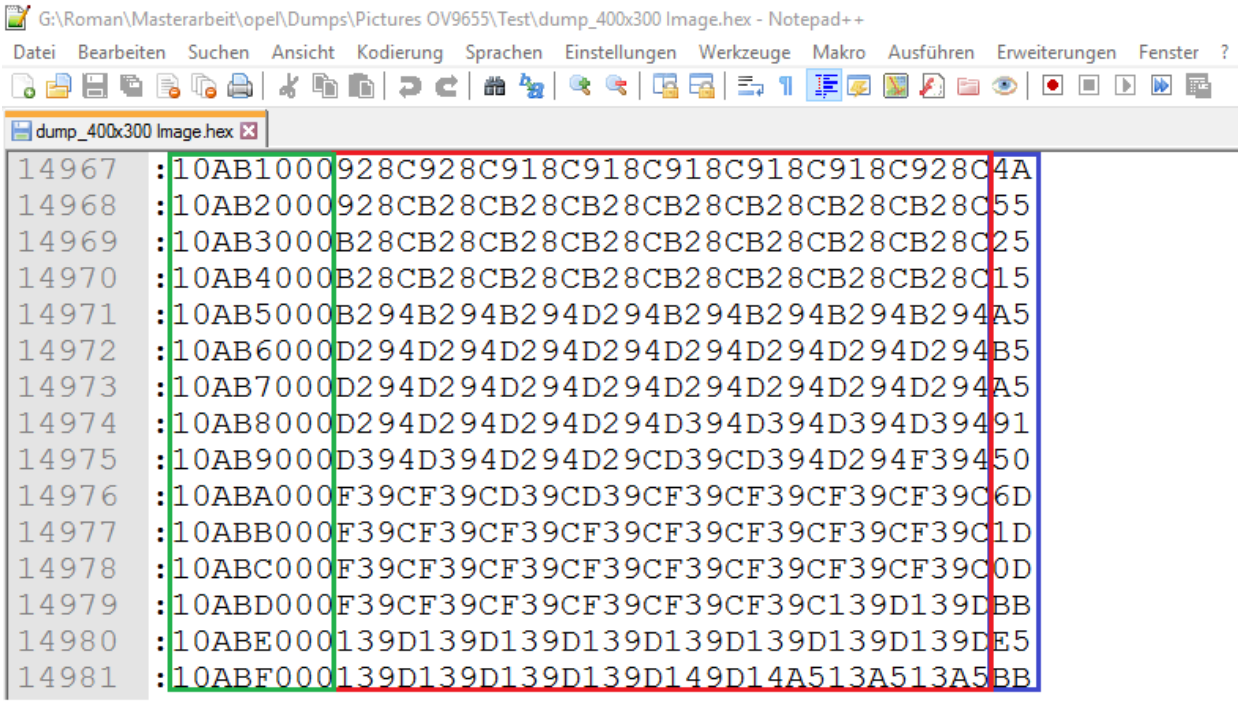


Abbildung 55: Auszug des Speicherabbildes von einem Kamerabild, Quelle: Eigene Darstellung.

Für das Generieren eines für den Computer darstellbaren Bildes wird das Dumpfile in einer in C# selbst erstellten Auswertesoftware eingelesen. Die Zusatzinformationen im Dumpfile werden für die Auswertung verworfen, zudem muss eine einfache Fehlerkorrektur durchgeführt werden, da die Speicherabbilder

[59] Vgl. Arm Limited (2011), Online-Quelle [19.10.2017].

regelmäßig Zeilensprünge mitten im Datenfeld aufweisen. Die bereinigten Daten werden anschließend soweit sortiert, dass zu jedem Pixel im Bild die zugehörigen 2 Bytes an Daten geordnet vorliegen. Anschließend wird das Bildformat RGB565 über Bitshifting und Skalierung in ein für Windows übliches Bildformat, RGB888, umgewandelt und das Bild Pixel für Pixel in einer Picturebox dargestellt. Die graphische Oberfläche des C#-Programms ist in Abbildung 56 ersichtlich, zu beachten ist, dass die Software für Testzwecke entworfen worden ist und dementsprechend nicht ausgereift und mit Bedacht zu verwenden ist. Die Software besteht aus einem einfachen Eingabepfad für das Einlesen des Hexfiles, wobei der Anwender vor dem Einlesen die zu erwartende Bildgröße und das Bildformat eingeben muss. Nach erfolgreichem Einlesen des Files kann das Bild, falls erwünscht, gedreht und als „jpeg“ abgespeichert werden. Die Funktion „Offset Dump“ erlaubt die Korrektur von Bildzeilen oder Linien, welche auf der falschen Seite des Bildes angehängt worden sind. Es kann nicht zur Gänze eruiert werden, wieso dieser sensor- und auflösungs-abhängige Offset zustande kommt, eine Vermutung ist jedoch, dass der Microcontroller nach Erhalt des VSYNC-Signals noch Zeilen vom nächsten Bild in die Anfangsadresse des Buffers schreibt. Eine weitere wichtige Funktion ist die Checkbox „Invert RGB565 Word“, welche besonders beim Kamerasensor OV7670 als Korrektur zum Einsatz kommt. Entgegen der Moduldokumentation werden Bilddaten vom Sensor byte-vertauscht verschickt (endianness), nicht zu verwechseln mit der LSB- und MSB-Anordnung, welche mittels eines Registers am Kameramodul verändert werden kann. Dies bedeutet, dass das erste Byte des Bildinformations-Words als Zweites kommt und das zweite Byte als Erstes, die Bitreihenfolge selbst wird nach Vertausch der Bytes vom Sensor OV7670 so wie erwartet geliefert. Abbildung 57 zeigt ein Falschfarb-Bild des OV7670, welches noch nicht in der Software korrigiert wurde.

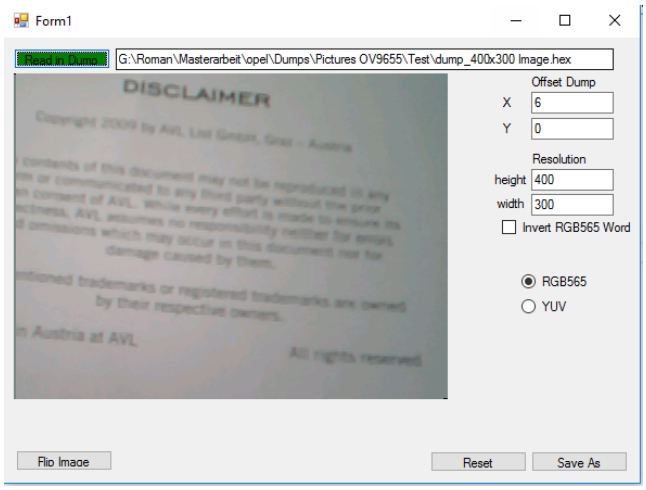


Abbildung 56: C# Auswertesoftware mit Bildvorschau, Quelle: Eigene Darstellung.

Abbildung 57: Bytevertauschtes Bild, Quelle: Eigene Darstellung.

Wesentlich für die die Konvertierung von RGB565 zu RGB888 ist der Aufbau des jeweiligen Datenformats. So besteht RGB888 aus 8 Bits für Rot, 8 Bits für Grün und 8 Bits für Blau, im Gegensatz dazu hat RGB565 nur 5 Bits für Rot, 6 Bits für Grün und 5 Bits für Blau. Um das jeweilige Bildformat in das andere zu überführen, gilt es, die Farbbits im Datenword je nach Farbe zu verschieben und zu maskieren. Anschließend müssen die Farbwerte skaliert werden, um sie in einen ädequaten Sättigungswert des anderen Bildfomrats zu überführen.

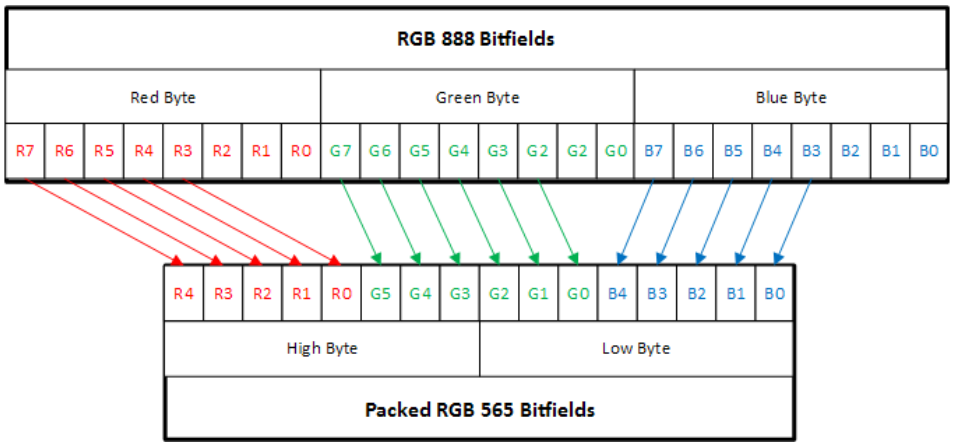


Abbildung 58: Zusammenhang zwischen RGB565 und RGB888, Quelle: http://alfredoer.com/wp-content/uploads/2015/01/RGB888convertToRGB565.png

```
private void RGB565_to_color()
{
    byte  red, green, blue;
    byte byte_temp_r;
    byte byte_temp_g;
    byte byte_temp_b;

    for (int i = 0; i < pixel.Length; i++)
    {
        byte_temp_r = (byte)((pixel[i] >> 11) & 0x1F);
        byte_temp_g = (byte)((pixel[i] >> 5) & 0x3F);
        byte_temp_b = (byte)(pixel[i] & 0x1F);
        red = (byte)((byte_temp_r * 527 + 23) >> 6);
        green= (byte)((byte_temp_g * 259 + 33) >> 6);
        blue =(byte)((byte_temp_b * 527 + 23) >> 6);
        pic.Add(red);
        pic.Add(green);
        pic.Add(blue);
    }
    red = 0;
}
```

Abbildung 59: RGB565 zu RGB888, Quelle: Eigene Darstellung.

```
private void Draw_Image(int width,int height, int bpp)
{
    int start = Convert.ToInt32(textBox1.Text)*3;
    int starty = Convert.ToInt32(textBox2.Text) * 3;
    Bitmap bmp = new Bitmap(width, height);
    for (int y = 0; y < height; y++)
    {for (int x = 0; x < width; x++)
        {int i = ((y * width) + x) * (bpp / 8);
            if (bpp == 24) // in this case you have 3 color values (red, green, blue)
            {if (i < start)
                {// first byte will be red, because you are writing it as first value
                    byte r = pic[i];
                    byte g = pic[i + 1];
                    byte b = pic[i + 2];
                    Color color = Color.FromArgb(r, g, b);
                    bmp.SetPixel(x, y, color);
                } else
                {// first byte will be red, because you are writing it as first value
                    byte r = pic[i+start];
                    byte g = pic[i+start + 1];
                    byte b = pic[i+start + 2];
                    Color color = Color.FromArgb(r, g, b);
                    bmp.SetPixel(x, y, color);
                } } }
    }
    bmp.RotateFlip(RotateFlipType.Rotate180FlipNone);
    dump_picture.Image = bmp;
    Picture = bmp;
    dump_picture.Size = new System.Drawing.Size(Convert.ToInt32(res_height.Text), Convert.ToInt32(res_width.Text));
}
```

Abbildung 60: Übernommener und adaptierter Code zum Erstellen eines Bildes Pixel für Pixel, Quelle: Eigene Darstellung.

## 6.2 Allgemeiner Vergleich

Die für die Arbeit verwendeten CMOS-Kameramodule, siehe Abbildung 61 bis Abbildung 64, unterscheiden sich nicht nur anhand des Sensortyps, sondern auch in ihrer nativen Auflösung und dem Aufbau des Kameraboards selbst. Ein wesentlicher Unterschied zwischen den Kameramodulen sind auch die verwendeten Linsen, die entweder eine starre oder einstellbare Brennweite ermöglichen. Die Verwendung eines verstellbaren Objektives gestattet die Adjustierung des Brennpunktes und trägt somit wesentlich bei die Bildqualität zu verbessen, bedarf aber eines größeren Aufbaus. Neben der einheitlichen 8-Bit CPI–Schnittstelle, die alle Module gemeinsam haben, gibt es Unterschiede in der Generierung des Systemtaktes XCLK, welcher bei dem Modul des OV2640 intern über einen Schwingkreis generiert wird und bei den anderen Modulen extern über den Pin XCLK zugeführt werden muss. Die Ansteuerung der Sensoren selbst folgt, abgesehen von unterschiedlichen Registereinstellungen, analog zueinander, jedoch ergaben sich wesentliche Unterschiede in der Ausgabe des Bildmaterials. Der Sensor OV7670 vertauscht die RGB565 Bytes was auf einen Unterschied in der Endianness zwischen der Ausgabe der Datenwörter zurück zu führen ist. Der Sensor OV2640 gibt entgegen seiner Registereinstellungen eine negierte HREF und VSYNC Logik aus. Ein wesentlicher Unterschied zwischen der Belichtung der Bilder und der jeweiligen Größe der Bildpixel kann nicht festgestellt werden, jedoch hat bei allen Kameramodulen die Bildrate einen wesentlichen Einfluss auf die Belichtungszeit der Sensorarrays. Der Sensor OV2640 wird einmal als Sensorboard mit verstellbarer Linse und fixem Systemtakt und einmal als Modul auf dem OV9655 Board getestet.

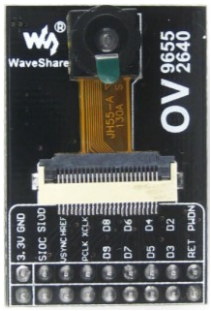


Abbildung 61: OV9655, Quelle: https://www.waveshare.com/img/devkit/accBoard/OV9655-Camera-Board/OV9655-Camera-Board-3.jpg

Abbildung 62: OV2640 Variante1, Quelle: http://www.arducam.com/wp-content/uploads/2012/11/OV2640_5T.jpg

Abbildung 63: OV7670, Quelle: https://core-electronics.com.au/media/catalog/product/cache/1/image/650x650/fe1bcd18654db18f328c2faaaf3c690a/d/e/device10_1000.jpg.

Abbildung 64: OV2640 Variante2, Quelle: https://ae01.alicdn.com/kf/HTB1SvuISVXXXXcrXFXXq6xXFXXXi/OV2640-camera-module-Module-2-million-pixel-electronic-integrated-with-jpeg-compression-new-big-promotion.jpg_640x640.jpg

## 6.3 Energieverbrauch

In folgenden Kapitel wird der Leistungsverbrauch der unterschiedlichen Kameramodule in Abhängigkeit von der Bildrate und der Auflösung dargestellt. Die Werte wurden wie in Kapitel 6.1.2 und 6.1.4 beschrieben gemessen.

### 6.3.1 OV2640 Variante 1

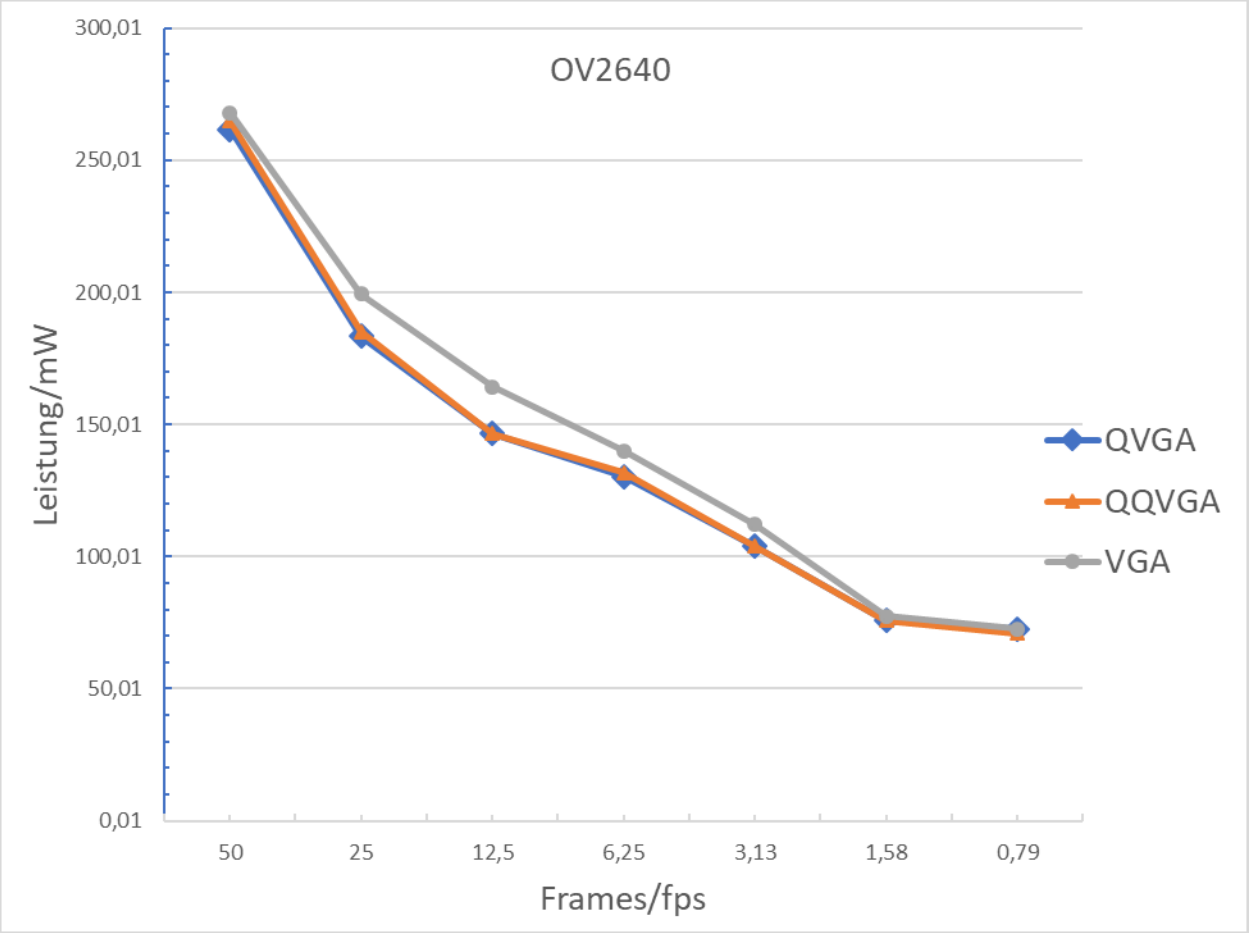


Abbildung 65: Energieverbrauch des OV2640 Variante 1, Quelle: Eigene Darstellung.

Tabelle 6: Energieverbrauch des OV2640 Variante 1, Quelle: Eigene Darstellung.

| Frames/fps | VGA-Verluste/mW | QVGA-Verluste/mW | QQVGA-Verluste/mW |
|---|---|---|---|
| 50 | 267,91 | 264,83 | 261,75 |
| 25 | 199,27 | 184,98 | 183,38 |
| 12,5 | 164,18 | 146,45 | 146,45 |
| 6,25 | 139,97 | 131,85 | 130,22 |
| 3,13 | 112,24 | 104,03 | 104,03 |
| 1,58 | 77,57 | 75,91 | 75,91 |
| 0,79 | 72,58 | 70,91 | 72,58 |

### 6.3.2 OV2640 Variante 2

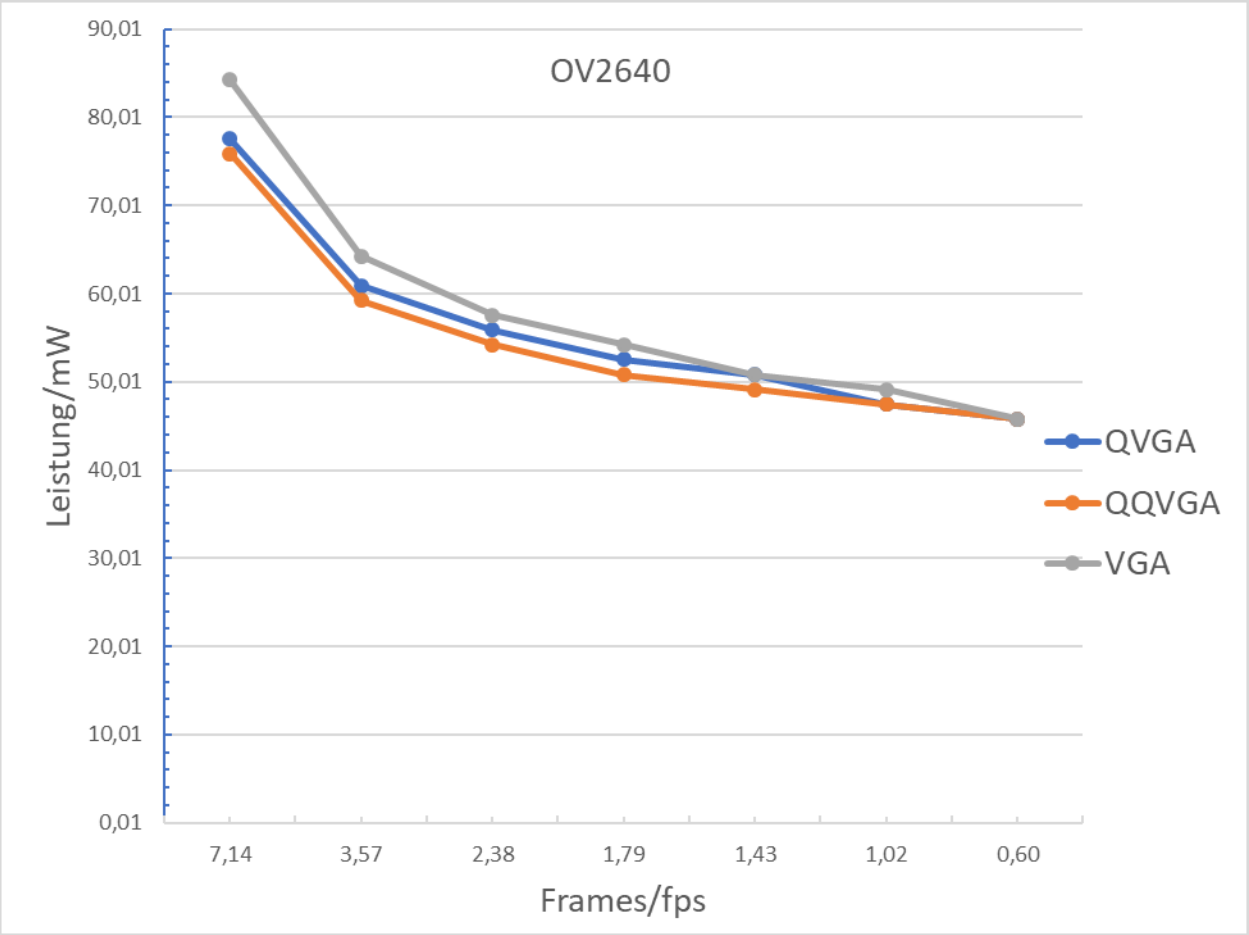


Abbildung 66: Energieverbrauch des OV2640 Variante 2, Quelle: Eigene Darstellung.

Tabelle 7: Energieverbrauch des OV2640 Variante 2, Quelle: Eigene Darstellung.

| Frames/fps | VGA-Verluste/mW | QVGA-Verluste/mW | QQVGA-Verluste/mW |
|---|---|---|---|
| 7,14 | 84,21 | 77,57 | 75,91 |
| 3,57 | 64,24 | 60,90 | 59,22 |
| 2,38 | 57,55 | 55,87 | 54,20 |
| 1,79 | 54,20 | 52,52 | 50,84 |
| 1,43 | 50,84 | 50,84 | 49,16 |
| 1,02 | 49,16 | 47,48 | 47,48 |
| 0,6 | 45,80 | 45,80 | 45,80 |

### 6.3.3 OV9655

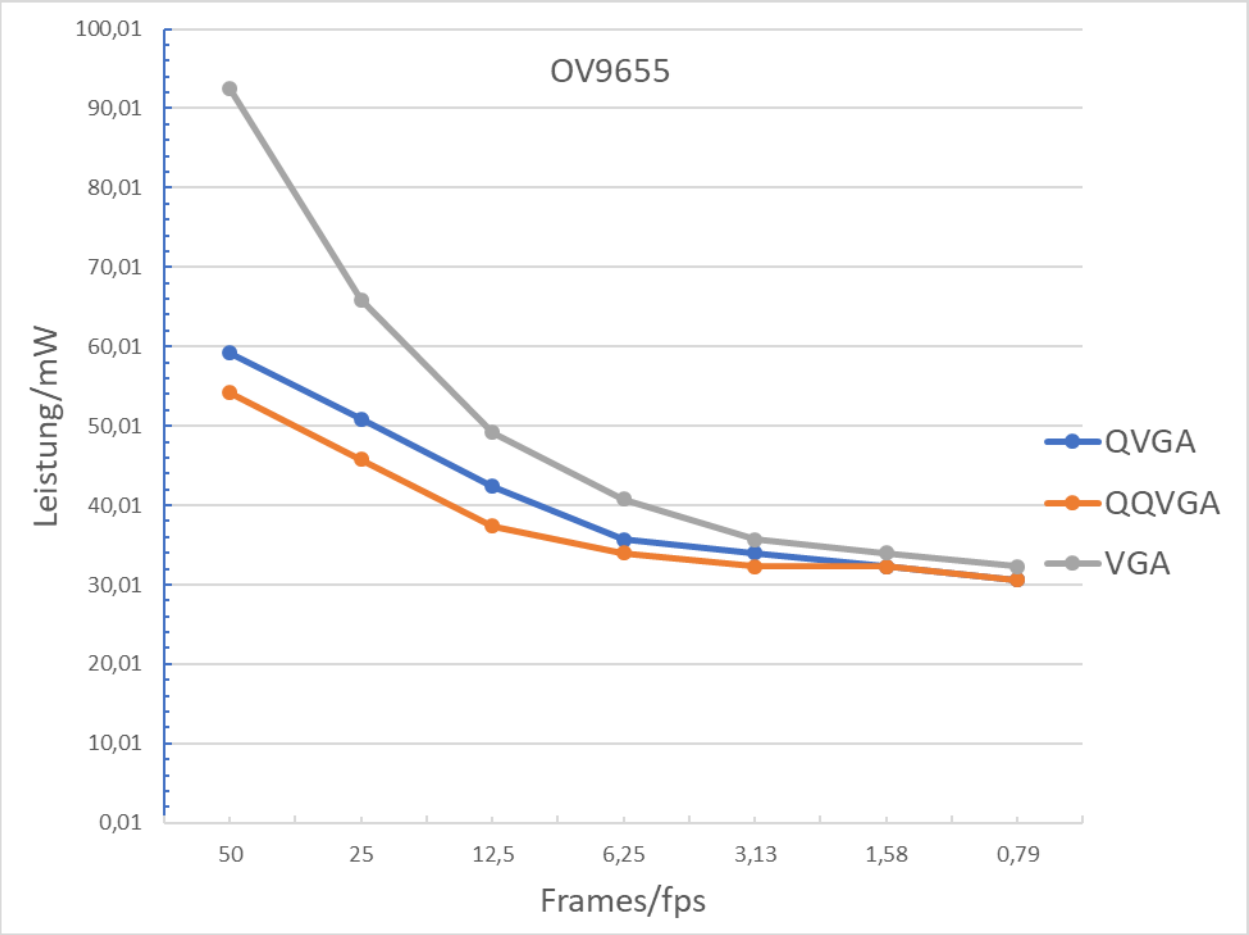


Abbildung 67: Energieverbrauch des OV9655, Quelle: Eigene Darstellung.

Tabelle 8: Energieverbrauch des OV9655, Quelle: Eigene Darstellung.

| Frames/fps | VGA-Verluste/mW | QVGA-Verluste/mW | QQVGA-Verluste/mW |
|---|---|---|---|
| 50 | 92,49 | 59,22 | 54,20 |
| 25 | 65,91 | 50,84 | 45,80 |
| 12,5 | 49,16 | 42,43 | 37,38 |
| 6,25 | 40,75 | 35,69 | 34,00 |
| 3,13 | 35,69 | 35,00 | 32,31 |
| 1,58 | 34,00 | 32,31 | 32,31 |
| 0,79 | 32,31 | 30,62 | 30,62 |

### 6.3.4 OV7670

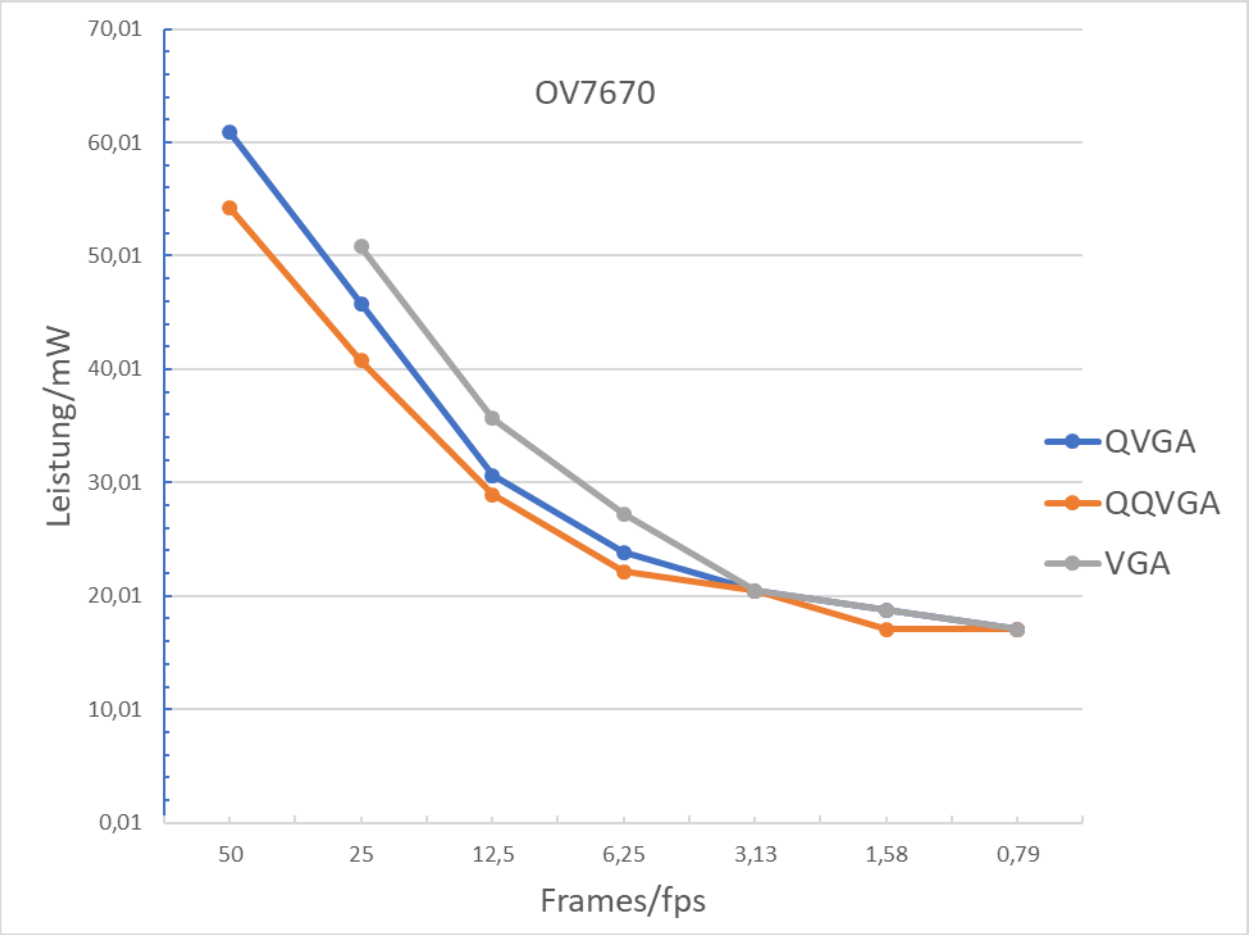


Abbildung 68: Energieverbrauch des OV7670, Quelle: Eigene Darstellung.

Tabelle 9: Energieverbrauch des OV7670, Quelle: Eigene Darstellung.

| Frames/fps | VGA-Verluste/mW | QVGA-Verluste/mW | QQVGA-Verluste/mW |
|---|---|---|---|
| 50 | - | 60,90 | 54,20 |
| 25 | 50,84 | 45,80 | 40,75 |
| 12,5 | 35,69 | 30,62 | 28,93 |
| 6,25 | 27,23 | 23,84 | 22,15 |
| 3,13 | 20,23 | 20,45 | 20,45 |
| 1,58 | 18,75 | 18,75 | 17,05 |
| 0,79 | 17,05 | 17,05 | 17,05 |

## 6.4 Bildvergleich

In folgenden Kapitel werden die aufgenommenen Bilder der unterschiedlichen Kameramodule dargestellt. Die Aufnahmen sind wie in Kapitel 6.1.2 beschrieben erstellt.

### 6.4.1 QQVGA

Abbildung 69: QQVGA OV2640 Variante2, Quelle: Eigene Darstellung

Abbildung 70: QQVGA OV7670, Quelle: Eigene Darstellung

Abbildung 71: QQVGA OV9655, Quelle: Eigene Darstellung

Abbildung 72: QQVGA OV2640 Variante1, Quelle: Eigene Darstellung

### 6.4.2 QVGA

Abbildung 73: QVGA OV2640 Variante 2, Quelle: Eigene Darstellung

Abbildung 74: QVGA OV7670, Quelle: Eigene Darstellung

Abbildung 75: QVGA OV9655, Quelle: Eigene Darstellung

Abbildung 76: QVGA OV2640 Variante 1, Quelle: Eigene Darstellung

### 6.4.3 VGA

Abbildung 77: VGA OV2640 Variante 1, Quelle: Eigene Darstellung

Abbildung 78: VGA OV2640 Variante 2, Quelle: Eigene Darstellung

Abbildung 79: QVGA OV7670, Quelle: Eigene Darstellung

Abbildung 80: VGA OV9655, Quelle: Eigene Darstellung[60]

[60] Die Ursache für die Bildartefakte wird in Kapitel 8.1 beschrieben.

# 7 BEISPIELAPPLIKATION

## 7.1 Sensorauswahl

Aufgrund des niedrigen Energieverbrauches des OV7670 und dessen verstellbarer Linse wird der Sensor als Erstes für die Beispielapplikation ausgewählt. Jedoch erschwert die schlechte Lichtempfindlichkeit und der eingeschränkte Kontrast des Sensors, siehe Abbildung 70, die Applikation erheblich, sodass der Sensor nicht sinnvoll eingesetzt werden kann. Aus diesem Grund wird der Sensor OV9655 mit dem zweitniedrigsten Energieverbrauch für die Anwendung ausgewählt. Abstriche müssen dadurch jedoch bei der Schärfe des Bildes gemacht werden. Der Sensor OV2640 wird aufgrund seines hohen Energieverbrauches nicht für die Anwendung in Betracht gezogen, jedoch wäre sowohl die Bildschärfe, der Kontrast und die Lichtempfindlichkeit für eine Auswertung optimal gewesen.

## 7.2 Beschreibung der Funktion

Aufgrund eines fehlenden Objektives und der dadurch resultierenden unscharfen Bilder für das Kameramodul OV9655 kann eine mögliche Objekt- oder Schrifterkennung über die Kanten, Konturen im Bild von Anfang an ausgeschlossen werden. Der hervorragende Kontrast der Kamerabilder und die passable Lichtausbeute sprechen jedoch für eine Bildauswertung über die Farbinformation des Bildes. Die Beispielapplikation, die im Sinne von Low-Power erstellt wird, wertet dabei zweimal pro Sekunde ein Kamerabild aus, das ebenfalls mit 2 fps vom Kameramodul zu Verfügung gestellt wird. Um hierbei unnötige MCU-Last zu vermeiden, wird dabei nur ein Bild in QQVGA-Auflösung vom Kameramodul zur Verfügung gestellt und analysiert. Die Auswertung des Bildes beinhaltet die Zerlegung der RGB565-Bildinformation Pixel für Pixel in dessen einzelne Rot-, Grün- und Blauwerte und die anschließende Bewertung, ob wesentliche Änderungen an der Summe aller Farbkanäle erkennbar sind. Bei Detektion einer Änderung wird das LCD initialisiert und die Kameraauflösung auf QVGA erhöht. Anschließend wird vom detektierten Bild eine Kopie in QVGA am Display für 5 Sekunden angezeigt und danach das Kameramodul und der Mikrocontroller in die QQVGA Einstellung mit Bildauswertung zurückversetzt. Um Energie einzusparen, werden soweit wie möglich alle unbenutzten I/Os auf den Modus „analoger-Eingang“ geschaltet, dies deaktiviert die internen Schmidt-Trigger der Eingänge und spart so Energie. Sämtliche LEDs, die von der MCU angesteuert werden, werden deaktiviert und die MCU bei Leerlauf in den Sleep-Modus gesetzt. Das Kameramodul OV9655 wird durch die niedrige Auflösung und Bildrate energiesparend eingebunden.

Die Beispielfunktion wird anhand eines roten Balls getestet, befindet sich das rote Objekt im QQVGA-Bild, wird nach 1-2 Sekunden von dem Objekt ein Schnappschuss in QVGA erstellt und skaliert auf 240x240 Pixel am LCD Display angezeigt. Bei der Anwendung kann eine große Abhängigkeit der Geschwindigkeit und Fehlerrate bei der Erkennung durch die gegebenen Lichtverhältnisse festgestellt werden.

Abbildung 81: QQVGA Bild, die Kamera erkennt ein rotes Objekt, Quelle: Eigene Darstellung

Abbildung 82: QVGA-Bild der Kamera skaliert auf 240x240, Quelle: Eigene Darstellung

## 7.3 Beschreibung des Aufbaus

### 7.3.1 Hardware

Die verwendete Hardware für die Beispielapplikation entspricht dem in Kapitel 6.1.1 für die Auswertung der einzelnen Kameramodule beschriebenen Aufbau. Der verwendete Sensor OV9655 wird, wie in Abbildung 39 schematisch abgebildet, integriert. Energiesparende Optimierungen wie das Entfernen von LEDs oder unbenutzten Modulen / ICs vom STM32l496g-Discovery Board werden nicht vorgenommen, da auf die Wiederverwendbarkeit des Mikrocontrollerboards Rücksicht genommen wurde. Um den Energieverbrauch der Beispielapplikation zu messen, wird wieder ein Messshunt in Serie in die IDD-Schnittstelle beziehungsweise in die Spannungsversorgung geschaltet, wodurch eine Abschätzung der Verlustleistung ermöglicht wird. Neben dem Microcontroller, dem Kameramodul und der Verbindungsleitungen, bzw. dem Adapterboard wird auch ein roter Ball für die Beispielapplikation verwendet.

### 7.3.2 Software

Das Grundgerüst des Mikrocontrollerprogramms wird mithilfe der STM32CubeMX-Software erstellt, ein von STMicroelectronics zur Verfügung gestelltes Tool, welches eine einfache Konfiguration aller verwendeten Schnittstellen, Taktraten und Controller-Einstellungen vornimmt. Für die Konfiguration wird die Application-Note „Digital Camera Interface (DCMI) for STM32 MCUs„[61] herangezogen, welche eine Einbindung beispielhaft für ein STM32F746G-DISCOVERY Board erklärt. Für die Erstellung des Beispielcodes wird in der STM32CubeMX Software die verwendete MCU – STM32L496AGIx ausgewählt und die benötigte Schnittstelle DCMI mit 8-Bit aktiviert, das LCD Panel kann in der Software nicht bei Verwendung der DCMI-Schnittstelle aktiviert werden und musste nachträglich im Programmcode eingebunden werden.

[61] AN5020 (2017).

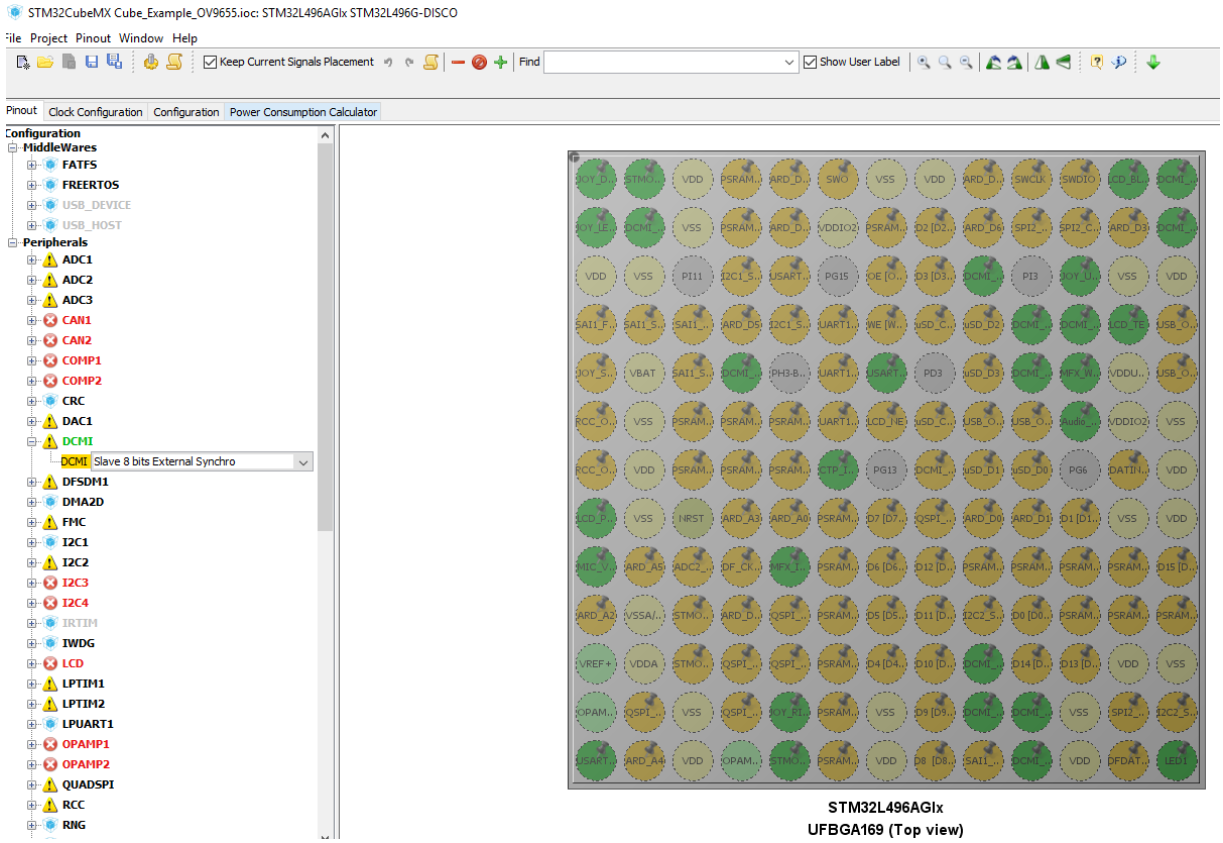


Abbildung 83: STM32CubeMX, Quelle: Eigene Darstellung

Die Taktrate der MCU und des DMA-Controllers wird anschließend in der „Clock-Configuration“ auf 80 Mhz gesetzt, dies ist die schnellstmögliche Taktrate der MCU. Der hohe Takt wird aufgrund des benötigen Systemtaktes für das Kameramodul von 20 Mhz, die benötige Rechenleistung der MCU für eine spätere Auswertung und für den DMA-Controller gewählt. Die Einstellungen für die einzelnen Prescaler werden aus der Application-Note entnommen und entsprechend adaptiert.

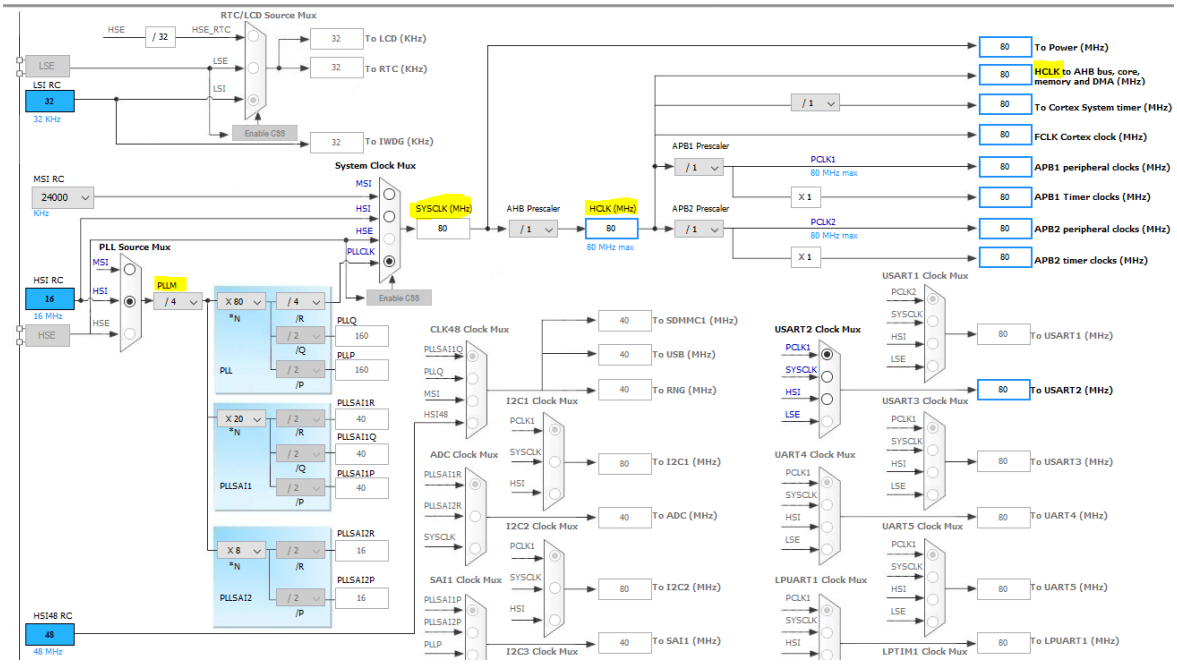


Abbildung 84: STM32L496AGIx Takteinstellungen, Quelle: Eigene Darstellung

Des Weiteren wird das DCMI-Handling via Interrupt, der Pin für den Kameratakt XCLK und der DMA2-Controller entsprechend der Applikation-Note parametrisiert. Im letzten Schritt wird noch der Power-Consumption-Calculator mit den Daten des Kameramoduls und den zu erwarteten Laufzeiten im Run- und Sleep-Modus bespielt, was bei einer 3.6 V / 3400 mAh Batterie eine erste Einschätzung der Laufzeit von 10 Tagen ergibt. Für die Abschätzung wurde eher ein begünstigendes Run zu Sleep Verhältnis von 50:50 gewählt. Ein Stop oder Deep-Sleep Modus wird nicht verwendet, da eine anschließende Re-Initialisierung des System-Takts und des Kameramodules die eingesparte Leistung übertreffen würde.


Abbildung 85: Abschätzung des Energieverbrauchs der Anwendung mit dem STM32CubeMX, Quelle: Eigene Darstellung

Der Programmcode für Keil selbst wird größtenteils von dem für die Auswertung bereits erstellten Code aus dem Kapitel 6.1.3 entnommen. Wesentliche Unterschiede ergeben sich durch die Deaktivierung der System-LEDs, des LCD-Moduls und der Initialisierung aller unbenutzten Schnittstellen als analoge Eingänge. Nach der erfolgreichen Initialisierung des Kameramoduls in QQVGA wird die MCU in den Sleep-Zustand versetzt. Trotz Deaktivierung der HREF- und VSYNC- generierten Interrupts muss aufgrund der regelmäßigen Interrupts der DCMI-Schnittstelle und des DMA-Controllers der Eintrittspunkt für den Sleep-Modus in eine Schleife gesetzt werden. Eine Umstellung auf WFE (Wait for Event), um den Sleep-Modus zu verlassen, ermöglicht auch keine Abhilfe. Alternativ kann an diesem Punkt auch eine HAL_Delay Funktion verwendet werden, jedoch werden wenige mW durch den Sleep-Modus eingespart.

```
do
{   HAL_PWR_EnterSLEEPMode(PWR_LOWPOWERREGULATOR_ON, PWR_SLEEPENTRY_WFI);
    w_cnt++;
} while (w_cnt < 0x100);   //100
```

Abbildung 86: Eintrittspunkt für Sleep-Modus, Quelle: Eigene Darstellung

Die anschließende eigentliche Auswertung des QQVGA-Bildes wird mit einer Schleife umgesetzt, die einmal über das gesamte Bild-Array schaut und die RGB-Information aus den Datenwort im Buffer herauslöst. Falls ein Pixel einen entsprechenden Farbwert aufweist wird ein Zähler inkrementiert, um die Häufigkeit einer Farbe im Bild zu bestimmen.

```
for( int a = 0; a < (160*120); a = a + 1 )
{
  red    = (pBuffer[a]>>3)&0x1f;
  blue   = (pBuffer[a]>>8)&0x1f;
  green  = ((pBuffer[a]&0xE000)>>13)+ ((pBuffer[a]&0x07)<<3);
  if (red > 0x18 && blue < 0x09 && green < 0x09) //we see red
  {              red_val++;     }
  if (red < 0x09 && blue > 0x18 && green < 0x09) //we see blue
  {              blue_val++;    }
  if (red < 0x09 && blue < 0x09 && green > 0x2F) //we see green
  {              green_val++;     }
}
```

Abbildung 87: Aufteilen der Bildinformationen in die RGB-Farbkomponenten, Quelle: Eigene Darstellung

Nach einer Abfrage, ob ein Wert für die Farbhäufigkeit überschritten wird und dies zutrifft, initialisiert das Programm das LCD-Panel und stellt die Kamera auf QVGA um. Anschließend wird ein Schnappschuss getätigt und ein starres Bild für 5 Sekunden am Display angezeigt. Die Programmroutine endet mit der Initialisierung der Ausgangslage.

```
if (avoid_error == 2 ) //RED!!!!
{
     BSP_LCD_Init();
     BSP_CAMERA_Init(CAMERA_R320x240);
     HAL_Delay(500);
     hal_status = HAL_DCMI_Start_DMA(&hDcmiHandler,DCMI_MODE_SNAPSHOT,(uint32_t)pBuffer ,
     (ST7789H2_LCD_PIXEL_WIDTH*ST7789H2_LCD_PIXEL_HEIGHT)/2 );
     OnError_Handler(hal_status != HAL_OK);
     while(start_the_camera_capture == 0) {;}
     hal_status = LCD_Write((uint32_t) (&pBuffer), (uint32_t)&(LCD_ADDR->REG),
     ST7789H2_LCD_PIXEL_WIDTH * ST7789H2_LCD_PIXEL_HEIGHT);
     OnError_Handler(hal_status != HAL_OK);
     HAL_DCMI_Stop(&hDcmiHandler);
     w_cnt = 0;
     HAL_Delay(5000);
     BSP_LCD_Reset();
     BSP_LCD_DeInit();
     BSP_CAMERA_Init(CAMERA_R160x120);
```

Abbildung 88: Anzeige des detektierten Bildes am LCD-Panel, Quelle: Eigene Darstellung

## 7.4 Energieverbrauch

Wie in Kapitel 7.2 beschrieben, wird der Energieverbrauch der Beispielanwendung über eine Vielzahl an Methoden reduziert. So wird, wie in Abbildung 89 ersichtlich, das Kameramodul auf eine Bildrate von ~1,9 fps bei QQVGA beschränkt. Dadurch ergibt sich ein über am Shunt gemessener und rückgerechneter Energieverbrauch für das Kameramodul von ~ 36 mW. Der Energieverbrauch des Kameramoduls nach Detektion eines Objektes und bei Darstellung des QVGA-Bildes am LCD Display erhöhte sich dabei durch Re-Initialisierung des Kameramoduls kurzfristig auf ~ 60 mW.

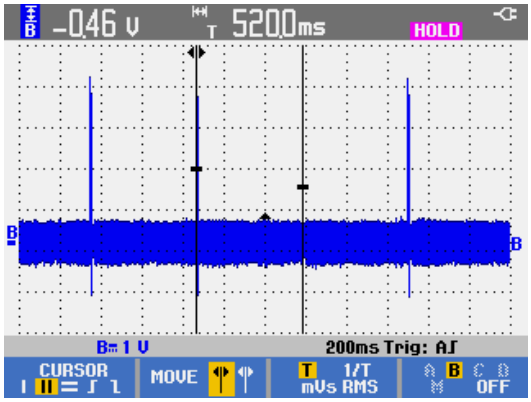


Abbildung 89: Bildrate Beispielapplikation, Quelle: Eigene Darstellung

Der Energieverbrauch der MCU wird mittels Shunt an der IDD-Schnittstelle gemessen, die Laufzeit der Anwendung wird dabei über Keil via Debugging ausgelesen und ergibt eine Auswertung des Bildmaterials mit ~ 2,5 Hz. Der gemessene MCU-Energieverbrauch für die QQVGA-Auswertung ist ~ 29 mW und bei Darstellung des QVGA Bildes ~ 39 mW. In Summe liegt der Energieverbrauch der Anwendung somit zwischen 65 mW und 99 mW, dies entspricht umgerechnet, mit der im STM32CubeMX verwendeten Batterie von 3,6 V / 3400 mAh, einer durchgehenden Laufzeit von ~ 5,2 bis 7,9 Tagen.

Tabelle 10: Messwerte der Beispielapplikation

| **Messgröße** | **Messwert** |
|---|---|
| Shunt | 1,9 Ohm |
| Spannungsversorgung Kamera | 3,25 V |
| Spannungsversorgung MCU | 3,2 V |
| Spannungsabfall Kamera QQVGA - Shunt | 0,021 V |
| Spannungsabfall Kamera QVGA - Shunt | 0,035 V |
| Spannungsabfall MCU QQVGA Auswertung - Shunt | 0,017 V |
| Spannungsabfall MCU QVGA Darstellung - Shunt | 0,023 V |

# 8 ERGEBNISSE UND AUSBLICK

## 8.1 Bewertung der Sensoren

Der 2-MP-Sensor OV2640 wird in zwei verschiedenen Varianten getestet, einmal als integrierter Sensor auf einer Platine mit Objektiv (Variante 2) und einmal als kompaktes Kameramodul mit fester Brennweite (Variante 1). Obwohl beide Module den gleichen Sensortyp haben, werden wesentliche Unterschiede sowohl im Bild als auch beim Energieverbrauch festgestellt. So kann einerseits festgestellt werden, dass die Verwendung von unterschiedlichen Objektiven das Bild bezüglich Schärfe und Helligkeit aufgrund des unterschiedlichen Lichteinfalls maßgeblich beeinflussen. Des Weiteren hängt der Energieverbrauch des Kameramoduls nicht nur vom Sensor, sondern auch von der peripheren Beschaltung ab, die in beiden Fällen unterschiedlich ausfällt. Im Allgemeinen kann sich der Sensor OV2640 aufgrund seiner Bildqualität im Vergleich zum OV9655 und dem OV7670 deutlich hervorheben. Der Sensortyp OV7670 hat sich in der Auswertung als besonders energiesparend gezeigt, was wahrscheinlich auf seine niedrige native Auflösung von 0,3 MP zurückzuführen ist, jedoch werden Abstriche bei der Bildqualität in Kauf genommen. Besonders bei schlechten Lichtverhältnissen wird zudem ein Rotschleier über dem Bild bemerkbar, welcher mit Hilfe der Kamera-Konfigurationsregister nur teilweise behoben werden kann. In Bezug auf ULP wäre der Sensor OV7670 die vorrangige Wahl, jedoch eignet er sich aufgrund des schlechten Bildes eher für Kanten und Schrifterkennung als für Applikationen mit Farbauswertungen. Der Sensor OV9655 kann als Kompromiss, bezogen auf die Bildqualität und den Energieverbrauch, zwischen dem OV2640 und dem OV7670 gesehen werden. Sein starrer Linsenaufbau ermöglicht eine gute Erkennung von Farben und Objekten, für eine Schrift und Barcodeauswertung wäre die starre Linse des Kameramoduls jedoch ungeeignet.

Neben den offensichtlichen Unterschieden im Bild ergeben sich auch Unterschiede bei der Störungsanfälligkeit der Module. Neben der allgemeinen Störungsabhängigkeit von der Höhe der Bildrate werden besonders beim Sensor OV9655 vermehrt Bildartefakte, bzw. Störungen auf der parallelen Datenschnittstelle festgestellt. Bei vermehrten Auftreten der Störungen wird sogar der Mikrocontroller aus der Programmroutine geworfen, jedoch liegt dies an Störungen in den Taktleitungen, welche zu einem falschen Interrupt-Handling und Asynchronität im Bildaufbau führen. Die Störungen sind mit großer Wahrscheinlicht nicht auf die Module selbst zurückzuführen, sondern auf den Aufbau der Applikation, bei der zu lange und ungeschirmte Leitungen verwendet werden.

Die Mikrocontroller-Integration der Sensoren ist dank genormtem Camera-Port-Interface und der Beschränkung auf Sensoren von OmniVision größtenteils einheitlich erfolgt. Zu beachten ist jedoch, dass jedes Modul sein eigenes Kameraregister, seine eigenen Einstellungen und eine unterschiedliche Ansteuerungslogik aufweist und dies im Vorhinein dem Datenblatt entnommen und dementsprechend konfiguriert werden muss.

## 8.2 Bewertung der Beispielapplikation

Die Beispielapplikation, die einen roten Ball aufgrund der Farbwertänderung im Bildframe erkennt, wird im Sinne von Low-Power umgesetzt. Dabei erweist es sich als besonders schwierig, den Cortex-M4 Mikrocontroller STM32L496AGI6 in energiesparende Modi zu setzten, ohne die Ansteuerung des Kameramoduls zu unterbrechen. Es kann zwar einerseits der Energieverbrauch durch Deaktivierung der LEDs und durch Setzen der unbenutzten I/Os auf analog gesenkt werden, jedoch ist es nur eingeschränkt möglich, den Betriebsmodus der MCU zu ändern. So ist es nicht zielführend, die energiesparenderen Modi wie STOP, STANDY oder SHUTDOWN zu verwenden, da diese zu einer Deaktivierung der XCLK, DCMI geführt hätten und somit eine Re-Initialisierung der Kamera nötig ist. STANDBY oder SHUTDOWN als Modi würden des Weiteren den SRAM leeren, womit neben der Kamera auch der Microcontroller neu initialisiert hätte werden müssen. Eine regelmäßige Deaktivierung und Re-Initialisierung der Komponenten ist bei Abänderung der Zeit für die Bildauswertung der Beispielapplikation von ~ 2 Hz auf eine niedrigere Auswerterate von z.B. 0,1 Hz jedoch durchaus ratsam. Aus oben genanntem Grund wird nur der Low-Power-Regulator und der Modus SLEEP verwendet, um den Microcontroller in regelmäßigen Abständen in einen energiesparenden Modus zu versetzen. Die Verwendung des DCMI mit einem DMA-Controller führt jedoch dazu, dass in regelmäßigen Abständen Interrupts/Events generiert werden, die für das DMA-DCMI-Handling benötigt werden. Dies hat zur Folge, dass der SLEEP-Modus, bei Konfiguration als WFI oder WFE, regelmäßig unterbrochen wird und im Programmcode der SLEEP-Modus wiederholt initialisiert werden muss.

Die aus den Messungen und Bildauswertungen gewonnen Kenntnisse, wie die starke Abhängigkeit des Energieverbrauchs von der Framerate des Kameramoduls und die Bildqualität der einzelnen Sensoren sind für die Beispielapplikation bedacht worden. Die Reduzierung der Bildrate des Kameramoduls auf ~1.9 fps in QQVGA hat den Sensor OV9655 an seine minimale Leistungsaufnahme gebracht. Diese hätte zwar durch die Verwendung eines anderen Kamerasensors wie dem OV7670 noch weiter reduziert werden können, jedoch hat sich dieser nicht optimal für die Beispielapplikation geeignet. Eine weitere Limitierung hat sich durch die Auswahl des Microcontrollers, der nur eine 8-Bit-DMCI Schnittstelle und nur einen DMA-Controller und einen Buffer zum DCMI zur Verfügung stellt, ergeben. Durch diese Limitierung können nur Auflösungen mit maximal 480x272 Pixel, bzw. ~262 kB pro Transfer in RGB565 und 720x364 in YUV übertragen werden. Somit ist es nicht möglich Bilder des 1.3 MP (1280x1024) Sensor OV9655 mit seiner nativen Auflösung, nach Detektion des roten Balls, zu machen, zudem ist der SRAM des STM32L496AGI6 mit nur 320 kB nicht ausreichend groß für diese Auflösungen.

Bei der Auswertung und Detektion des roten Balls im QQVGA-Bild wird eine sehr starke Abhängigkeit von den Lichtverhältnissen festgestellt, so muss bei Änderung des Lichts die Detektionsschwelle neu angepasst werden. Nach Anpassung der Detektionsschwelle funktioniert die einfache Auswertung und die anschließende Erstellung des Schnappschusses jedoch sehr zuverlässig.

## 8.3 Zusammenfassung

Trotz der genormten Schnittstelle wurde mehr Zeit als erwartet in die Treiberkonfiguration und Anpassung der Module gesteckt, insbesondere um die Sensoren die Kamerabilder intern auf vordefinierte Auflösungen skalieren zu lassen. Viel Energie ist auch in den Sensor OV2640 investiert worden, der entgegen seiner Registereinstellungen die Logik der Synchronisationstakte negiert weitergegeben hat, und auch in den Sensor OV7670, der seine Word-Bildinformation byte-vertauscht an den Mikrocontroller verschickt hat. Eine Einbindung, Ansteuerung und Analyse von Kameramodulen sollte aufgrund der hohen Taktraten und vielen Abhängigkeiten deswegen mithilfe eines Logic-Analyzers oder eines Oszilloskops durchgeführt werden. Ein weiterer wichtiger Punkt ist die maximale Auflösung der Kamerabilder, die nicht nur durch den Kamerasensor selbst sondern auch durch die Übertragungsschnittstelle und die Ressourcen des Microcontrollers bestimmt wird. Neben der hohen Störungsanfälligkeit der Übertragung bei hohen Bildraten ist auch die Verwendung von kurzen Leitungen ratsam, da bei einem Systemtakt des Kameramoduls von ~ 20 Mhz eine Störungsanfälligkeit nicht vermieden werden kann.

Für die Auswertung der Kamerabilder am Microcontroller selbst ist es zu empfehlen, die Daten auf einem ausreichend großen Display darzustellen oder auf den PC zu übertragen und dort auszuwerten. In der Arbeit werden beide Möglichkeiten verwendet, indem einerseits das Kamerabild skaliert auf das 240x240 Pixel-Display ausgegeben und andererseits die Bilddaten als SRAM-Dump auf den PC übertragen wird, und dort nach Konvertierung in ein gängiges Bildformat umgewandelt und dargestellt wird. Der Grund für die Verwendung beider Methoden ist die unzureichende Auflösung des Displays am Mikrocontroller, die keine genaue Betrachtung und Auswertung zu lies und aus diesem Grund nur als Bildvorschau Verwendung findet. Die Auswertung der Bilder hat wie erwartet Unterschiede zwischen der Bildqualität und der Lichtabhängigkeit der Sensormodule bei gleicher Auflösung ergeben. Dies kann einerseits auf Unterschiede im Aufbau der Sensoren und dessen interne Verarbeitung zurückgeführt werden, andererseits jedoch auch auf die gewählten Kameraeinstellungen, die leicht zwischen den Modulen variieren. Überraschend sind jedoch die merklichen Bildunterschiede zwischen OV2640 Variante 1 und Variante 2, die sich nur am Ansteuerungsmodul, der Linse und am Systemtakt unterschieden haben. Der Grund für diese Abweichung zwischen den Bildern bei gleichen Sensortypen ist die unterschiedliche Belichtungszeit aufgrund des unterschiedlichen XCLK und der Unterschied der Linse bzw. Blende der Module, die einen unterschiedlichen Lichteinfall auf den CMOS-Sensor verursacht haben.

Der Energieverbrauch der Kameramodule selbst ist primär von der Bildrate des Sensormodules abhängig. So haben alle Module gemeinsam, dass durch Reduzierung der Bildrate der Energieverbrauch halbiert werden hat können. Des Weiteren kann festgestellt werden, dass der Sensor mit der höchsten nativen Auflösung von 2 MP am meisten und der Sensor mit der geringsten Auflösung von 0,3 MP am wenigsten Energie verbraucht. Rückschließend, natürlich abhängig vom Sensortyp, wäre deswegen der Sensor mit der geringst nötigen Auflösung für die Applikation bevorzugt zu wählen, um den Energieverbrauch so niedrig wie möglich zu halten. Eine Abhängigkeit des Energieverbrauchs der Module von der ausgegebenen Auflösung wie QVGA oder VGA ist bemerkbar, jedoch hat sich kein wesentlicher Mehrverbrauch bei einer höheren Auflösung ergeben. Der vermutete Grund dafür ist der

erhöhte Energieverbrauch des Sensormoduls selbst, welcher für die Skalierung von einer höheren in eine niedrigere Auflösung verbraucht wird.

Das Microcontrollerboard STM32l496g-Discovery ist aufgrund seines LCDs, der DCMI-Schnittstelle, des geringen Energieverbrauchs und wegen des DMA-Controllers für diese Arbeit ausgewählt worden. Die Limitierung auf eine 8-Bit-CPI und die Datenrate über das DMA-Interface haben sich aber als Flaschenhals für größere Auflösungen als QVGA herausgestellt, so konnte eine VGA-Auflösung nur durch eine Zwischenlösung umgesetzt werden.

Die für die Arbeit erstellte Beispielapplikation, welche ein Kamerabild zweimal pro Sekunde nach einem roten Objekt auswertet, kann durch Aktivierung mehrerer energiesparender Maßnahmen eine abgeschätzte durchgehende Laufzeit von ~ 5,2 bis 7,9 Tagen mit einer einzelnen 3,6 V / 3400 mAh Batterie erreichen. Eine Verlängerung der schon erreichten Laufzeit ist durch die Auswahl eines moderneren Kamerasensors, weitere Optimierungen des Programmcodes, der Energie-Modi Low-Power-Run und Sleep durchaus gegeben. Besonders bei Abänderung der Auswertungsfrequenz der Bildanalyse z.B. auf eine Bildauswertung einmal alle 10 Sekunden würde die Applikation genügend Energie einsparen, um die Laufzeit um mehrere Tage zu verlängern.

Das persönliche Resümee zu dieser Arbeit ist sehr positiv, da das Hauptziel der Auswertung und Analyse von verschiedenen Kameramodulen erfolgreich umgesetzt worden ist. Die erzwungene Verwendung von älteren Kameramodulen aufgrund des nicht öffentlichen neueren MIPI-Standards, und die miteinhergehende Beschränkung der Datenrate zwischen dem Mikrocontroller und dem Kameramodul hat das Ergebnis nur dahingehend getrübt, dass bei Verwendung neuerer Module ein höheres Energie-Einsparungspotential und eine bessere Bildqualität zu erwarten gewesen wäre.

## 8.4 Ausblick

Die vermehrte Auswertung und Analyse von Kamerabildern ist ein Trend, der nicht nur am Verbrauchermarkt, zum Beispiel als Identifikation via Gesichtserkennung am Smartphone, immer stärker Verwendung findet, sondern ist auch ein Thema, das in der Industrie immer mehr an Bedeutung gewinnt. Dabei werden sehr oft leistungsstarke Prozessoren bemüht, mithilfe von komplexen Algorithmen, hochauflösende Bilder auszuwerten, und dabei wird der Aspekt einer Leistungsoptimierung sehr oft als zweitrangig behandelt. Das im Zuge dieser Arbeit aufgezeigte große Einsparungspotential bei dem Energieverbrauch von Kameramodulen durch eine intelligente Ansteuerung, hilft hierbei Anwendungen unabhängig von einer festen Energieversorgung umzusetzen und Applikationen als mobile Auswerteeinheiten zu realisieren. Die miteinhergehenden Kompromisse bei einer energiesparenden Ansteuerung von niedriger Bildrate und Auflösung können durch eine intelligente Programmierung und Aufteilung der Ressourcen größtenteils umgangen werden. Ein weiterer wichtiger Aspekt ist die Auswahl der passenden Linse, Blende und die Möglichkeit eines verstellbaren Brennpunktes für den Bildsensor, da sich die Auswahl als wesentlich für den Erfolg einer Bildauswertung herausgestellt hat. Lichteinstrahlungen auf das Objekt der Auswertung sollten daneben konstant gehalten werden, um eine anschließende Auswertung so einfach wie möglich zu gestalten.

Weiterführend wäre im Zusammenhang mit der Arbeit auch die Analyse von unterschiedlichen Bildauswertungsalgorithmen bezüglich ihres Energieverbrauchs und der Effizienz auf der MCU sehr interessant. Besonders im Bereich der Objekt- und Schrifterkennung wäre die Auswertung von Algorithmen zur Kantendetektion für eine industrielle Anwendung zu empfehlen, um fortsetzend an diese Arbeit anzuknüpfen.

# LITERATURVERZEICHNIS

**Gedruckte Werke (4)**

Dudenredaktion (Hrsg.) (2009): *Duden Band 1. Die deutsche Rechtschreibung: Das umfassende Standardwerk auf der Grundlage der neuen amtlichen Regeln*, 25. Auflage, Dudenverlag, Mannheim.

Durini, Daniel; Durini, D.; Arutinov, D.; Lesser, M.; Choubey, B.; Lahav, A.; Mughal, W.; Gouveia, L; Ginhac, D.; Gove, R.; Fenigstein, A; Strum, A; Nikzad, S.; Centen, P.; Turchetta, R.; Bogaerts, J.; De Locht, C.; Van den Bröck, H.; Henderson, R.; Rae, B.R.; Li, D.-U.; Fenigstein, A.; Strum, A. (2014): *High Performance Silicon Imaging*, 1 Auflage, Woodhead Publishing, Printed and bound in the United Kingdom.

Erlenkötter, Helmut (1990): *C Programmieren von Anfang an*, 23 Auflage, rororo, Hamburg.

Westphalen, Christian (2016): *Die große Fotoschule*, 3 Auflage, Rheinwerk Fotografie, Bonn.

**Online-Quellen (25)**

QIMAGING (2014): *https://www.qimaging.com*
https://www.qimaging.com/ccdorscmos/pdfs/RollingvsGlobalShutter.pdf [Stand: 14.08.2017].

Foto-Mosaik-Edda (2017): *http://www.fmedda.com*
http://www.fmedda.com/de/article/dpi_ppi [Stand: 15.08.2017].

Universität Münster*https://www.uni-muenster.de*
https://www.uni-muenster.de/ZIV/Lehre/MM_HWK/V001S02.htm [Stand: 15.08.2017].

TU Clausthal (2010): *http://techwww.in.tu-clausthal.de/*
http://techwww.in.tu-clausthal.de/ [Stand: 15.08.2017].

FTDI (2015): *http://www.ftdichip.com*
http://www.ftdichip.com/Support/Documents/TechnicalNotes/TN_158_What_Is_The_Camera_Parallel_Interface.pdf [Stand: 20.08.2017].

Microchip Technology Inc. (2009): *http://www.microchip.com/*
http://ww1.microchip.com/downloads/en/DeviceDoc/01146B_chapter%202.pdf [Stand: 26.08.2017].

Texas Instruments (2015): *http://www.ti.com*
http://www.ti.com/lit/an/slva704/slva704.pdf [Stand: 02.09.2018].

Simply Embedded (2017): *http://www.simplyembedded.org*
http://www.simplyembedded.org/tutorials/msp430-uart/ [Stand: 2017.09.02].

Olympusmicro (2012): *http://www.olympusmicro.com*
http://www.olympusmicro.com/primer/digitalimaging/cmosimagesensors.html [Stand: 03.09.2017].

Open MV (2017): *https://openmv.io/*
https://openmv.io/ [Stand: 11.09.2017].

STMicroelectronics (2017): *http://www.st.com*
http://www.st.com/content/ccc/resource/technical/document/data_brief/group1/8b/5b/a6/24/98/8b/49/2f/DM00353124/files/DM00353124.pdf/jcr:content/translations/en.DM00353124.pdf [Stand: 12.09.2017].

Arm Limited (2011): *http://www.keil.com*
http://www.keil.com/support/docs/1584/ [Stand: 19.10.2017].

STMicroelectronics (2017): *http://www.st.com*
http://www.st.com/content/st_com/en/products/embedded-software/mcus-embedded-software/stm32-embedded-software/stm32cube-embedded-software/stm32cubel4.html [Stand: 29.10.2017].

Stemmer Imaging (2018): *https://www.stemmer-imaging.de*
https://www.stemmer-imaging.de/media/uploads/cameras/avt/de/de-Allied-Vision-WhitePaper-CCD-vs-CMOS-0416-KAVTO115-201604.pdf [Stand: 13.1.2018].

Baker, R. (2010): *https://www.u-cursos.cl*
https://www.u-cursos.cl/usuario/9553d43f5ccbf1cca06cc02562b4005e/mi_blog/r/CMOS_Circuit_Design__Layout__and_Simulation__3rd_Edition.pdf [Stand: 14.08.2017].

Christensson, Per (2007): *https://techterms.com*
https://techterms.com/definition/pmu [Stand: 13.09.2017].

Dönicke, Thomas (2009): *https://tu-dresden.de*
https://tu-dresden.de/ing/informatik/ti/vlsi/ressourcen/dateien/dateien_studium/dateien_lehstuhlseminar/vortraege_lehrstuhlseminar/hs_ss09/AHB-Bus.pdf?lang=de [Stand: 13.09.2017].

Krottmaier, Harald (2011): *FH CAMPUS 02*
http://www.campus02.at [Stand: 17.05.2011].

Lesser, Michael; Fossum, Eric R.; AY, Suat U. (2002): *http://www.ee.uidaho.edu*
http://www.ee.uidaho.edu/ee/analog/suatay/papers/CMOS_APS_4836_41.pdf [Stand: 14.8.2017].

Metz, Bernhard (2014): *http://www6.in.tum.de*
http://www6.in.tum.de/pub/Main/TeachingWs2014ProseminarMicrocontrollerEmbedded/Was_ist_ein_Microcontroller.pdf [Stand: 27.08.2017].

Mogensen, Torben (2000 – 2010): *http://www.diku.dk*
http://www.diku.dk/~torbenm/Basics/basics_lulu2.pdf [Stand: 25.087.2017].

Neges, Matthias (2003): *http://www.diss.fu-berlin.de*
http://www.diss.fu-berlin.de/diss/servlets/MCRFileNodeServlet/FUDISS_derivate_000000001443/00_Neges.pdf?hosts= [Stand: 06.08.2017].

Neuhaus, Sven (2004-2017): *www.sven.de*
https://www.sven.de/dpi/ [Stand: 15.08.2017].

Ponikvar, Dusan (2014): *https://www.fmf.uni-lj.si* https://www.fmf.uni-lj.si/~ponikvar/STM32F407%20project/Ch4%20-%20The%20use%20of%20ports.pdf [Stand: 1.09.2017].

Schröder, Wolfgang (2016): *www.cpp-tutor.d* http://www.cpp-tutor.de/cpp/le08/include.html [Stand: 25.08.2017].

**Normen (7)**

STMicroelectronics (Hrsg.) (2017): *AN5020: Digital camera interface for STM32 MCUs*

OmniVision (Hrsg.) (2006): *OV2640: OV2640*

OmniVision (Hrsg.) (2005): *OV7670: www.ovt.com*

OmniVision (Hrsg.) (2004): *OV9650: OV9650*

OmniVision (Hrsg.) (2006): *OV9655: OV9655/OV9155 CMOS SXGA*

(2013): *OVM7692: http://www.ovt.com*

STMicroelectronics (Hrsg.) (2017): *UM2160 User manual: UM2160 User manual*

# ABBILDUNGSVERZEICHNIS

Abbildung 1: Innerer photoelektrischer Effekt (leicht modifiziert); Quelle: https://photovoltaiksolarstrom.com/photovoltaiklexikon/leitungsband-und-valenzband/ [Datum 06.08.2017] ..... 3

Abbildung 2: Äußerer photoelektrischer Effekt (leicht modifiziert); Quelle: http://www.gironimo.org/wissenschaft/photoelektrische-effekt.html [Datum 06.08.2017] ..... 3

Abbildung 3: (a) full frame CCD,(b) frame transfer CCD, (c) interline transfer CCD, (d) orthogonal transfer CCD.; Quelle: Lesser (2014) Kapitel 3 [Datum 06.08.2017] ..... 5

Abbildung 4: CMOS-Inverter Schaltlogik (leicht modifiziert); Quelle: https://upload.wikimedia.org/wikipedia/commons/5/51/CMOS_inverter_model_E.PNG [Datum 06.08.2017] ..... 6

Abbildung 5: Active Pixel Sensor (leicht modifiziert); Quelle: http://www.ee.uidaho.edu/ee/analog/suatay/papers/CMOS_APS_4836_41.pdf [Datum 06.08.2017] ..... 6

Abbildung 6: (a) Frontseitenbeleuchteter CMOS-Sensor (b) Rückseitenbeleuchteter CMOS-Sensor; Quelle: Lahav/Fenigstein/Strum (2014) Kapitel 4 ..... 7

Abbildung 7: Links Global Shutter, Rechts Rolling Shutter; Quelle http://lfa.mobivap.uva.es/~fradelg/phd/figures/global-shutter.png [Datum 06.08.2017] ..... 8

Abbildung 8: Veränderung der PPI bei gleichbleibender Bildgröße; Quelle: http://www.fmedda.com/de/article/dpi_ppi [Datum 06.08.2017] ..... 9

Abbildung 9: Bayer-Muster; Quelle: https://www.uni-muenster.de/ZIV/Lehre/MM_HWK/V001S02.htm [Datum 06.08.2017] ..... 10

Abbildung 10: Bayer-Muster Rekonstruktion; Quelle: http://www.olympusmicro.com/primer/digitalimaging/cmosimagesensors.html [Datum 03.09.2017] ..... 10

Abbildung 11:CPI mit RGB565 oder RGB555; Quelle: FTDI (2015), Online-Quelle [20.08.2017] ..... 11

Abbildung 12: Vereinfachter Aufbau einer Kamera; Quelle: http://maurer-tech.com/Debian/v4l/DieKamera.jpg [03.09.2017] ..... 12

Abbildung 13: OmniVision OVM7692 Funktionsblockdiagramm; Quelle: OVM7692 (2013) ..... 13

Abbildung 14: Aufbau eines ARM Cortex M3/4; Quelle: https://www.arm.com/assets/images/M3_4_system.png [Datum 27.08.2017] ..... 16

Abbildung 15: STM32F40x GPIO Aufbau; Quelle: https://www.fmf.uni-lj.si/~ponikvar/STM32F407%20project/Ch4%20-%20The%20use%20of%20ports.pdf, Seite 1 [Datum 01.09.2017] ..... 17

Abbildung 16: I2C Datenübertragung mit Abbruch; Quelle: http://www.ti.com/lit/an/slva704/slva704.pdf, Seite 6 [Datum 02.09.2017] ..... 18

Abbildung 17: UART Übertragung (Modifiziert); Quelle: https://learningmsp430.files.wordpress.com/2014/01/w_example.png [Datum 02.09.2017] ..... 19

Abbildung 18: Vom Quellcode zum Programm; Quelle: Erlenkötter (1990) S.13 ..... 21

Abbildung 19: Taktraten des Bildsignals; Quelle: FTDI (2015), Online-Quelle [20.08.2017] ..... 24

Abbildung 20: Arduino Nano; Quelle: http://www.etechpk.net/wp-content/uploads/2016/02/ARDUINO_NANO_03.png [Datum 11.09.2017] ..... 25

Abbildung 21: OpenMV Kameramodul; Quelle: https://cdn.sparkfun.com//assets/parts/1/2/1/1/7/14186-01a.jpg [Datum 11.09.2017] ..... 25

Abbildung 22: CPI Kameramodul; Quelle: http://www.waveshare.com/img/devkit/accBoard/OV9655-Camera-Board/OV9655-Camera-Board-2.jpg [Datum 11.09.2017] ..... 26

Abbildung 23: UART Kameramodul; Quelle: http://www.si-cube.com/UpFiles/Article/201106/201162155413577.jpg [Datum 11.09.2017] ..... 26

Abbildung 24: Composite Video Kameramodul; Quelle: https://cdn.sparkfun.com//assets/parts/1/9/9/2/08773-03-L.jpg [Datum 11.09.2017] ..... 26

Abbildung 25: MIPI Kameramodul; Quelle: https://www.e-consystems.com/images/denebola/Denebola-CX3-RDK-camera-Module.jpg [Datum 11.09.2017] ..... 26

Abbildung 26: IR-Filter über dem CMOS Sensor; Quelle: http://wiki.raspberrytorte.com/index.php?title=File:RemoveLensFromRPiCamModule.png [Datum 12.09.2017] ..... 27

Abbildung 27: Vorderseite 32L496G-DISCOVERY (Modifiziert); Quelle: http://www.mikrozone.sk/obrazky/newspost_images/12.27.19.jpg [Datum 12.09.2017] ..... 28

Abbildung 28: 32L496G-DISCOVERY Übersicht Vorderseite; Quelle: UM2160 User manual (2017) S.11 ..... 29

Abbildung 29: 32L496G-DISCOVERY Übersicht Rückseite; Quelle: UM2160 User manual (2017) S.12 29

Abbildung 30: DCMI FIFO Ablage; Quelle: STM32L4DCMI S.5 ..... 30

Abbildung 31: CN1 Anschluss Vorderseite; Quelle: UM2160 User manual (2017) S.30 ..... 31

Abbildung 32: Schnittstellenbeschreibung CN1; Quelle: UM2160 User manual (2017) S.30 ..... 31

Abbildung 33: DMA; Quelle: STM32L4DMA S.2 ..... 32

Abbildung 34: Blockschaltbild OV9655; Quelle: OV9655 (2006) ..... 33

Abbildung 35: Blockschaltbild OV2640; Quelle: OV2640 (2006) ..... 34

Abbildung 36: Blockschaltbild OV5640; Quelle: OV7670 (2005) ..... 35

Abbildung 37: Discovery Board verbunden mit einem FPC- zu DIP-Adapter, Quelle: Eigene Darstellung. ..... 36

Abbildung 38: FPC- zu DIP-Adapter verbunden mit dem OV9655 Kameramodul, Quelle: Eigene Darstellung. ..... 36

Abbildung 39: Schematische Darstellung der Kameramodulintegration - Allgemein, Quelle: Eigene Darstellung. ..... 37

Abbildung 40: Schematische Darstellung der Kameramodulintegration – OV2640 Variante 2, Quelle: Eigene Darstellung. ..... 38

Abbildung 41: Projektstruktur, Quelle: Eigene Darstellung. ..... 40

Abbildung 42: Sensorauswahl, Quelle: Eigene Darstellung. ..... 40

Abbildung 43: Zerschneiden eines VGA Bildes zu QVGA, Quelle: Eigene Darstellung. ..... 41

Abbildung 44: Initialisierung des OV2640, Quelle: Eigene Darstellung. ..... 42

Abbildung 45: YUV SXGA Registereinstellungen für den OV9655, Quelle: Eigene Darstellung. ..... 42

Abbildung 46: Beschneidung des QVGA Bildes auf 240x240 für das LCD-Display, Quelle: Eigene Darstellung. ..... 43

Abbildung 47: Konfiguration des DCMI nach Logik des HREF und VSYNC Signals, Quelle: Eigene Darstellung. ..... 43

Abbildung 48: Aufnahme von HREF (Blau) und VREF(Rot) für ein Frame, Quelle: Eigene Darstellung. . 44

Abbildung 49: Aufnahme von HREF (Blau) und VREF(Rot) im Detail, Quelle: Eigene Darstellung. ..... 44

Abbildung 50: XCLK, Quelle: Eigene Darstellung. ..... 44

Abbildung 51: Bestimmung der Framerate mithilfe eines Oszilloskops, Quelle: Eigene Darstellung. ..... 44

Abbildung 52: Display am µC Board mit Kamerabild, Quelle: Eigene Darstellung. ..... 45

Abbildung 53: Test Pattern eines Kameramoduls, Quelle: Eigene Darstellung. ..... 45

Abbildung 54: Erstellen eines Speicherabbildes in Keil µVision, Quelle: Eigene Darstellung. ..... 46

Abbildung 55: Auszug des Speicherabbildes von einem Kamerabild, Quelle: Eigene Darstellung. ..... 46

Abbildung 56: C# Auswertesoftware mit Bildvorschau, Quelle: Eigene Darstellung. ..... 47

Abbildung 57: Bytevertauschtes Bild, Quelle: Eigene Darstellung. ..... 47

Abbildung 58: Zusammenhang zwischen RGB565 und RGB888, Quelle: http://alfredoer.com/wp-content/uploads/2015/01/RGB888convertToRGB565.png ..... 48

Abbildung 59: RGB565 zu RGB888, Quelle: Eigene Darstellung. ..... 48

Abbildung 60: Übernommener und adaptierter Code zum Erstellen eines Bildes Pixel für Pixel, Quelle: Eigene Darstellung. ..... 48

Abbildung 61: OV9655, Quelle: https://www.waveshare.com/img/devkit/accBoard/OV9655-Camera-Board/OV9655-Camera-Board-3.jpg ..... 49

Abbildung 62: OV2640 Variante1, Quelle: http://www.arducam.com/wp-content/uploads/2012/11/OV2640_5T.jpg ..... 49

Abbildung 63: OV7670, Quelle: https://core-electronics.com.au/media/catalog/product/cache/1/image/650x650/fe1bcd18654db18f328c2faaaf3c690a/d/e/device10_1000.jpg. ..... 49

Abbildung 64: OV2640 Variante2, Quelle: https://ae01.alicdn.com/kf/HTB1SvuISVXXXXcrXFXXq6xXFXXXi/OV2640-camera-module-Module-2-million-pixel-electronic-integrated-with-jpeg-compression-new-big-promotion.jpg_640x640.jpg ..... 49

Abbildung 65: Energieverbrauch des OV2640 Variante 1, Quelle: Eigene Darstellung. ..... 50

Abbildung 66: Energieverbrauch des OV2640 Variante 2, Quelle: Eigene Darstellung. ..... 51

Abbildung 67: Energieverbrauch des OV9655, Quelle: Eigene Darstellung. ..... 52

Abbildung 68: Energieverbrauch des OV7670, Quelle: Eigene Darstellung. ..... 53

Abbildung 69: QQVGA OV2640 Variante2, Quelle: Eigene Darstellung ..... 54

Abbildung 70: QQVGA OV7670, Quelle: Eigene Darstellung ..... 54

Abbildung 71: QQVGA OV9655, Quelle: Eigene Darstellung ..... 54

Abbildung 72: QQVGA OV2640 Variante1, Quelle: Eigene Darstellung ..... 54

Abbildung 73: QVGA OV2640 Variante 2, Quelle: Eigene Darstellung ..... 54

Abbildung 74: QVGA OV7670, Quelle: Eigene Darstellung ..... 54

Abbildung 75: QVGA OV9655, Quelle: Eigene Darstellung ..... 54

Abbildung 76: QVGA OV2640 Variante 1, Quelle: Eigene Darstellung ..... 54

Abbildung 77: VGA OV2640 Variante 1, Quelle: Eigene Darstellung ..... 55

Abbildung 78: VGA OV2640 Variante 2, Quelle: Eigene Darstellung ..... 55

Abbildung 79: QVGA OV7670, Quelle: Eigene Darstellung ..... 56

Abbildung 80: VGA OV9655, Quelle: Eigene Darstellung ..... 56

Abbildung 81: QQVGA Bild, die Kamera erkennt ein rotes Objekt, Quelle: Eigene Darstellung ..... 58

Abbildung 82: QVGA-Bild der Kamera skaliert auf 240x240, Quelle: Eigene Darstellung ..... 58

Abbildung 83: STM32CubeMX, Quelle: Eigene Darstellung ..... 60

Abbildung 84: STM32L496AGIx Takteinstellungen, Quelle: Eigene Darstellung ..... 60

Abbildung 85: Abschätzung des Energieverbrauchs der Anwendung mit dem STM32CubeMX, Quelle: Eigene Darstellung ..... 61

Abbildung 86: Eintrittspunkt für Sleep-Modus, Quelle: Eigene Darstellung ..... 61

Abbildung 87: Aufteilen der Bildinformationen in die RGB-Farbkomponenten, Quelle: Eigene Darstellung ..... 62

Abbildung 88: Anzeige des detektierten Bildes am LCD-Panel, Quelle: Eigene Darstellung ..... 62

Abbildung 89: Bildrate Beispielapplikation, Quelle: Eigene Darstellung ..... 63

# ABKÜRZUNGSVERZEICHNIS

| | |
|---|---|
| CCD | Charge Coupled Device |
| CMOS | Complementary Metal Oxide Semiconductor |
| MOSFET | Metall-Oxid-Halbleiter-Feldeffekttransistor |
| NMOS | N-type Metal-Oxide Semiconductor |
| PMOS | P-type Metal-Oxide Semiconductor |
| APS | Active Pixel Sensor |
| DPI | Dots per Inch |
| PPI | Pixel per Inch |
| CPI | Camera Parallel Interface |
| SoC | System-on-a-Chip |
| VSYNC | Vertical SYNC |
| HREF | Horizontal Reference |
| PCLK | Pixel Clock |
| XCLK | Kameramodul Grundtakt |
| BCPL | Basic Combined Programming Language |
| MCU | Micro Computer Unit |
| CPU | Central Processing Unit |
| µC | Mikrocontroller |
| ALU | Arithmetic Logic Unit |
| GPIO | General Purpose Input Output |
| IDE | Integrated Development environment |
| WFE | Wait for Event |
| WFI | Wait for Interrupt |
| ULP | Ultra Low Power |
| DCMI | Digital Camera Interface |
| IDE | Integrated Development Environment |