Themenblatt

Problemanalyse

1 Allgemeines:

Um ein Programm, wie einen Simulator für Mikrocontroller schreiben zu können, muss man zuerst einige Vorarbeiten erledigen. Man sollte in keinem Fall einfach wild drauflos programmieren.

Das Wichtigste ist, dass man die Aufgabe richtig versteht. Dazu muss man sich in die Problematik bzw. Thematik der Aufgabe einarbeiten. Sobald man die Aufgabe richtig verstanden hat, kann man dazu übergehen diese Gesamtaufgabe in kleine Module aufzuteilen. Dieses ist eine Problemanalyse, bei der man ein großes Problem in immer kleinere Probleme (Aufgaben) aufteilt, um zum Schluss ein zusammenhängendes Netz an Modulen vor sich zu haben.

In diesem Themenblatt wird diese Vorgehensweise an Hand des Simulatorprojekts dargestellt. Dabei werden nur die ersten Schritte gezeigt, nicht jedoch die vollständige Beschreibung.

WAS ist das Ziel der Aufgabe?
WIE teile ich die Aufgabe in kleine Funktionsgruppen auf?
Welche Probleme treten dabei auf?
WIE kann man diese in einem PAP[1] darstellen?
WELCHE Programmiersprache wird verwendet?

Diese Grundsätze müssen Sie sich zu Beginn und während der Realisierung des Projekts immer wieder stellen!

2 Aufgabe:

Ein Simulator soll die Funktionsweise des Mikrocontrollers PIC 16F84 nachbilden. Dabei steht die Funktionalität und nicht die naturgetreue Nachbildung der Hardware im Vordergrund. Soll z.B. eine Addition durchgeführt werden, muss dafür keine eigene ALU (Arithmetic Logic Unit) programmiert werden, sondern es genügt, die Addition in der Hochsprache durchzuführen. Allerdings darf das, was sonst noch bei dieser Addition im Controller passiert, z.B. das Bedienen der Flags (Zustandsbits), nicht vernachlässigt werden.

Das Alles wird in Form einer Checkliste zusammengestellt. Dabei wird auch hier eine Untergliederung durchgeführt. Erfahrungsgemäß führt eine detaillierte Liste zu einer schnelleren und überschaubaren Realisierung. Wenn man den einzelnen Punkten zusätzlich eine voraussichtliche Bedarfszeit zuordnet, wird auch das Zeitmanagement erfolgreich sein.

1 PAP bedeutet Programmablaufplan z.B. Flussdiagramm oder Nassi-Shneiderman-Diagramm

Obige Grundsätze auf unsere Aufgabe angewendet lauten:

Ziel der Aufgabe ist die Nachbildung eines µC mittels einer App!
Die Funktionsgruppen entsprechen im Groben denen des Originals!
Problem: WAS aus dem PIC kann man WIE im Simulator umsetzen!

2.1 Schritt 1:

2.1.1 Bedienoberfläche (Frontend)

Am Besten man erstellt am Anfang ein Mockup[2] der GUI. Hier werden dann die vom Mikrocontroller unabhängigen Funktionen, wie z.B. Breakpoints, Einzelschritt etc.) sowie die Darstellung der Register und Hilfsregister definiert. Dazu stellt man sich die Frage, wie soll der Simulator bedient werden. Was muss von den Interna dargestellt werden, damit ein vernünftiges Arbeiten mit dem Simulator möglich wird. Das wird in einer (Check-)liste genau erfasst und entsprechend untergliedert.

Ein entsprechendes Themenblatt für die Erstellung einer GUI ist ebenfalls verfügbar.

2.1.2 Backend

Verstehen wie ein Mikrocontroller funktioniert. Dazu muss man in der passenden Dokumentation nachlesen, welche Funktionsgruppen dieser Controller hat und wie diese miteinander interagieren.

Die erste Wahl, sich einen Überblick zu verschaffen, ist das Blockschaltbild. Hier erkennt man die grobe Struktur und die Funktionsblöcke. In Abbildung 1 ist das Original und in Abbildung 2 ein modifiziertes Blockschaltbild des PIC 16F84 zu sehen. Für einen groben Überblick reicht jedoch Abbildung 1.

Alle nachfolgenden Punkte und noch viele weitere mehr kommen in die (Check-)liste.

Man erkennt oben links den Programmspeicher mit einer Größe von 1k x 14. Das Datenblatt gibt an anderer Stelle Auskunft, was 1k x 14 bedeutet. Damit ist klar, dass der Simulator einen Programmspeicher benötigt. Rechts daneben liegt der Program Counter (Programmzähler) und unter diesem der Stack.

Bereits an dieser Stelle kann man sich überlegen, welche Struktur und von welchem Datentyp dieser Programmspeicher sein sollte. Da der Programmzähler die Adresse beinhaltet, unter der der nächste Befehl zu finden ist, stellt er einen Zeiger dar. Die einzelnen Elemente eines Arrays können mittels Zeiger (Index) angesprochen werden. Um den Datentyp zu bestimmen, muss man sich überlegen, in welcher Form die spätere Befehlsbestimmung (Befehlsdekoder) am Besten funktioniert. Die Befehle bestehen aus irgendwelchen Bitkombinationen, sind also einfache Zahlen. Genauso die Daten die diese Befehle verarbeiten sollen. Deshalb ist hier der Datentyp des Integers am Besten geeignet.

Der Programmzähler ist ein Zeiger und damit auch eine Integer Variable.

2 Anschauungsmodell, z.B. realitätsnahe Skizze

Der Stack steht in direkter Verbindung zum Programmzähler. Gemäß dem Pfeil können Werte aus dem Programmzähler in den Stack und umgekehrt geschrieben werden. Im Stack haben maximal 8 Adressen Platz. Somit lässt sich dieser Stack als Integer Array der Größe 8 nachbilden.

Rechts neben Stack und Programmzähler findet man den Datenspeicher. Er ist 68 x 8 groß. Liest man das entsprechende Kapitel im Datenblatt sieht man, dass der adressierbare Bereich des Daten-

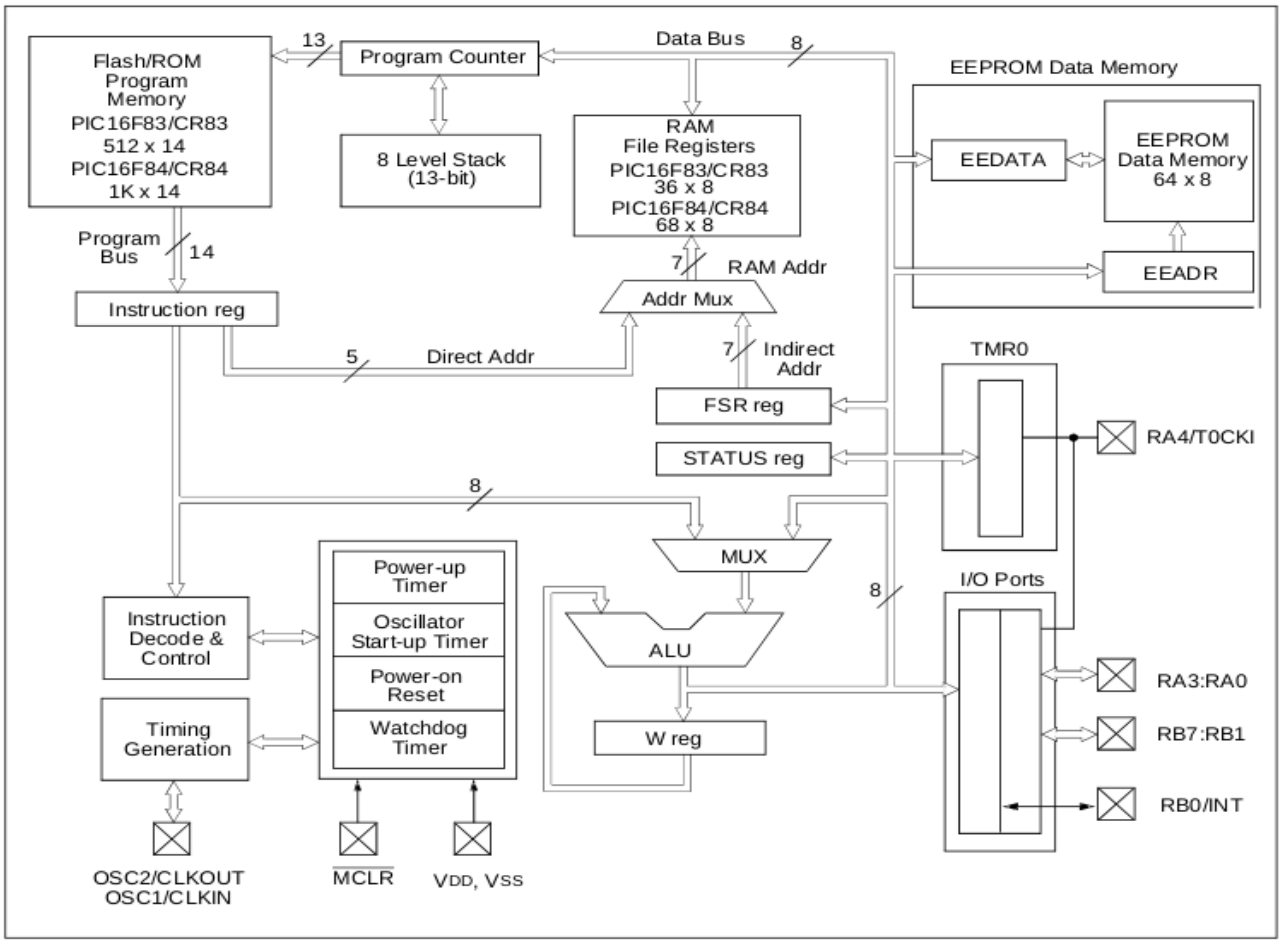


Abbildung 1: Original Blockschaltbild aus dem Datenblatt es 16F84

speichers viel größer als 64 ist. Es sind maximal 2 x 128 Adressen ansprechbar, von denen jedoch nur 64 mit Speicherelementen bestückt sind. Das Datenblatt verrät auch, was bei einem Zugriff auf unbestückte Adressen passiert.

Der Datenspeicher ist in zwei Bereiche unterteilt. Es sind dies die SFR (Special Function Registers) und der GPR-Bereich (General Purpose Registers). Diese Aufteilung ist im Datenblatt durch entsprechende Tabellen sehr gut beschrieben.

Außerdem ist der Datenbereich in zwei Bänke aufgeteilt. Jede dieser Banken besitzt einen adressierbaren Bereich von 128 Adressen. Das Umschalten erfolgt mittels einem speziellen Bit im _____register.

Ein weiterer Datenspeicher ist das EEPROM Data Memory. Es lässt sich wie die anderen Speicher auch als Array von Integer nachbilden. Das Datenblatt gibt Auskunft, dass der Zugriff auf diesen Speicher mittels den Registern _____ und _____ sowie den Steuerregister _____ und _____ erfolgt.

Unterhalb des Programmspeichers ist das Instruction Register angesiedelt. Hier wird jeweils der nächste Befehl zwischengespeichert. Da der Befehl aus dem Befehlsspeicher stammt, ist auch dieses Register vom Typ _____ .

Der Befehlsdekoder ist eine Funktion, die das aktuelle Bitmuster aus dem Befehlsregister untersucht, um herauszufinden, um welchen Befehl es sich handelt. Die dafür notwendige Information findet man im Datenblatt bei der Befehlsübersicht.

Das Timing definiert die Geschwindigkeit mit der die einzelnen Befehle abgearbeitet werden. Eine passende Anzeige für den Bediener wäre sicher von Vorteil.

Power-Up-Timer und Oscillator Start-Up-Timer können in einem Simulator, wie er hier verlangt wird, nicht sinnvoll realisiert werden.

Die Funktion des Watchdogs ist komplex aber im Datenblatt gut beschrieben. Der Watchdog kann nicht per Software ein- und ausgeschaltet werden.

2.2 Schritt 2

Um die ganze Funktionalität dieses einfachen Mikrocontrollers zu verstehen, muss das Datenblatt und die entsprechenden Themenblätter durchgearbeitet werden. Zusätzlich zeigt ein verfügbarer Foliensatz wie der PIC die einzelnen Befehle abarbeitet.

Üblicherweise werden diese Unterlagen während der ganzen Entwicklungszeit benötigt.

Auch die kommentierten Testprogramme helfen die Funktionsweise des PIC Mikrocontrollers besser zu verstehen. Außerdem können sie als Leitfaden benutzt werden, um die Reihenfolge der zu implementierenden Befehle festzulegen. Auf diese Weise bleibt die Übersicht besser gewahrt.

Auch zu diesem Punkt sind viele Einträge in der (Check-)liste notwendig.

2.3 Schritt 3

Überlegen, welche Funktionsgruppen auf der GUI sichtbar sein müssen und welche lediglich im Backend vorkommen. So muss beispielsweise das Datenregister in jedem Fall auf der GUI sichtbar sein. Dabei kann man diese visuelle Komponente i.d.R nur mit einigem Aufwand direkt als Datenspeicher nutzen. Somit ist es besser, die Daten im Hintergrund zu speichern und dann die GUI bei Bedarf aktualisieren.

Welche Bedienelemente sind notwendig? Hier kann man sich an existierenden Simulatoren etwas orientieren.

Wie soll eine Simulation gestartet werden? Ist ein Einzelschritt notwendig? Sollten Breakpoints realisiert werden? Ist eine Editmöglichkeit im Datenspeicher sinnvoll? Sollte man unterschiedliche Quarzfrequenzen einstellen können? Welchen Einfluss hat eine andere Quarzfrequenz auf den Simulationsablauf? Wie können die Portein- und -ausgänge durch den Bediener verändert werden? Wie zeigt man dem Bediener, welches der Befehl ist, der als nächstes abgearbeitet wird?
Neben dem Aufbau und der Funktionsweise der GUI muss auch überlegt werden, wie die internen Funktionsgruppen in Software realisiert werden können. U.a. sind folgende Fragen zu beantworten:

Wann und wie wird auf den Programmspeicher zugegriffen? Wie wird der Befehl dekodiert? Wie lassen sich die unterschiedlich langen Argumente vom Befehl trennen? Wie werden die Funktionen der einzelnen Befehle realisiert? Wie wird der Zugriff auf die unterschiedlichen Bänke des Datenarrays bestimmt? Welche Wirkung hat das Destinationbit? Gibt es zusätzliche Auswirkungen auf andere Funktionsgruppen, wenn bestimmte Register verändert werden?

Ganz wichtig ist auch, dass man die einzelnen Schritte in den Testprogrammen verstanden hat. Dazu muss man die Wirkungsweise der einzelnen Befehle gut kennen und das Datenblatt verstehen.

Beispiel: Bedienknopf:
Aus der obigen (unvollständigen) Liste soll der Punkt "Start der Simulation" an einem Beispiel etwas genauer betrachtet werden.

Was passiert bei einem Klick auf den Knopf "Einzelschritt"?

- Das Anklicken sorgt dafür, dass genau ein Befehl abgearbeitet wird und anschließend das Programm auf die nächste Aktion des Bedieners wartet.
- Nach dem Klick wird der Befehl, dessen Adresse im Programmzähler steht, aus dem Programmspeicher gelesen.
- Dieser Befehl wird dem Befehlsdekoder übergeben und der Programmzähler um eins erhöht.
- Der Dekoder analysiert das Bitmuster und sucht nach dem entsprechenden Befehl.
- Ist der Befehl erkannt, wird die entsprechende Funktion z.B. ADDWF aufgerufen.
- Ist der Befehl komplett behandelt worden, wird die GUI aktualisiert.

Hinweis:
Oft ist es notwendig, dass man überlegt, ob das eine oder andere gerade implementierte auf der GUI dargestellt werden soll. Ggf. muss die GUI an diesem Punkt noch nachgebessert werden. Am Ende des Projekts könnte sich sogar herausstellen, dass das eine oder andere Element aus der Planungsphase gar nicht benötigt wird.

Der ganze Programmierprozess ist im Prinzip ein iterativer Vorgang.

- Problem erfassen
- Problemlösung beschreiben
- Codierung
- Test.

2.4 Beispiel

Als Beispiel für das unter 2.3 Beschriebene soll der Befehlsdekoder etwas näher betrachtet werden.
Einen Befehl findet man entweder durch Vergleich des Befehlscode oder durch Suche des passenden Zahlenbereichs, den ein Befehl umfassen kann.

So lässt sich der Befehl ADDWF finden:
Entweder man isoliert den Befehlscode mit der Bitmaske 0x3F00.

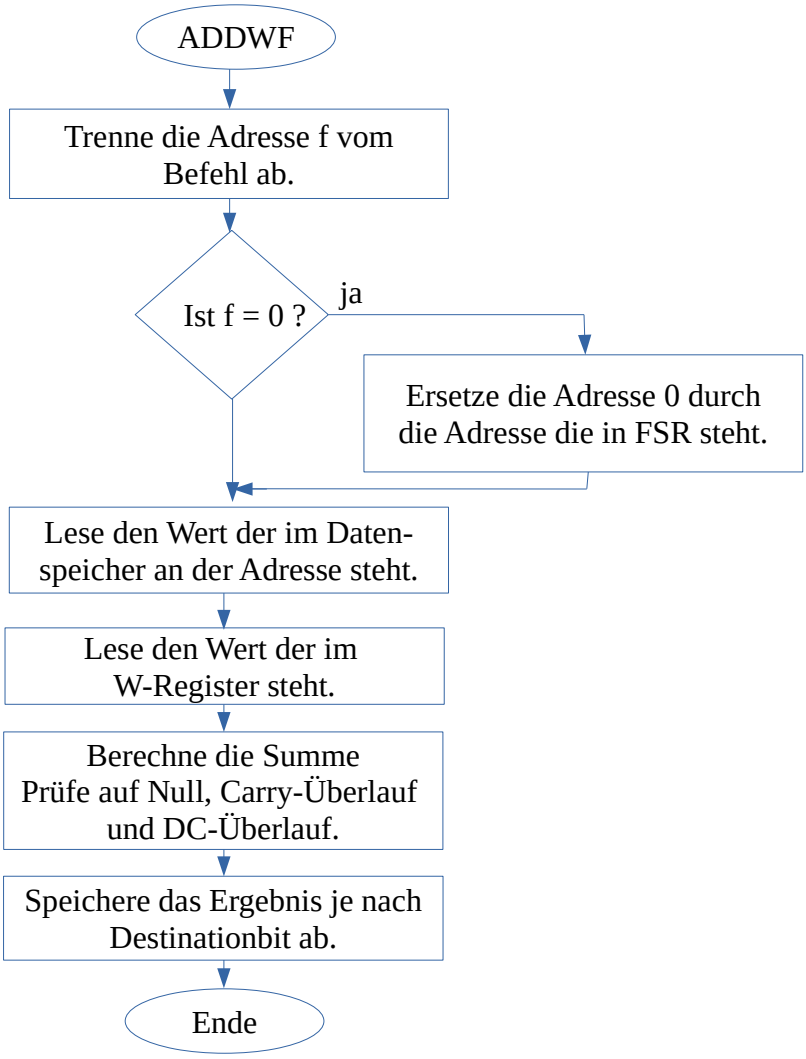
IF (befehl AND 0x3FF) = 0x0700 THEN addwf

Oder man prüft den Zahlenbereich für diesen Befehl.

IF (befehl >= 0x0700) AND (befehl <= 0x07FF) THEN addwf

Beide Methoden funktionieren, **bis auf wenige Ausnahmen**, bei allen Befehlen.

PAP: Funktion des ADDWF Befehls (grober Ablaufplan)



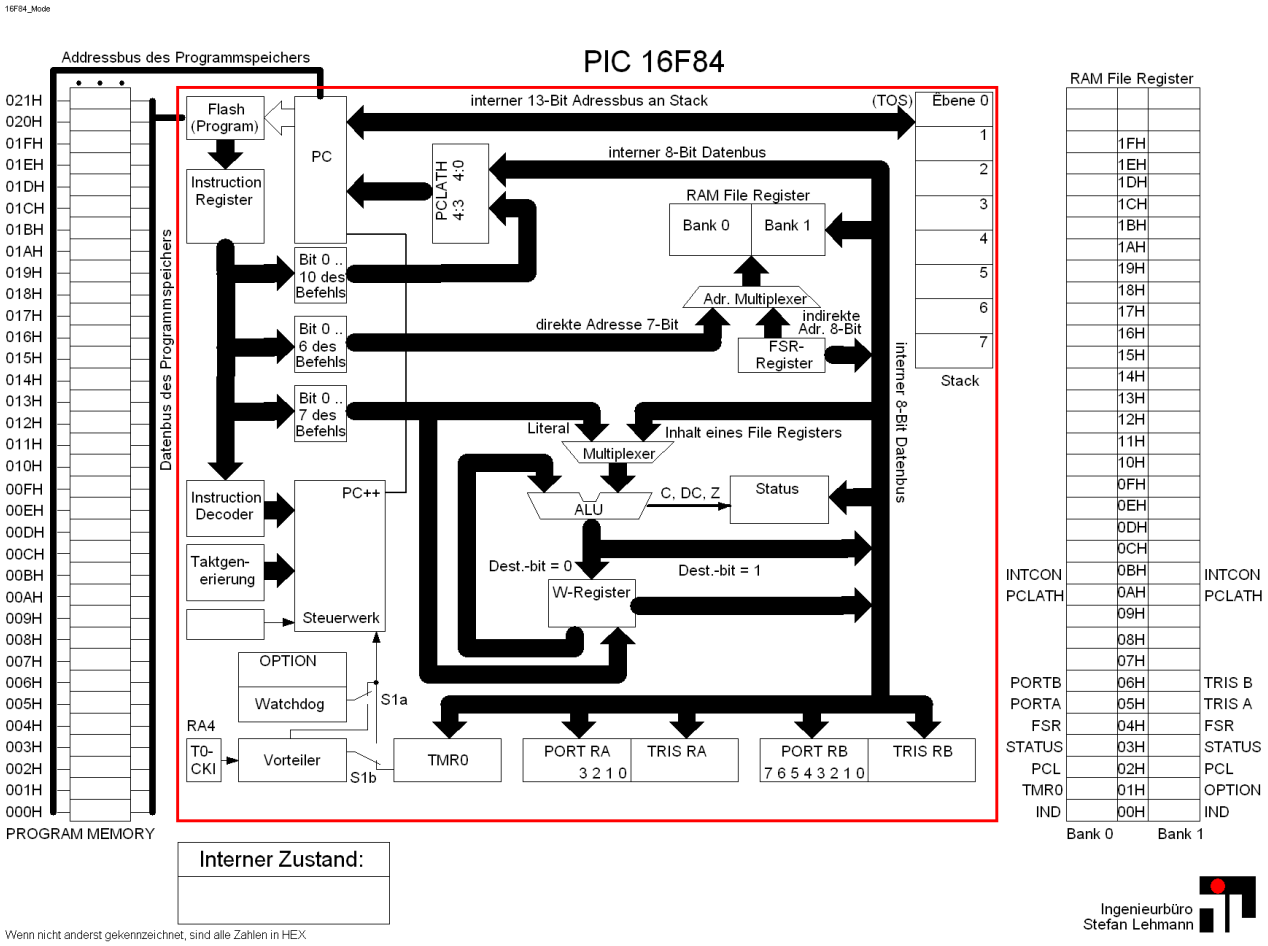


Abbildung 2: Erweitertes Blockschaltbild des PIC 16F84

3 Nachfolgende Punkte sollten in jedem Fall auf der Liste stehen!

- Z-, C- und DC-Flag
- Z-Flag auch beim MOVF-Befehl
- Das Verhalten des C- und DC-Flags bei Subtraktionsbefehlen
- Bankumschaltung mittels RP0
- Spiegelung einiger Register von Bank 0 in Bank 1
- Berücksichtigung des Destinationbits
- Indirekte Adressierung
- Verhalten des PCLATH bei GOTO und CALL
- Verhalten des PCLATH bei Manipulation des PCL-Registers
- Vorteiler beim TMR0
- Vorteiler beim Watchdog
- Watchdog mittels Button oder Checkbox ein- und ausschaltbar
- TMR0 inkrementieren durch den internen Befehlstakt
- TMR0 inkrementieren durch eine passende Flanke an RA4
- Wirkung der Bits im OPTION-Register
- Wirkung des Daten- und Tris-Register bei den Ports
- Stackpointer auf 0-7 beschränken (Ringbuffer)
- Sleep

- und viele weitere, die Sie sich erarbeiten müssen.

4 Weiterführende Quellen

https://entwickler.de/online/development/einfuehrung-programmierung-grundlagen-teil-1-197189.html

https://users.fmi.uni-jena.de/~hinze/eidp-sose16/repository/eidp-vorlesungsteil05-sose16.pdf

https://www.uni-ulm.de/fileadmin/website_uni_ulm/adprostu/Mod_Master/Publikationen/Handreichung_Lernziele-Flyer.pdf

https://ethz.ch/content/dam/ethz/special-interest/infk/inst-cs/lst-dam/documents/Education/Classes/Fall2018/0027_Intro/Slides/18E0918.pdf

Sehr verständliche Einführung in die Problemanalyse geben die folgenden Seiten:
https://www.inf-schule.de/programmierung/kara

http://cgd.zum.de/wiki/Programmieren_lernen_mit_Robot_Karol