Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

UNIVERSITE BORDEAUX 1 Sciences Technologies

RAVEL

Studiengang Systemtechnik

Vertiefungsrichtung Infotronics

Diplom 2007

Michel Gemmet

Sensor zur Erkennung der Anwesenheit eines Fahrzeugs auf Basis der Standards IEEE 802.15.4 und ZigBee

Dozent Pierre Pompili
Experten Eric Stempin

F – Bordeaux, 21.12.2007

Gemmet Michel 2007

Sensor zur Erkennung der Anwesenheit eines Fahrzeugs auf Basis der Standards IEEE 802.15.4 und ZigBee

Capteur de détection de présence de véhicule utilisant les standards IEEE 802.15.4 et ZigBee

Ziel

Es soll ein Sensor entwickelt werden, der mittels Veränderung des Erdmagnetfeldes die Präsenz eines Fahrzeugs auf einem Parkplatz feststellen kann. Der aktuelle Zustand des Parkfeldes soll dem Koordinator über eine drahtlose ZigBee-Kommunikation im 2,4 GHz Frequenzbereich zugestellt werden. Da die Sensoren mittels Batterien betrieben werden, muss bei der Entwicklung besonders auf eine möglichst geringe Energiekonsumation Wert gelegt werden.

Resultate

Die Software Anwendung konnte entwickelt und ausgetestet werden. Für die Hardware konnten durch die Entwicklung der Software und der gemachten Tests einige Verbesserungen aufgefunden werden, die bei einer Revision zur Anwendung gelangen. Die Anwendung wurde auf dem IEEE 802.15.4 Standard aufgebaut. Es wurde also nicht vom gesamten ZigBee-Protokollstapel Gebrauch gemacht, da eine solche Entwicklung deutlich mehr Zeit und Geld beansprucht hätte.

Schlüsselwörter

ZigBee, IEEE 802.15.4, Sensor, Magnetfeld, Fahrzeugerkennung, MSP430, CC2420, 2.4 GHz, Drahtlose Medien, Wireless

Objectif

Il devrait être développé un capteur qui peut constater la présence d'un véhicule sur un parking au moyen du changement du magnétisme terrestre. La condition actuelle du champ de parc devrait être livrée au coordinateur par une communication ZigBee (sans fil) dans la bande de fréquence de 2,4 GHz. Comme les capteurs fonctionneront à l'aide d'une batterie, la solution développée devra être à faible consommation d'energie..

Résultats

Le logiciel application a pu être développée et testée. Pour la partie hardware, quelques améliorations seront apportées suite au développement et aux tests de la partie software. L'application est basée sur le standard IEEE 802.15.4. Il n'a pas été fait ainsi par la pile ZigBee total parce qu'un tel développement aurait revendiqué distinctement plus de temps et argent.

Mots-clés

ZigBee, IEEE 802.15.4, capteur, magnétisme terrestre, détection de présence de véhicule, MSP430, CC2420, 2.4 GHz, connexion sans fil, wireless

Danksagungen

Ich möchte an dieser Stelle an folgende Personen / Institutionen im Speziellen meinen Dank aussprechen:

- Herrn Pierre Pompili für meine Betreuung als Diplomand im Ausland
- Herr Eric Stempin für die Möglichkeit meine Diplomarbeit bei ihm in Bordeaux auszuführen
- Herr Jean-Christoph Martin für seinen Rat und Tat auf Fragen in Sachen Elektronik vor allem im Zusammenhang mit der vorhandenen Hardware
- Herr Stéphane Koehler für seine Hilfsbereitschaft bei Fragen und Gedankengängen im Zusammenhang mit der Diplomarbeit
- Herr Adrien Favot ebenfalls für seine Hilfsbereitschaft bei Fragen und Gedankengängen rund um die Diplomarbeit
- Herr Laurent Magdelaine für seinen Rat und Tat auf Fragen in Sachen Informatik und Programmierung vor allem im Zusammenhang mit dem Algorithmus zur Fahrzeugerkennung
- Büro MOVE für die Ermöglichung einer Ausland-Diplomarbeit

Inhaltsverzeichnis

1. Verfasser

Michel Gemmet
Laboratoire IMS - Bât. A31
351, Cours de la libération
33405 Talence Cedex - France

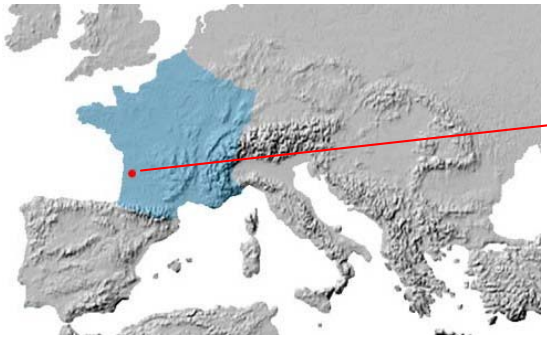
2. Einleitung und Zielsetzungen

Dieses Projekt mit dem Titel „Sensor zur Erkennung der Anwesenheit eines Fahrzeugs auf Basis der Standards IEEE 802.15.4 und ZigBee (CEP)“ wurde im Rahmen einer (Erasmus-)Diplomarbeit von der Unternehmung RAVEL der Hes·so Wallis vorgeschlagen. Die folgenden Abschnitte definieren den genauen Rahmen des Projektinhaltes.

2.1 Vorstellung Unternehmung RAVEL

Im folgenden Abschnitt wird die Unternehmung RAVEL vorgestellt, von welcher der Projektvorschlag dieser Diplomarbeit kommt. Sämtliche praktische Arbeiten wurden hier ausgeführt.

2.1.1 Standort



Réseau Aquitain Véhicules ELectriques
Laboratoire IMS - Bât. A31
351, Cours de la libération
33405 Talence Cedex (Bordeaux) – France

☎ +33 (0) 5 40 00 88 05
+33 (0) 5 40 00 28 00
ravelec@ixl.fr
http://www.ravel-vehicule-electrique.com

2.1.2 Firmenbezeichnung, Rechtsstellung und Struktur

Die Unternehmung RAVEL (Réseau Aquitain Véhicules ELectriques) ist angelehnt an das Mikroelektronik-Labor IXL (http://www.ixl.fr) bzw. IMS (http://www.ims-bordeaux.fr) und bildet eine Zelle für den Technologie-Transfer.

Seine Aufgabe ist es, auf industrielle Probleme zu antworten, indem sie eine Verbindung zwischen Industrie und Forschung bildet.

RAVEL ist eine Technologie-Plattform, auf welche sich die industrielle Umgebung abstützen kann, um ihre Projekte in den Bereichen der Leistungselektronik, der eingebetteten Systeme und des Energiemanagement zu definieren, zu konzipieren und zu fabrizieren.

Sie kann aus der Erfahrung und dem Fachwissen, sowie der technischen Mittel, des Mikroelektronik-Labors IXL profitieren, um im Elektro- und Hybridfahrzeugbereich zu arbeiten.

Durch die Unterstützung und Führung des Leiters der Zelle, werden sämtliche Tätigkeiten als Team-Projekte durchgeführt. Diese Durchführungsart gestattet eine schnelle Reaktionszeit und erlaubt daher eine Garantie für eine minimale Verzögerungszeit an die Kundengemeinschaft.

Die Partnerschaft mit der Industrie konkretisiert sich im Allgemeinen durch Forschungsverträge, Studienverträge, Testverträge, Gutachten, aber auch durch die Bereitstellung von Ressourcen und technischen Mitteln durch das Mikroelektronik-Labors IXL.

Die Struktur wird vom Verband ADERA (Association pour le Développement de l'Enseignement, et des Recherches auprès des universités, des centres de recherche et des entreprises d'Aquitaine) geleitet. (http://www.adera.fr)

2.1.3 Tätigkeitsbereich

Der Tätigkeitsberich von RAVEL ist der folgende:

Eingebettete Systeme
- Anwendungen auf Basis von Mikroprozessoren und Mikrocontrollern (PIC, HC12, HC08, ARM, PowerPC, Intel XScale, TI MSP430)
- Feldbus (CAN, LIN, ISO9141, …)
- Datenspeicher
- Ortungssysteme
- Technologien GSM / GPRS
- Technologie GPS
- CEM
- Echtzeitsysteme (Linux)
- CAO Elektronik
- Schnittstellen Mensch-Maschine

Leistungselektronik
- Stromversorgungen und Netzumwandler (1 W bis 100 kW)
- Batterie-Aufladegeräte
- Resonanzwandler
- AC/DC Wandler
- Superkondensatoren
- Vereinigung der elektrischen Energie von Hybridquellen (Energie / Leistung)

Energiemanagement
- Charakterisierung und Verhaltensregelung der Batterien
- Autonome Systeme zur Energieproduzierung
- Batteriebetriebene Überwachungsmodule
- Energiemanagementsysteme
- Solarzellen
- Brennstoffzellen
- Windturbinen

Elektro- und Hybridfahrzeuge
- Elektrofahrzeuge
- Elektroboote
- Elektrolastwagen
- Spezifische Fahrzeuge

2.2 Projektantrag

Es soll ein Sensor entwickelt werden, der die Präsenz eines Fahrzeuges auf einem Parkplatz ermitteln kann. Der aktuelle Zustand des Parkplatzes soll über eine drahtlose Verbindung einem Koordinator zugeschickt werden, der diese Daten dann verwaltet und auswertet.

Die Grundidee hinter diesen Sensoren ist in erster Linie die folgende:
In erster Linie soll durch den Einsatz dieser Sensoren in einem Parkhaus ermittelt werden, wie viele Parkplätze zurzeit besetzt sind und wie viele noch frei sind. In einem zweiten Schritt soll einem Fahrzeuglenker, der das Parkhaus betritt, der Weg zum nächsten freien Parkplatz angezeigt werden. Beispielsweise durch ein Licht oder durch andere Wegweiser. Mit diesen Sensoren ist ja eine genaue Zustandsbestimmung der einzelnen Parkfelder möglich und das System weiss somit genau welcher Parkplatz noch frei ist und welcher nicht.

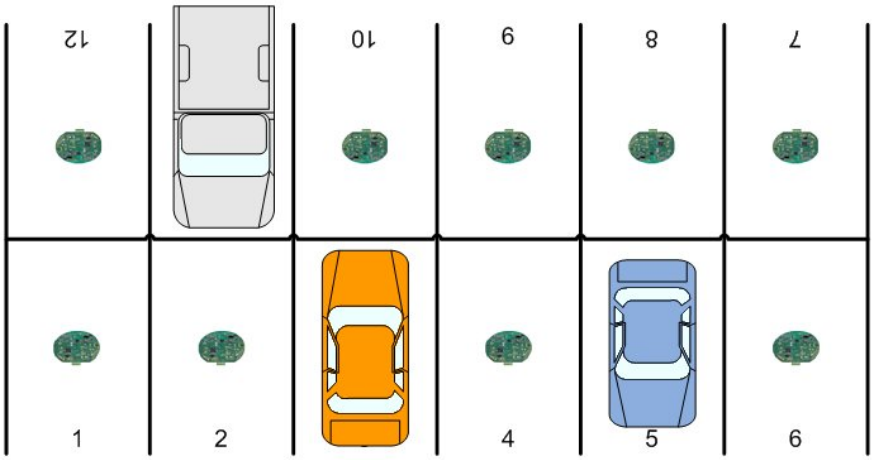


Abb. 1 : Schema Projektantrag - Parkplatz

Der Einsatzbereich dieser Sensoren ist weitläufig und kann daher sicherlich später auch für andere Anwendungen eingesetzt werden.

Beteiligte	verant-wortlich	mitbeteiligt	interessiert	Name	Telefon
Hes·so Wallis (Abteilung Infotronics)	X			Pierre Pompili	+41 (0)27 606’87’58
Projektverantwortlich(e) (RAVEL)		X X		Eric Stempin Jean-Christoph Martin	+33 (0)5 40’00’88’05 +33 (0)5 40’00’27’70
C-Zame Technologies und/oder andere Parkplatzinhaber			X	-	-

Projektart	Nutzung	Projektgrösse
☒ Neulösung ☐ Erweiterung / Ànderung	☒ wiederholt ☐ einmalig	☐ gross ☒ mittel ☐ klein
Termine	Projektstart: 10.09.2007 Projektabgabe: 21.12.2007 Projektdauer: 15 Wochen	

Der ursprünglich eingegangene Projektvorschlag von RAVEL ist im Anhang 1 beigelegt.

2.3 Pflichtenheft

Wie im Projektantrag (☞ *Kapitel 2.2*) bzw. im Projektvorschlag (🗐 *Anhang 1*) angetönt, liegt das Hauptaugenmerk hierbei dem Energieverbrauch. Viele Parkplätze befinden sich ein wenig abgelegen von Stromquellen, so dass meist ein Betrieb mittels Batterien unumgänglich ist. Daher soll der Sensor einen möglichst geringen Energieverbrauch erzeugen, denn die Batterien sollen möglichst so lange wie möglich halten.

Die einzigen präzisen Projektrichtlinien sind durch den Projektvorschlag (🗐 *Anhang 1*) vorgegeben. Diese lassen aber einen relativ grossen Spielraum offen und daher ist es auch nahe liegend, dass das Projektziel während der Durchführung dieser Diplomarbeit das ein oder andere Mal neu überdacht wurde.

Die einzigen klaren und allgemein gültigen Vorgaben sind die folgenden:

- Geringer Energieverbrauch
  - Benutzung des Mikrokontrollers MSP430F169 von Texas Instruments für die Anwendung
  - Benutzung des Transceivers CC2420 von Chipcon für die Partie RF
- Kommunizierung der Sensorzustände mittels dem Standard ZigBee (bzw. IEEE 802.15.4)
- Schluss-Anwendung muss in der IAR Workbench Programmierumgebung funktionieren

Im Rahmen dieser Diplomarbeit wird primär die drahtlose Kommunikation zwischen Sensor und Koordinator betrachtet d.h. der Betriebsablauf des Sensors. Der Sensor selber d.h. die Hardware wurde schon in einem Rahmen einer früheren Diplomarbeit erstellt (☞ *Kapitel 4.2*). Als Koordinator / Router dient anfangs ein eingekauftes Modul namens „XBee Pro Serie 1“ von der Firma MaxStream (🌍 http://www.maxstream.net). In einem späteren Zeitpunkt, je nach Verlauf der Diplomarbeit noch während dieser Diplomarbeit, wird der Koordinator auch selber erstellt.

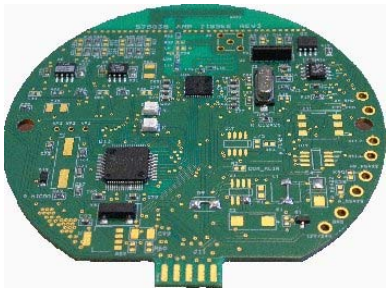


Abb. 2 : Hardware (CEP)

Abb. 3 : MaxStream XBee Pro Modul

Die Messung der Präsenz eines Fahrzeugs erfolgt über magnetische Sensoren. Die eingesetzten Sensoren messen die Veränderung des Erdmagnetfelds, das auftritt wenn ein Fahrzeug auftaucht.

Abb. 4 : Erdmagnetfeld bei einem Fahrzeug

2.4 Projektumriss

In diesem Abschnitt des Dokuments wird der Projektumriss aufgezeigt. D.h. wer ist im Projekt mit involviert und wie sieht der Zeitplan aus.

2.4.1 Struktur

Das Projekt kann in die zwei Hauptteilbereiche Hardware und Software unterteilt werden.

Die Hardware wurde - wie schon erwähnt - bereits in einer früheren Diplomarbeit erstellt.
Als Ansprechpartner für Fragen und Bemerkungen steht diesbezüglich Herr Jean-Christophe Martin zur Verfügung.

Die Software kann wiederum in zwei Teilbereiche unterteilt werden.
Der erste Teil ist ein Algorithmus zur Erkennung der Fahrzeuge anhand der magnetischen Sensoren. Dieser Teil wurde ebenfalls in Form einer früheren Projektarbeit bereits erstellt. Jedoch muss er auf die nun zur Verfügung stehende Hardware ein wenig angepasst werden. Als Ansprechpartner für Fragen und Bemerkungen steht mir in diesem Fall Herr Laurent Magdaleine zur Verfügung.
Der zweite Teil ist die Anwendung für die RF-Kommunikation mittels ZigBee-Protokollstapel. Dieser Teil ist die Hauptaufgabe dieser Projektarbeit.

Projektleiter ist Herr Eric Stempin.

Zum Systemtest und für allgemeine beratende Gespräche im Zusammenhang mit diesem Projekt stehen zusätzlich die Herren Stéphane Koehler und Adrian Favot zur Verfügung.

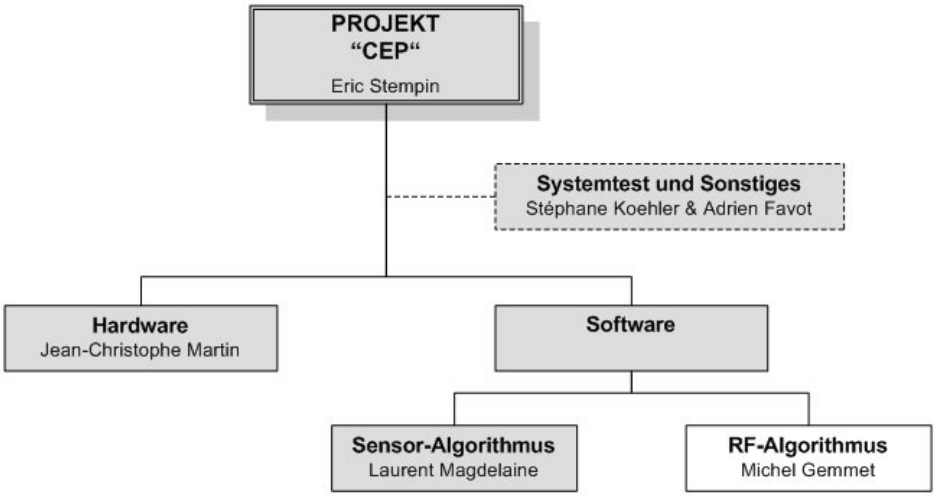


Abb. 5 : Projektstruktur

2.4.2 Zeitplan

Der Zeitplan gibt einen Überblick über den geplanten Ablauf des Gesamtprojekts:

Tätigkeit	Sept. 2007			Okt. 2007					Nov. 2007				Dez. 2007		
	10.	17.	24.	01.	08.	15.	22.	29.	05.	12.	19.	26.	03.	10.	17.
Einarbeitung															
Ferien / Frei															
Entwicklung															
Systemtest															
Dokumentation															

Abb. 6 : Projektzeitplan

2.5 Spezifikation

In diesem Abschnitt des Dokuments gilt es die allgemein gültigen Regeln zum Projekt zu definieren. D.h. welche Software wird benutzt, wie sieht die Syntax und die Struktur aus.

2.5.1 Software

Lösungsvarianten :

- IAR Embedded Workbench for MSP430 v3.41 (http://www.iar.com)
- MSPGCC – GCC Toolchain for MSP430 (http://mspgcc.sourceforge.net)
- usw.

Aufgrund der umfassenderen Anwendermöglichkeiten und damit ein gewisser Industriestandard eingehalten wird, wurde die Programmierumgebung „IAR Embedded Workbench“ gewählt, um das Projektvorhaben umzusetzen.

Der Hauptvorteil vom kostenpflichtigen „IAR Embedded Workbench“ gegenüber der Freeware-Tool „MSPGCC“ ist die Debug Möglichkeit.

Es ist vielleicht noch zu vermerken, dass es vom kostenpflichtigen „IAR Embedded Workbench“ auch eine Gratis-Version mit dem Namen „IAR Embedded Workbench KickStart“ gibt. Mit dieser Version lassen sich aber nur kleine Anwendungen kompilieren (max. 4 KByte).

2.5.2 Syntax

Die Syntax der Anwendung wird folgendermassen definiert:

Typ	Definition	
Methode / Funktion / Operation	getMethodName get_method_name	[Beginn mit Kleinbuchstabe] [Trennung durch _]
Attribut / Variable	attributName gAttributName (global)	[Beginn mit Kleinbuchstabe] [Beginn mit kleinem g]
Konstante	cConstName	[Beginn mit kleinem c]
Makro	MAKRO_NAME	[Grossbuchstaben]
Definition	DEFINITON_NAME	[Grossbuchstaben]
Enumeration	eEnumerationName	[Beginn mit kleinem e]
Union	uUnionName	[Beginn mit kleinem u]
Structure	structureName_s	[Ende mit kleinem _s]
Typedefinition	typedefName_t	[Ende mit kleinem _t]
Pointer	pPointerName	[Beginn mit kleinem p]
…	…	…

3. Konzept

Im Konzept wird die Idee des Aufbaus und die Hard- und Softwareumgebung aufgezeigt.

3.1 Hard- und Softwareumgebung

Auf der Hardwareseite stehen folgende Komponenten zur Verfügung:

- 2 CEPs Version 2.0 (mit MSP430F169 Mikrocontroller und CC2420 RF Transceiver)
- 7 CEPs Version 3.0 (mit MSP430F169 Mikrocontroller und CC2420 RF Transceiver)
- 7 Batteriegehäuse für/mit 2 D-Batterien
- RS-232 Interface
- XBee Pro Serie 1 inkl. Development Board USB (Typ : XBP24-AWI-001) inkl. USB Kabel
- XBee Pro Serie 1 inkl. Development Board RS-232 (Typ : XBP24-AWI-001) inkl. RS-232 Kabel
- Tektronix Oszilloskop (Typ : DPO 4054)
- MSP430 USB-Debug-Interface (JTAG) inkl. USB Kabel zum PC
- 2 Agilent Power Supplys
- etc.

Seitens der Software wird folgendes benutzt:

- IAR Embedded Workbench für MSP430 v.3.41
- MSPGCC – GCC Toolchain for MSP430
- MSYS – Minimal SYStem v.1.0.10
- X-CTU (XBee Software) von Digi-Maxstream
- Windows Hyperterminal für Testausgaben und Debugs
- Microsoft Visual C Sharp Express
- Microsoft Visual C++ Express
- Desktop Activity Recorder für Systemtests
- etc.

3.2 Aufbau

Dieser Abschnitt des Dokuments zeigt die grundlegende Aufbaustruktur des Projektes in Form der Softwarearchitektur und eines Modells.

3.2.1 Softwarearchitektur

Die Kommunikation zwischen Mikrokontroller und RF Transceiver erfolgt mittels SPI (Serial Peripheral Interface). Diesbezüglich ist folgende Architektur vorgesehen:

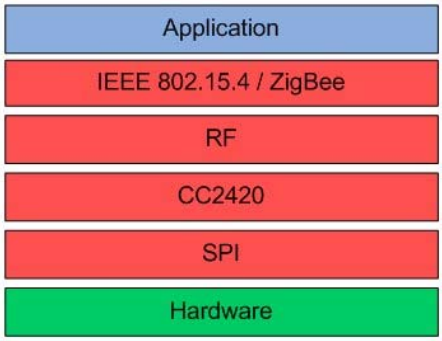


Abb. 7 : Softwarearchitektur

4. Realisierung

Dieser Abschnitt dokumentiert alle praktischen Arbeiten, die während der Durchführung des Projekts getätigt wurden. Es wird versucht die Idee hinter den vollzogenen Arbeiten, sowie auch deren Implementation, Validierung und der durchgeführten Tests (☞ *Kapitel 5*), zu erläutern. Des Weiteren sind mögliche Verbesserungsvorschläge oder weiterführende Gedankengänge aufgeführt (☞ *Kapitel 6)*.

4.1 Grundlagen der Standards IEEE 802.15.4 und ZigBee

Im folgenden Abschnitt werden die Standards IEEE 802.15.4 und ZigBee in ihren Grundlagen erläutert, dies allerdings ohne zu Tief in die Details zu gehen.

Es gilt vorgängig zu bemerken, dass wenn man von ZigBee spricht, dies den Standard IEEE 802.15.4 eigentlich mit beinhaltet. In den folgenden Abschnitten wird also mit ZigBee der ganze Protokollstapel gemeint und es wird nicht weiter zwischen IEEE 802.15.4 und ZigBee unterschieden.

4.1.1 Charakteristiken und Kenndaten

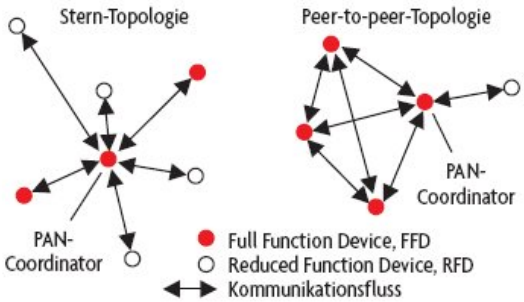
ZigBee (und sein zugrunde liegender Standard IEEE 802.15.4) ist ein offener Funknetz-Standard für zuverlässige, kostengünstige und verlustleistungsarme Sensor- und Aktor-Netzwerke, der sich gegenwärtig teilweise noch in der Markteinführung befindet.

Die allgemeinen Charakteristiken von ZigBee sind die folgenden:

- Datenübertragungsrate von 20 bis 250 kBit/s
- Reichweite bis zu 100 m
- 3 Frequenzbereiche mit insgesamt 26 Kanälen

Bereich	Frequenzband	Region	Kanalnummer	Datenrate	Modulation
868 MHz	868 – 838,6 MHz	Europa	0	20 kBit/s	BPSK
915 MHz	902 – 928 MHz	Amerika	1-10	40 kBit/s	BPSK
2.4 GHz	2400 – 2483,5 MHz	weltweit	11-26	250 kBit/s	O-QPSK

- Störungssicher
- Unterstützt verschiedene Netzwerktopologien



Auf dieserGrundlage lassen sich nun weitere leistungsfähige und flexible vermaschte Netzwerke aufbauen wie z.B. „Cluster-Trees“.

Abb. 8 : Grundlegende Netzwerktopologien

- Kurze Verbindungszeiten : < 25 ms
- Protokollstapel von 25 bis 60 kBytes
- 128-Bit AES Verschlüsselung und Authentifizierung
- Sendeleistung bis 1 mW
- Schiedsrichteralgorithmus CSMA/CA
- etc.

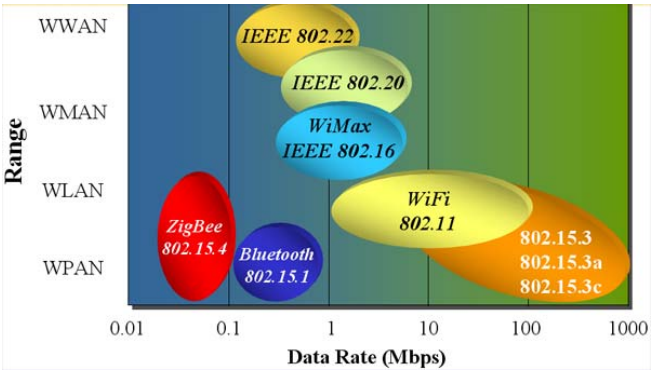


Abb. 9 : Positionierung in Bereich und Datenrate

4.1.2 Komponenten

Grundsätzlich kann man die Geräte eines „Low Data Rate – Wireless Personal Area Networks“ (LD-WPAN) - wie ZigBee es ist - in zwei Gruppen einteilen:

- FFD (Full Function Device)
- RFD (Reduced Function Device)

Ein FFD beherrscht die volle Funktionalität des Protokollstapels, während ein RFD nur eine reduzierte Funktionalität aufweist.
FFDs können sowohl mit RFDs, als auch mit anderen FFDs kommunizieren, während RFDs nur mit FFDs Daten austauschen können.

In einem ZigBee-Netzwerk gibt es folgende 3 Komponenten:

- Koordinator
- Router
- Endgerät

In einem LR-WPAN nach dem IEEE 802.15.4 Standard muss als PAN Koordinator immer ein FFD fungieren. Der Koordinator stellt administrative Funktionen bereit. Diese Funktionen beinhalten unter Anderem die Adressierung der einzelnen Endgeräte und Synchronisation der Datenübertragung bei Verwendung von „Slotted CSMA-CA“ (🗐 *Anhang 1*). Dabei übernimmt er je nach Netzwerktopologie weitere Aufgaben, wie Routing oder Sicherheitsfunktionen.
Der PAN Koordinator ist für die Zugriffskontrolle des Netzes zuständig. Er kann zusätzlichen Endgeräten erlauben dem Netzwerk beizutreten und diese auch wieder aus dem Netz weisen.

Die Router in einem LR-WPAN sind optional. Ihre Existenz hängt von der Netzwerktopologie ab. In gleicher Weise wie der Koordinator muss ein Router als FFD fungieren. Seine Hauptaufgabe liegt in der Weiterleitung der über ihn gesendeten Nachrichtenpakete.

Ein Endgerät ist meist dann der Sensor bzw. „das Arbeitstool“ welches seinerseits meist eine Messung durchführt und deren Resultat dann an den Koordinator schickt. Der während dieser Diplomarbeit zu programmierende Sensor ist ein solches Endgerät.
Endgeräte können sowohl als FFDs als auch als RFDs fungieren.

4.1.3 Protokollstapel

Die Architektur des ZigBee-Protokollstapels basiert auf dem OSI-Modell (7 Schichten). Jede Schicht stellt seinen benachbarten Schichten eine gewisse Anzahl Primitive zur Verfügung, welche durch den Standard [01] und [02] spezifiziert sind.

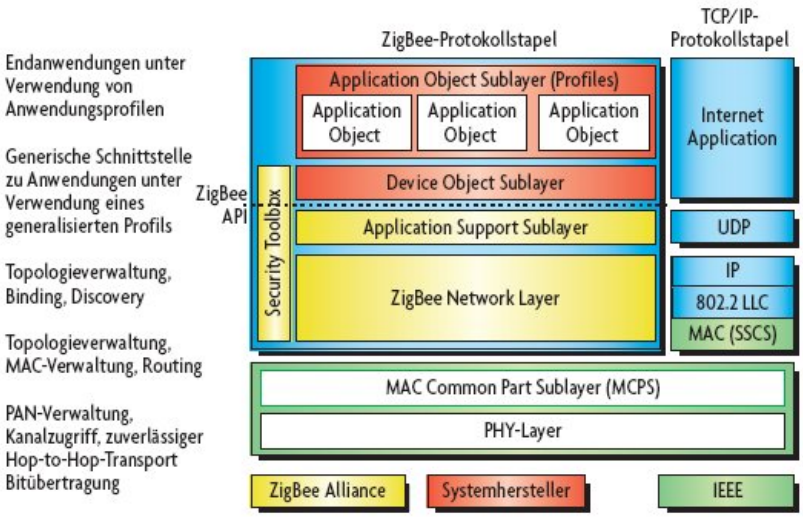


Abb. 10 : ZigBee-Protokollstapel

Wie in Abbildung 10 ersichtlich ist, basiert der ZigBee-Protokollstapel auf dem Standard IEEE 802.15.4 [01]. Dieser definiert die untersten 2 Schichten - die physikalische Schicht (PHY) und die MAC Schicht – und wird spezifiziert von der IEEE[1].
Aufbauend auf diese beiden Schichten definiert die ZigBee Allianz[2] die höher gelegenen Netzwerk- (NWK) und Applikations-Schichten (APL) [02].
Der Applikationsrahmen bleibt dem Systemhersteller überlassen.

Die einzelnen Schichten haben die folgenden Aufgabenbereiche:

PHY-Layer

- Senden und Empfangen von PPDUs über den Radio-Kanal:
- Aktivieren und Deaktivieren der Sende-Empfangseinheit
- Signaldetektion
- Kanalqualitätsanzeige
- Kanalselektion
- Kanalbelegung
- etc.

MAC-Layer

- Aufbau der logischen Kanäle:
- Beacon Management
- Kanalzugriff mit garantierten Zeitschlitzen
- Fehlerbehandlung mit Bestätigung
- Sicherheitsfunktionen (Der Standard unterstützt zwar Verschlüsselung auf MAC Layer Ebene, den Austausch der Schlüssel wird aber an höhere Schichten delegiert.)
- etc.

[1] IEEE : Das Institute of Electrical and Electronics Engineers ist ein weltweiter Berufsverband von Ingenieuren aus den Bereichen Elektronik und Informatik (http://www.ieee.org).
[2] ZigBee Allianz : Die ZigBee Allianz ist ein Zusammenschluss von derzeit mehr als 150 Unternehmen, welche die weltweite Entwicklung dieser Technologie vorantreiben (http://www.zigbee.org).

NWK-Layer

- Vergabe der 16-Bit Netzwerkadressen
- Aufbau des Netzwerks
- Teilnahme und Verlassen von Geräten in bzw. aus dem Netz
- Sicherheit des zu übertragenden Frames
- Routing in einem Mesh- oder Tree-Netzwerk
- etc.

APL-Layer

- Definition des logischen Geräte Typs
- Multiplexing der eingehenden Daten
- Anwendung/Entfernung von Sicherheitsmechanismen auf dem Applikation Layer
- Nachrichten Reflexion für indirekte Adressierung
- Verwaltung der Binding-Tabelle
- Weiterleitung der Nachrichten zwischen Geräten (welche über Binding verbunden)
- etc.

ZigBee beschreibt alle sieben Schichten des OSI-Models, wobei festgelegt ist, dass die unteren drei Schichten durch den Standard IEEE 802.15.4 abgedeckt werden. Die Schichten vier bis sechs werden dann durch den ZigBee-Protokollstapel abgedeckt, während die siebte Schicht die eigentliche Anwendung darstellt. Die folgende Abbildung zeigt eine grafische Darstellung der Zuordnung der ZigBee-Protokolle zum OSI-Model.

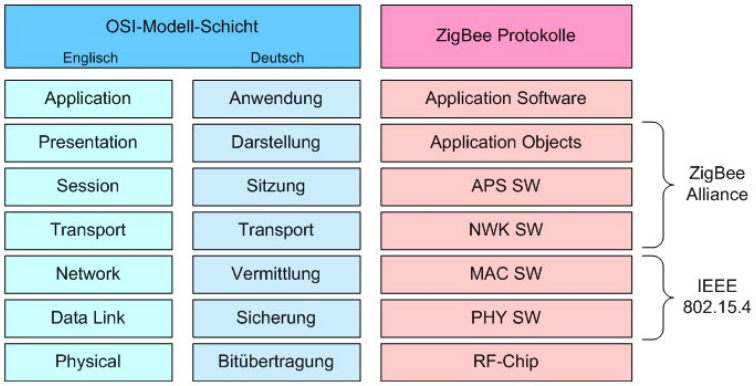


Abb. 11 : ZigBee im OSI-Modell

Der ZigBee Protokollstapel ist streng modular aufgebaut, so dass der Anwender-Software möglich ist, auf jeder beliebigen Schicht aufzusetzen. Daraus ergeben sich eine Vielzahl von Implementierungsmöglichkeiten, die in der folgenden Abbildung dargestellt sind.

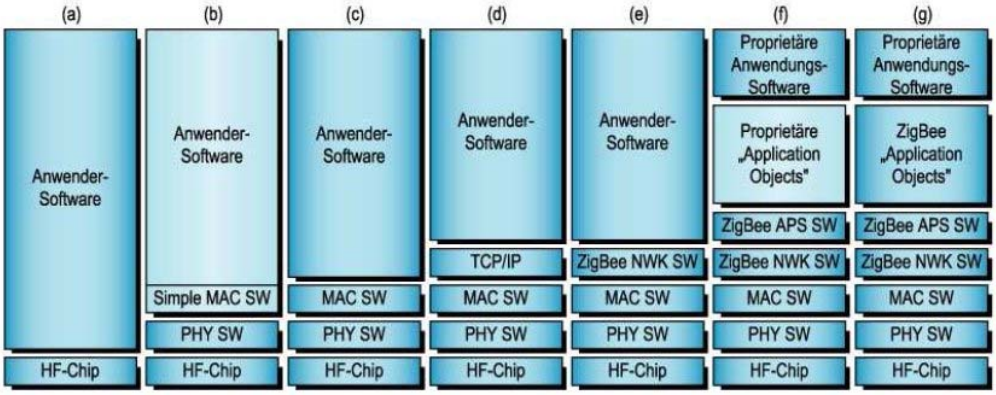


Abb. 12 : Implementierungsmöglichkeiten auf dem ZigBee-Protokollstapel

In diesem Projekt wird schlussendlich die Implementierungsmöglichkeit (c) angewandt (☞ *Kapitel 4.3*).

4.1.4 Frame-Struktur

Hier wird der Frameaufbau bzw. die Frame-Struktur der Schichten NWK, MAC und PHY des ZigBee-Protokollstapels kurz betrachtet.

NWK-Layer

Auf Network-Ebene stehen zwei verschiedene Rahmenarten zur Verfügung:
- Data
- NWK-Command

2 Byte	m Byte	n Byte
Frame Control	Routing Fields	Data Payload
NHR		NSDU

Abb. 13 : Daten-Rahmen auf NWK-Layer-Ebene

Die Daten-Rahmen beinhalten die Daten auf NWK-Ebene oder den Rahmen-Inhalt der höheren Ebene.

2 Byte	m Byte	1 Byte	n Byte
Frame Control	Routing Fields	NWK Command Identifier	NWK Command Payload
NHR		NSDU	

Abb. 14 : NWK-Command-Rahmen auf NWK-Layer-Ebene

Die NWK-Command-Rahmen erlauben eine Realisierung einer zentralisierten Verwaltung eines Netzwerks.

Ein detaillierterer Beschrieb zu den einzelnen Rahmenstrukturen ist in [02] nachzulesen.

MAC-Layer

Auf MAC Ebene stehen vier verschiedene Rahmenarten zur Verfügung:
- Beacon
- Data
- ACK
- MAC-Command

2 Byte	1 Byte	4 / 10 Byte	2 Byte	k (7 - 16) Byte	m Byte	n Byte	2 Byte
Frame Control	Sequence Number	Addressing Fields	Superframe Specification	GTS Fields	Pending Address Field	Beacon Payload	Frame Check Sequence
MHR			MSDU				MFR

Abb. 15 : Beacon-Rahmen auf MAC-Layer-Ebene

Die Beacon-Rahmen erlauben die Synchronisation der Netzteilnehmer.

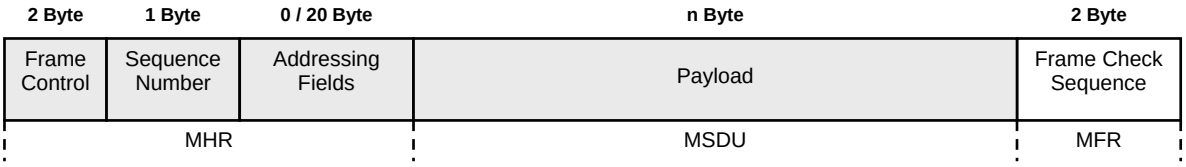


Abb. 16 : Daten-Rahmen auf MAC-Layer-Ebene

Die Daten-Rahmen beinhalten die Daten auf MAC-Ebene oder den Rahmen-Inhalt der höheren Ebene.

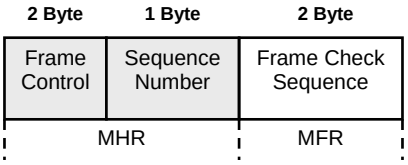


Abb. 17 : Acknowledge-Rahmen auf MAC-Layer-Ebene

Die Acknowledge-Rahmen melden den erfolgreichen oder fehlerhaften Empfang eines Datenpakets an den Sender zurück.

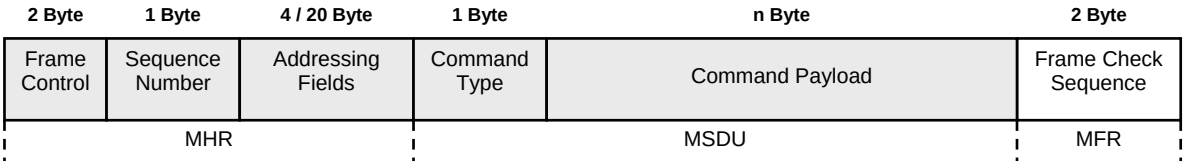


Abb. 18 : MAC-Command-Rahmen auf MAC-Layer-Ebene

Die MAC-Command-Rahmen stellen einen Mechanismus zur Steuerung und Konfiguration von Knoten zur Verfügung. Darüber hinaus erlauben die Beacon-Rahmen eine zusätzliche Erweiterung der Netzfunktionalität.

Ein detaillierterer Beschrieb zu den einzelnen Rahmenstrukturen ist in [01] nachzulesen.

PHY-Layer

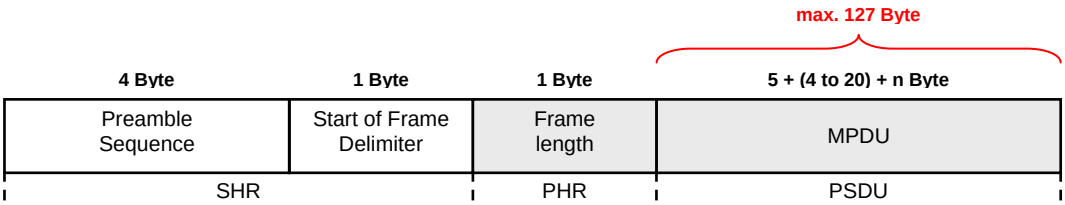


Abb. 19 : Rahmen auf PHY-Layer-Ebene

Der PHY-Layer-Rahmen ergänzt den zu übertragenen MAC-Layer-Rahmen um layerbezogene Steuerinformationen. D.h. der MAC-Layer-Rahmen wird mit einem Synchronisations-Header ergänzt, der, wie auch bei drahtgebundenen Systemen, die Synchronisation des Empfängertakts auf den Sendetakt ermöglicht. Des Weiteren wird noch die Länge des MAC-Layer-Rahmens hinzugefügt. Die maximale Grösse auf MAC-Ebene beträgt 127 Bytes, jedoch definiert ZigBee darüber hinaus auch die Möglichkeit grössere Pakete fragmentiert zu versenden.

Ein detaillierterer Beschrieb zu den einzelnen Rahmenstrukturen ist in [01] nachzulesen.

4.2 Design des Hardwaremodules

Wie bereits weiter oben in diesem Dokument erwähnt, wurde das Hardwaremodule während einer früheren Projektarbeit erstellt.

Zu Beginn der Arbeit stand die Revision 2 der Hardwareplatine bereit - die Revision 3 war gerade in Bearbeitung. Der Unterschied dieser beiden Revisionen liegt aber nur beim Speisungsteil der Schaltung. Revision 3 kann nun mit Batterien betrieben werden.

Anmerkung : Die Revision 1 wurde nur zur Entwicklung des Algorithmus für die Erkennung eines Fahrzeugs benutzt. Der RF-Teil wurde aber hier nicht implementiert, da bei der Entwicklung des Algorithmus schon erste „notwendige“ Verbesserungen erkannt wurden.

4.2.1 Die Platine

Hier wird die Platine (Revision 3) aufgezeigt und die wichtigsten Komponenten darauf erläutert:

Vorderansicht

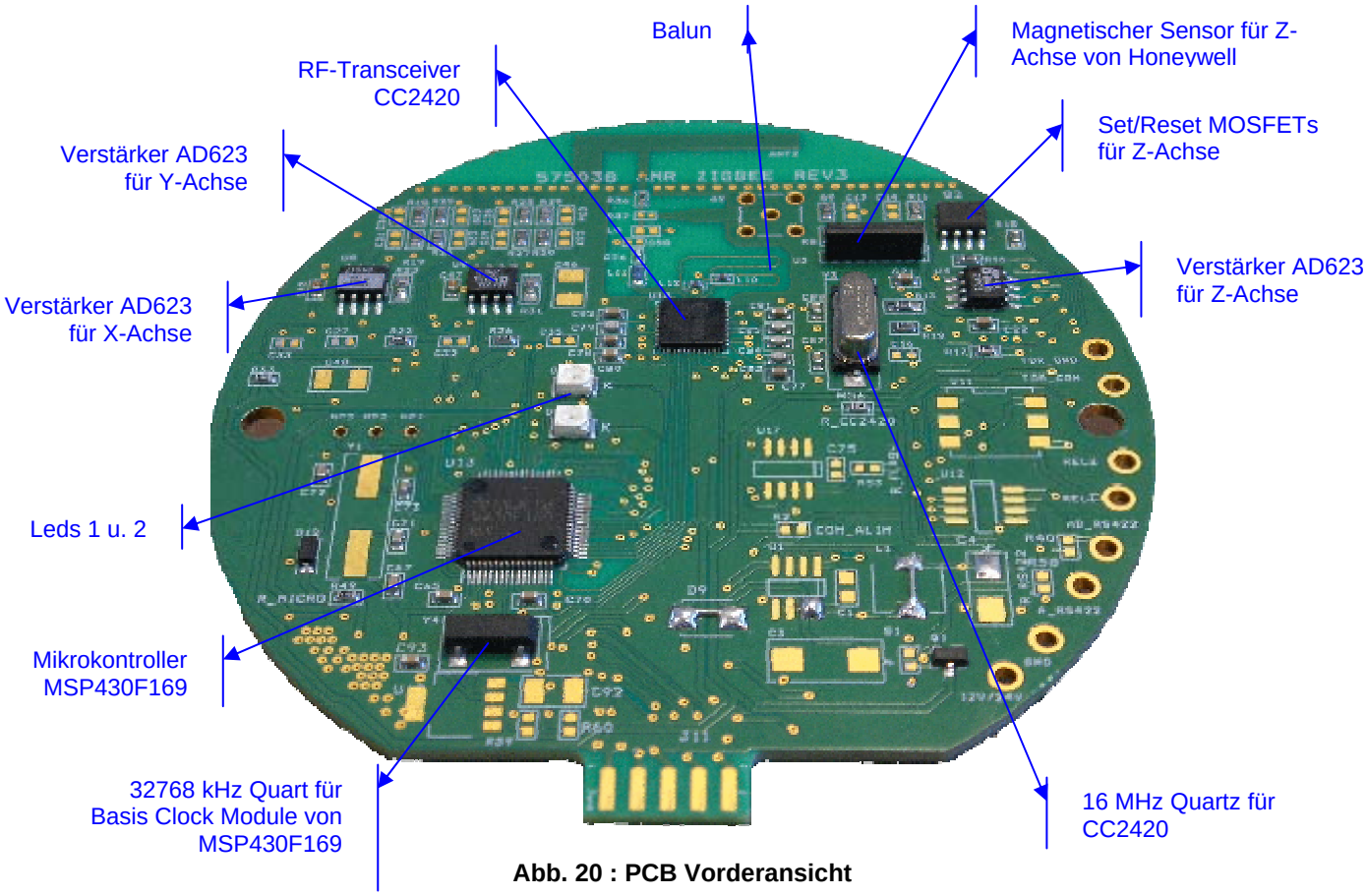


Abb. 20 : PCB Vorderansicht

Hinteransicht

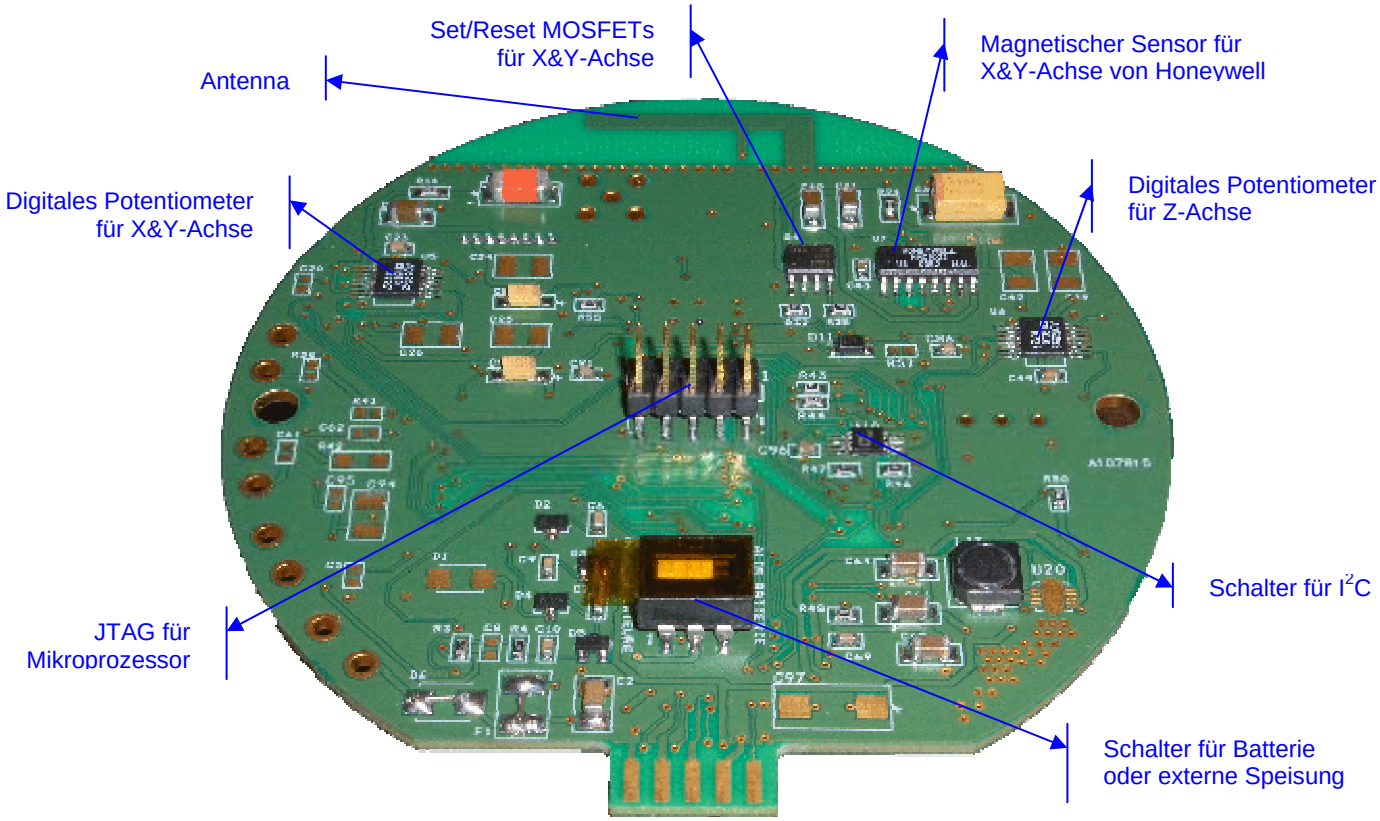


Abb. 21 : PCB Hinteransicht

4.2.2 Das Schema

Um eine gute Software-Anwendung zu schreiben, ist es nötig die allgemeine Funktionsweise der Hardware zu verstehen.

Daher zeigt dieser Abschnitt die wichtigsten Teile des Hardwareschemas auf und erklärt die allgemeine Funktionsweise.

A) Partie RF

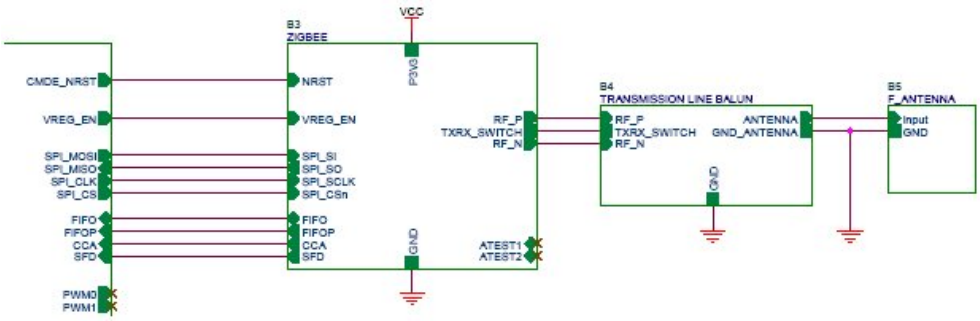


Abb. 22 : Partie RF des Elektroschemas

Der RF-Transceiver CC2420 von Chipcon [05] kommuniziert mit dem Mikrokontroller MSP430F169 [03] [04] von Texas Instruments grundsätzlich mittels SPI-Bus.

Der CC2420 hat zwei FIFO-Buffer mit einer Grösse von 128 Bytes.
Einer ist für den Empfang (RXFIFO) und der andere für den Versand (TXFIFO) von ZigBee-Paketen vorgesehen. Der Inhalt des TXFIFO wird über ein Balun (🗐 *Anhang 1*) zur Antenne zum Versand gesendet. Empfängt der CC2420 ein gültiges ZigBee-Paket, so legt er es in den RXFIFO und indiziert dem Mikrokontroller über die Pins (Flags) `FIFO`, `FIFOP`, `CCA` und `SFD` diesen Empfang. So weiss der Mikrokontroller, dass er über den SPI-Bus den RXFIFO auslesen muss.

Mittels Pin `VREG_EN` kann der CC2420 ausgeschaltet werden. Um eine möglichst geringe Energiekonsumation zu haben ist es zwingend den CC2420 während Sendepausen auszuschalten.

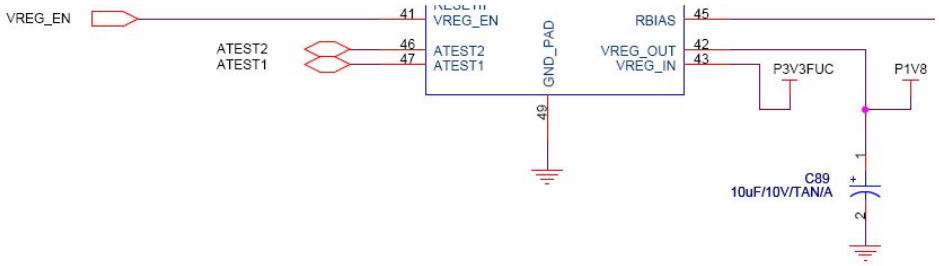


Abb. 23 : Speisung vom RF-Transceiver CC2420

Über den Pin `VREG_IN` wird der CC2420 mit 3.3 Volt gespiesen. Der CC2420 transformiert dann diese Spannung von 3.3 Volt in eine Spannung von 1.8 Volt um und gibt diese an Pin `VREG_OUT` aus. Dies aber nur, wenn der Pin `VREG_EN` aktiv ist. Ansonsten wird an `VREG_OUT` keine Spannung ausgegeben. Dies ist dann gleichbedeutend mit der Abschaltung von CC2420, denn die Ausgangsspannung an `VREG_OUT` betreibt den CC2420.

🗐 *Das detaillierte Elektroschema zum Studium des gesamten Ablaufs ist im Anhang 2 beigelegt. Die Beschreibung zum CC2420 ist in [05] nachlesbar.*

B) Partie Magnetsensoren

Das folgende Schema zeigt die Funktionsweise der Magnetsensoren auf. Das Prinzip ist für beide Sensoren, d.h. für den Sensor für die Z-Achse, sowie wie für den 2-Achsen-Sensor für die X- und Y-Achse, das Selbe.

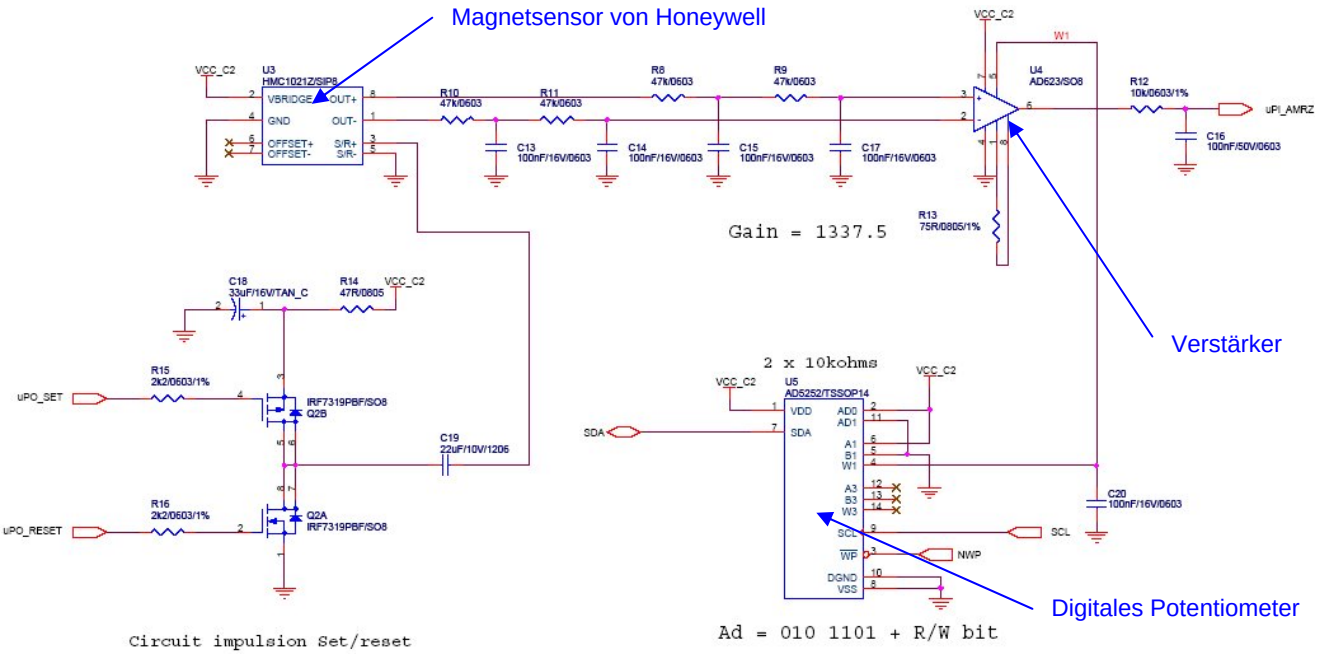


Abb. 24 : Magnetsensor-Partie des Elektroschemas

Der Magnetsensor von Honeywell (http://www.honeywell.com) misst beim Erhalt einer Betriebsspannung das Magnetfeld in seiner unmittelbaren Umgebung und gibt es in Form einer analogen Spannung an seinen Pins `OUT+` und `OUT-` aus.

Da dieser Spannungswert sich im Millivolt-Bereich befindet, wird er mittels Verstärker verstärkt, so dass eine Spannung im Volt-Bereich erzielt wird. Die verstärkte Ausgangsspannung am Verstärker wird dann für die Verarbeitung bzw. Kalkulation des Erkennungsalgorithmus der Fahrzeuge benutzt. Mittels Analog-Digital Konverter (ADC) konvertiert der Mikrokontroller dieses Signal in einen digitalen Wert um, der den aktuellen Wert des Magnetfeldes indiziert.

Der Verstärker wird mit einem Offset versehen, welcher von einem digitalen Potentiometer bestimmt wird. Das Potentiometer wird so eingestellt, dass der Offset für den Verstärker so eingestellt ist, dass das verstärkte Sensorsignal schlussendlich bei einem „ruhigen“ Umfeld, d.h. bei einem Umfeld ohne Fahrzeug in unmittelbarer Umgebung (Messbereich), schön in der Mitte des Spannungsbereichs liegt.

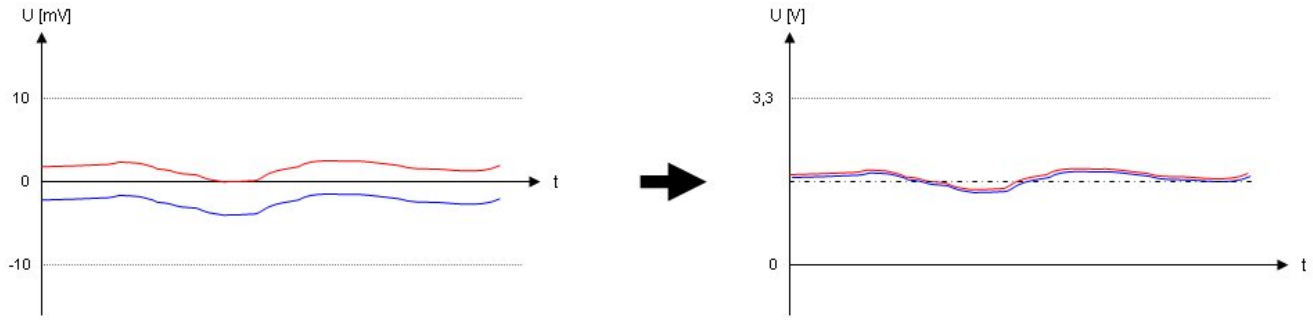


Abb. 25 : Verstärkung Sensorsignale

Das digitale Potentiometer wird über den I^2C-Bus konfiguriert.

Von Zeit zu Zeit ist es notwendig den Magnetsensor nachzustellen. Dies wird mit einem Set/Reset-Impuls gemacht. Mit einem Set-Impuls wird ein Stromimpuls in eine Richtung auf den Sensor geschickt. Mit dem Reset-Impuls wird dann auf die andere Richtung ein Stromimpuls geschickt. Dies hat zum Zweck, dass die Sensorelemente wieder geordnet ausgerichtet werden.

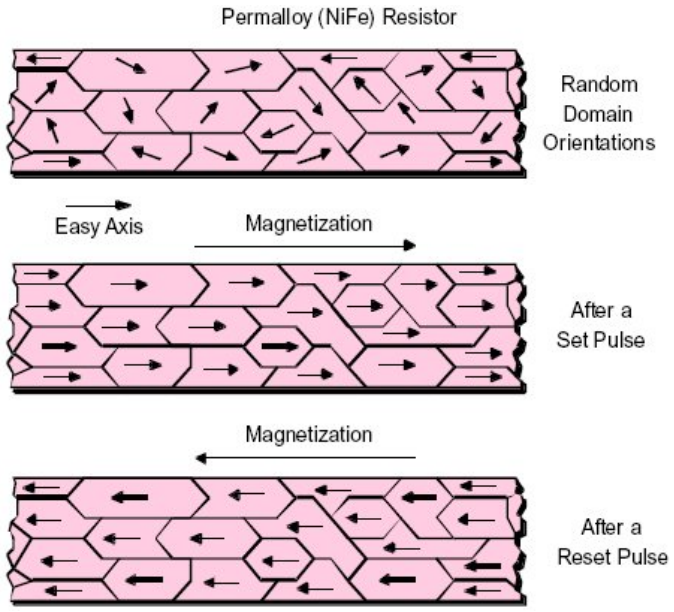


Abb. 26 : Set/Reset Vorgang des Magnetsensors

Das detaillierte Elektroschema zum Studium des gesamten Ablaufs für beide Magnetsensoren ist im Anhang 2 beigelegt.

C) Ladungspumpe

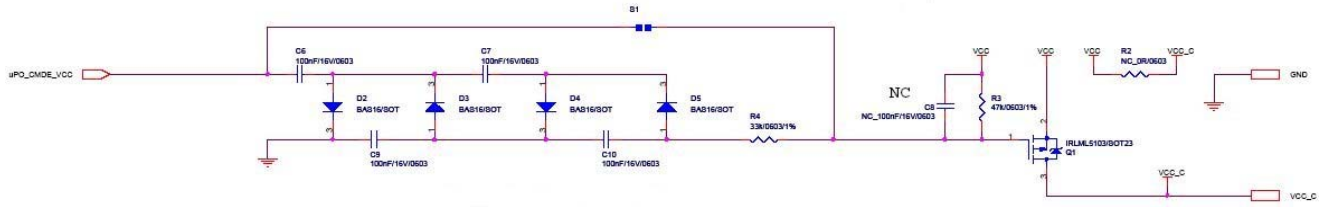


Abb. 27 : Ladungspumpe

Um einen möglichst geringen Energiekonsum zu gewährleisten, wurde eine Ladungspumpe hinzugefügt, die die Spannung für die Komponenten der Magnetsensoren liefert. Mittels PWM-Signal wird die Ladungspumpe „gefüttert“. Um die Komponenten der Magnetsensoren bei Nichtbenutzung auszuschalten, genügt es, das PWM-Signal zu unterbrechen.

D) I²C - Switch

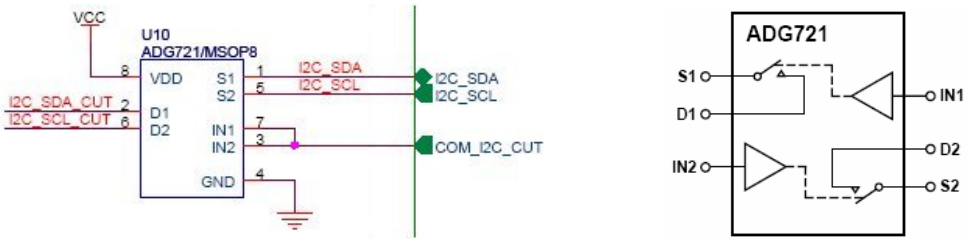


Abb. 29 : Switch für I²C-Signal

Um sicher zu gehen, damit das digitale Potentiometer bei dessen Ausschaltung auch wirklich ausgeschaltet wird und nicht über die I²C-Pins noch Kurzgeschlossen wird, wurde als reine Vorsichtsmassnahme ein analoger Schalter zur Abklemmung der I²C-Signale eingebaut.

Über den Pin `COM_I2C_CUT` des Mikrokontrollers kann nun bestimmt werden, ob die I2C-Signale aktiv sind oder nicht. D.h. ob sie bis zu den Potentiometern durchkommen oder nicht.

E) Mikrokontroller

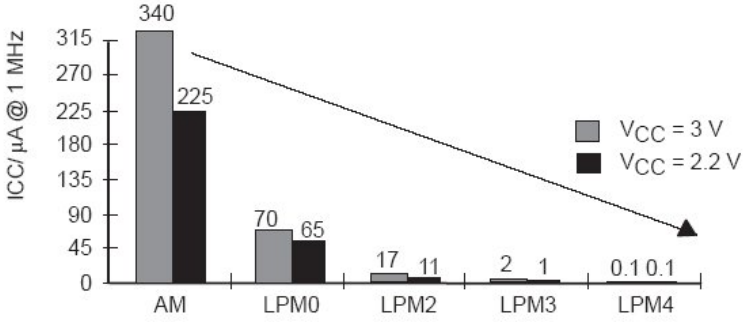


Abb. 28 : Betriebsmodi des Mikrokontrollers

Der Mikrokontroller eignet sich ausgezeichnet für Anwendung mit geringer Energiekonsumation, weil er in insgesamt sechs verschiedene Modi gesetzt werden kann. Neben dem Aktivmodus (AM) kann er in fünf verschiedene Low-Power-Modi (LPMx) geschaltet werden, damit weniger Energie verbraucht wird.

✋ *Für mehr Informationen siehe hierzu auch Kapitel 4.3 oder [03] und [04]*

4.2.3 Die Installation

Die Installation vor Ort sieht wie folgt aus:

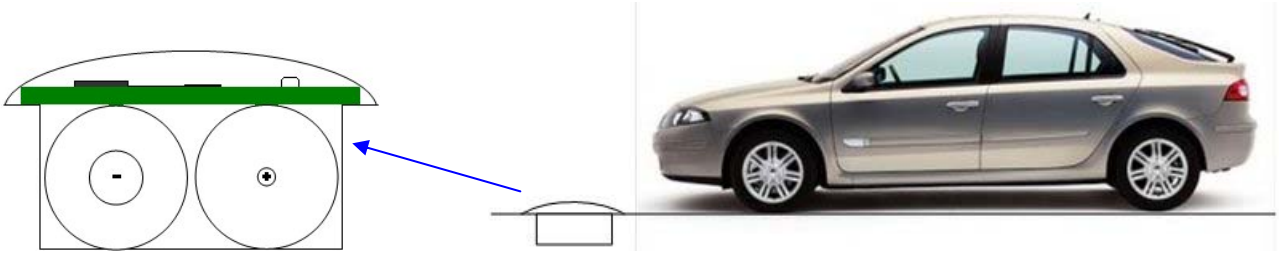


Abb. 29 : Installationsplan des Sensors

Bei jedem Parkplatz wird ein Loch in den Boden gemacht, um den CEP hineinzusetzen.

Der Aufbau sieht so aus, dass zu unterst zwei Batterien des Typs D (http://www.varta.com) platziert sind. Sie haben eine elektrische Ladung von 16,5 Ah.
Darüber kommt die Hardwareplatine installiert und zu guter letzt wird das Ganze mit einem stabilen Deckel abgedeckt.

Abb. 30 : Installationsaufbau

4.3 Software Design

Der folgende Abschnitt dokumentiert die verschiedenen Aspekte der erstellten Software. Dies ist der grundlegende Teil dieser Diplomarbeit.

4.3.1 Der Mikrokontroller MSP430F169

Wie weiter oben bereits mehrfach erwähnt wurde als Mikrokontroller der MSP430F169 von Texas Instruments eingesetzt. In diesem Abschnitt werden kurz die wichtigsten Eigenschaften dieses Mikrokontrollers erläutert.

Kenndaten

Als Kenndaten gibt es folgende Aspekte zu erwähnen:

- 16-Bit RISC CPU Architektur
- Tiefer Netzspannungsbereich : 1.8 V bis 3.6 V
- 6 Betriebsmodi mit geringer Energiekonsumation
- 2 serielle Kommunikationsschnittstellen
  - o USART0
    Asynchron : UART
    Synchron : I^2C oder SPI
  - o USART1
    Asynchron : UART
    Synchron : SPI
- Flexibles Clock System
  - o MCLK
  - o SMCLK
  - o ACLK
- 2 16-Bit Timer
  - o Timer A
  - o Timer B
- 12-Bit Analog-Digital Konverter
- 12-Bit Digital-Analog Konverter
- Watchdog Timer
- Flash Memory Kontroller
- DMA Kontroller
- Netzspannungsüberwachung
- 6 digitale I/O Ports
- Spannungsvergleicher
- …

Betriebsmodi

Wie bereits erwähnt, hat der Mikroprozessor 6 verschiedene Betriebsmodi. Einen Aktiv Modus (AM) und fünf Low-Power-Modi (LPM0-4). Durch einen Interrupt-Event kann der Mikroprozessor vom Low-Power-Modus wieder in den Aktiv Modus zurückgeschaltet werden.

Folgend sind die verschiedenen Modi und deren Eigenschaften aufgeführt:

Active Mode (AM)	**Low-Power Mode 0/1** (LPM0 u. LPM1)	**Low-Power Mode 2** (LPM2)	**Low-Power Mode 3** (LPM3)	**Low-Power Mode 4** (LPM4)
Alle Clocks sind aktiv	- MCLK deaktiviert - SMCLK aktiv - ACLK aktiv (- DCO deaktiviert)	- MCLK deaktiviert - SMCLK deaktiviert - ACLK aktiv - DCO aktiv	- MCLK deaktiviert - SMCLK deaktiviert - ACLK aktiv - DCO deaktiviert	- MCLK deaktiviert - SMCLK deaktiviert - ACLK deaktiviert - DCO deaktiviert - Oszillator gestoppt

Pinnbelegung

Die Belegung der einzelnen Pins des MSPF169 ist wie folgt definiert:

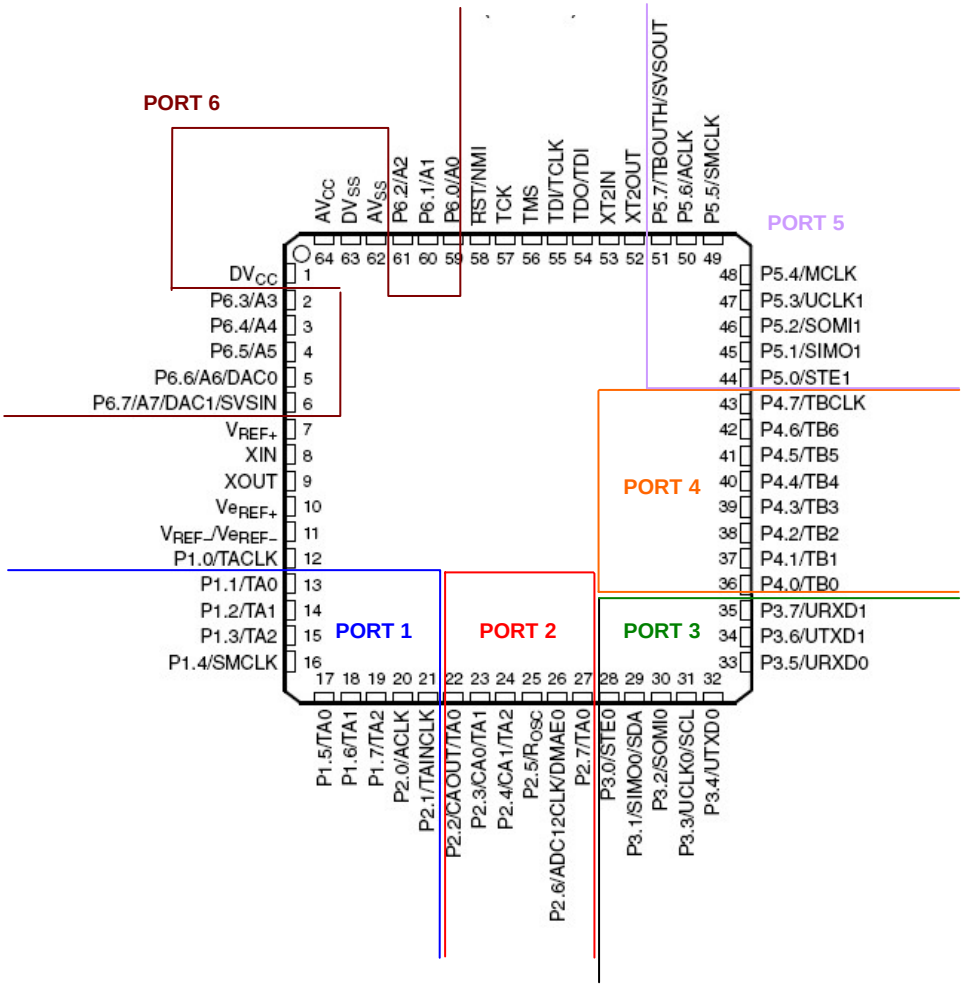


Abb. 31 : Pinnbelegung des MSP430F169

Für die exakte Belegung der Hardwarekomponente siehe das Elektroschema im Anhang 2 oder für mehr Informationen betreffend Funktionalität siehe [03] und [04]

4.3.2 Projektentwicklung

Die ursprüngliche Projektidee zu Beginn dieser Diplomarbeit war die folgende:

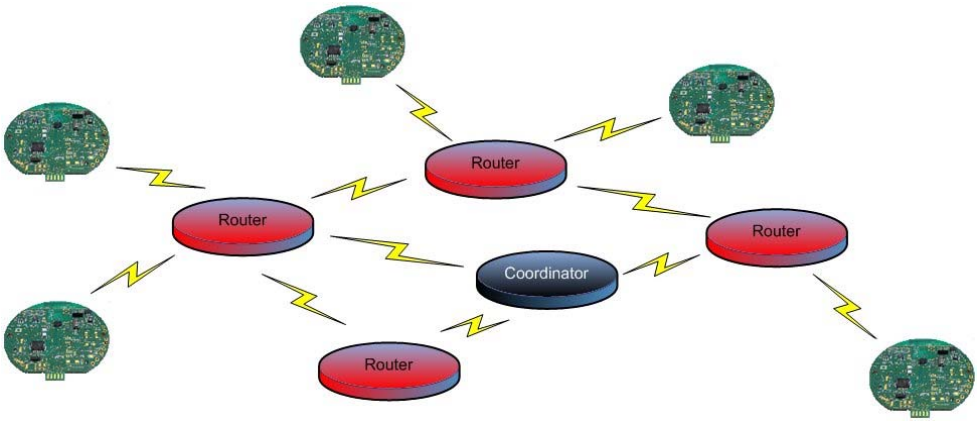


Abb. 32 : Idee der ursprünglichen Realisierungsform (ZigBee)

Der ZigBee Standard befindet sich aber zurzeit noch im Entwicklungsstadium und es ist daher relativ schwierig geeignete Komponenten zu finden. Online sind zwar einige Komponenten auffindbar, doch sind diese entweder noch gar nicht, nur im Asiatischen Raum oder nur in Amerika verfügbar. Beispielsweise hat sich ergeben, dass die „XBee Pro“-Module der Serie 1, welche in unserem Fall erstmals als Koordinator eingesetzt werden sollen, nur den Standard IEEE 802.15.4 unterstützen und erst ab Serie 2 auch den Standard ZigBee unterstützen würden. Leider sind die Serie 2 Modelle aber noch nicht erhältlich.

Des Weiteren wurde einige Zeit investiert, um nach bereits bestehenden ZigBee-Stacks zu suchen und gegebenenfalls diese zu implementieren. Es wurde von TI bereits ein spezifischer ZigBee-Stack erstellt, der für Mikroprozessoren des Typs MSP430F mit dem RF-Transceiver CC2420 gedacht ist. Die Anwendung/Anpassung dieses Stacks zu unserer Hardware erwies sich aber als unmöglich. Die Ressourcen des MSP430F169 sind zu klein in Anbetracht des kompilierten Stacks:

Gerätetyp	Grösse Flash	Grösse RAM	MSP430F169
Koordinator	50 kBytes	4.3 kByte	Flash : 60 kByte RAM : 2 kByte
Router	49 kBytes	4.3 kByte	
Endgerät	38 kBytes	3.6 kByte	

Weitere Stacks wurden analysiert, alle jedoch ohne gewünschtes Resultat.

Eine entsprechende Realisierung auf dem Standard ZigBee bedarf also zurzeit noch eines relativ grossen Zeitaufwands. Da aber diese entsprechende Zeit nicht zur Verfügung steht, wurde nun für eine erste Version die Idee der ursprünglichen Realisierungsform auf folgendes System vereinfacht:

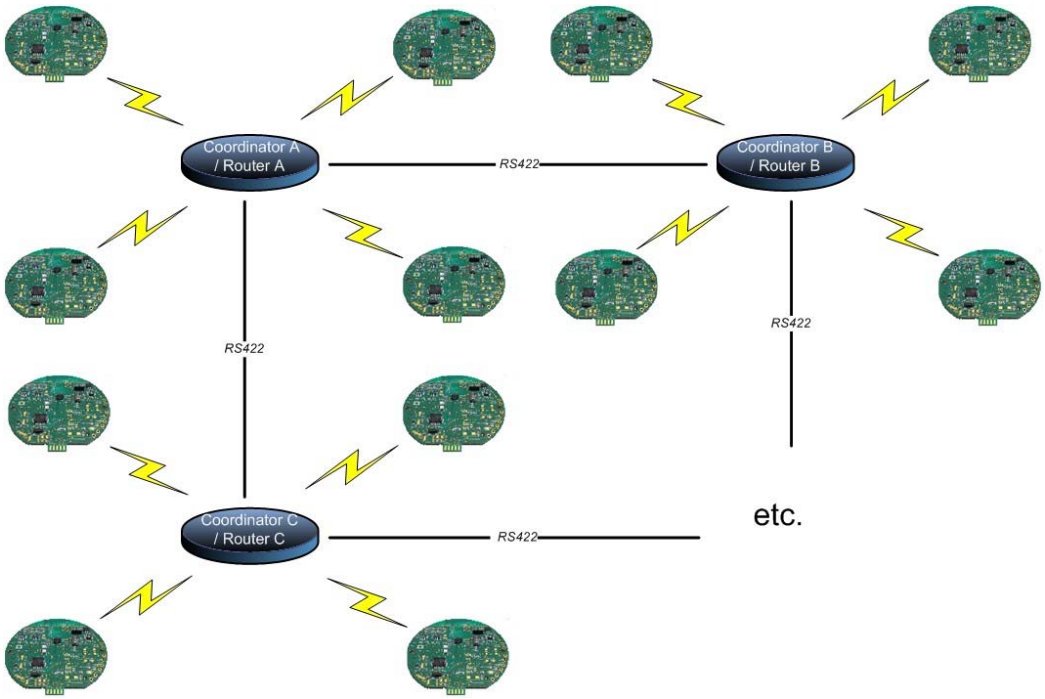


Abb. 32 : Idee der neuen Realisierungsform (IEEE 802.15.4)

4.3.3 Klassen

In diesem Abschnitt wird aufgezeigt, welche Klassen erzeugt oder übernommen wurden, um eine funktionsfähige Anwendung zu erreichen.

4.3.3.1 Clock

Hier wird die Klasse „clock“ genauer erklärt.

Beschrieb

Diese Klasse definiert das „Basic Clocking Sub-system“ für den MSP430. D.h. hier werden die Frequenzen der drei Clocks MCLK, SMCLK und ACLK definiert.

Die Clocks sollen auf folgende Frequenzen eingestellt werden:

MCLK : 4 MHz
SMCLK : 1 MHz
ACLK : 32768 kHz

Funktionen

- `void clock_init()`
  (Initialisierung der Basic Clocks)

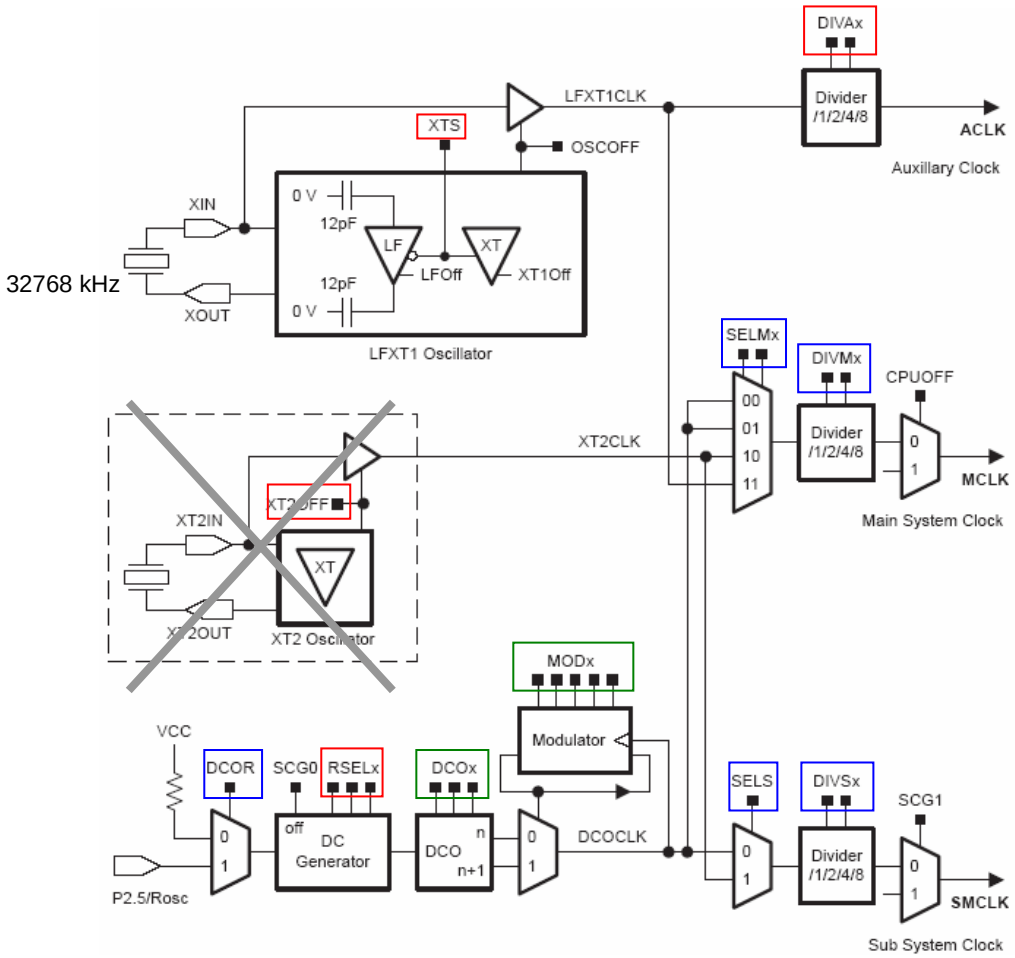


Abb. 33 : Basic Clock Blockdiagramm

DCOCTL =

7	6	5	4	3	2	1	0
DCOx				MODx			
rw-0	rw-1	rw-1	rw-0	rw-0	rw-0	rw-0	rw-0

(DCO2 + DCO0 + MOD4 + MOD2)

BCSCTL1 =

7	6	5	4	3	2	1	0
XT2OFF	XTS	DIVAx		XT5V	RSELx		
rw-(1)	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-1	rw-0	rw-0

(XT2OFF + RSEL2 + RSEL1 + RSEL0)

BCSCTL2 =

7	6	5	4	3	2	1	0
SELMx		DIVMx		SELS	DIVSx		DCOR
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-0	rw-0	rw-0

(0x04)

Für eine genauere Beschreibung der einzelnen Register des Basic Clock Moduls siehe [03]

4.3.3.2 SPI

Hier wird die Klasse „spi“ genauer erklärt.

Beschrieb

Diese Klasse verwaltet die SPI-Kommunikation zwischen dem Mikrokontroller und dem RF-Transceiver CC2420.

Anmerkung : Da für die digitalen Potentiometer der I2C-Bus benötigt wird und dieser nur auf der seriellen Kommunikationsschnittstelle USART0 zur Verfügung steht, wurde USART1 für die SPI Kommunikation gewählt.

Funktionen

- `void spi_init(void)`
  (Initialisierung von SPI auf der seriellen Kommunikationsschnittstelle USART1)
- `void spi_send(char)`
  (Versenden eines Charakters – Setzen eines Charakters auf den TX Buffer)
- `char spi_receive(void)`
  (Lesen eines Charakters vom RX Buffer)

Für eine genauere Beschreibung der einzelnen Register des USART1 siehe [03].
Eine kurze Beschreibung zum SPI-Bus ist im Anhang 1 beigefügt.

4.3.3.3 CC2420

Hier wird die Klasse „cc2420“ genauer erklärt.

Beschrieb

Diese Klasse definiert die Schnittstelle zum RF-Transceiver CC2420 von Chipcon und ist aufbauend auf der Klasse „spi“ (*Kapitel 4.3.3.2*).

Funktionen

- `void cc2420_init(void)`
  (Initialisierung des Mikrokontrollers für die Nutzung des CC2420 RF-Transceivers)
- `void cc2420_set_reg(uint8_t addr, uint16_t data)`
  (Schreiben von Daten in ein Register des CC2420)
- `uint16_t cc2420_get_reg(uint8_t addr)`
  (Auslesen eines Registerinhalts des CC2420)
- `uint8_t cc2420_get_status(void)`
  (Liest den aktuellen Status des CC2420)
- `void cc2420_write_fifo(uint8_t len, uint8_t * data)`
  (Schreiben in den TX-Buffer – Versand eines RF-Datenpakets)
- `void cc2420_read_fifo(uint8_t len, uint8_t * data)`
  (Lesen des RX-Buffers – Erhalt eines RF-Datenpakets)
- `void cc2420_strobe(uint8_t cmd)`
  (Aufrufen eines Stroberegisters [Befehlsregister])

- `void cc2420_set_reg_ram(uint8_t addr, uint8_t len, uint16_t * data)`
  (Schreiben in den RAM des CC2420)
- `void cc2420_get_reg_ram(uint8_t addr, uint8_t len, uint16_t * data)`
  (Lesen vom RAM des CC2420)
- *`interrupt(PORT1_VECTOR) port1_isr(void)`*
  (Interrupt für Erhalt eines RF-Datenpakets)

Für eine genauere Beschreibung des CC2420 siehe [05]

4.3.3.4 RF2

Hier wird die Klasse „rf2“ genauer erklärt.

Beschrieb

Diese Klasse verwaltet die Nutzung des RF-Moduls über den CC2420 von Chipcon. Sie ist aufbauend auf der Klasse „cc2420“ (*Kapitel 4.3.3.3*).

Funktionen

- `void rf2_init(void)`
  (Initialisierung des CC2420 RF-Transceivers)
- `void rf2_power_on(void)`
  (Einschalten des CC2420)
- `void rf2_power_off(void)`
  (Ausschalten des CC2420)
- `void rf2_sleep(void)`
  (CC2420 schlafen legen – Oszillator stoppen)
- `void rf2_idle(void)`
  (CC2420 aufwecken – Oszillator starten)
- `uint8_t rf2_send(uint8_t len)`
  (Schreiben von Daten in den RX Buffer des CC2420 – Senden eines RF-Rahmens)
- `void rf2_receive_on(void)`
  (CC2420 in Empfangsmodus setzen)
- `void rf2_receive_off(void)`
  (CC2420 von Empfangsmodus zurücksetzen)
- *`static void rf2_set_channel(uint8_t channel)`*
  (Sendekanal (Frequenzband im 2.4 GHz - Bereich) wählen bzw. setzen)

Für eine genauere Beschreibung der einzelnen Register des CC2420 siehe [05]

4.3.3.5 Mac_send

Hier wird die Klasse „mac_send“ genauer erklärt.

Beschrieb

Diese Klasse stellt die korrekten MAC-Layer Daten-Rahmen (nach IEEE 802.15.4 Standard) zusammen, um Daten über den RF-Transceiver zu versenden. Sie ist aufbauend auf der Klasse „rf“ (*Kapitel 4.3.3.4*).

Funktionen

- `void mac_send(uint8_t event)`
  (Sendet eine IEEE 802.15.4 MAC-Layer Daten-Rahmen über den CC2420 RF-Transceiver)
- `static void mac_send_set_state(uint8_t * buf)`
  (Stellt ein Daten-Rahmen mit Parkplatz-Nummer und aktuellem Zustand zusammen)
- `static void mac_send_set_vizgraph(uint8_t buf, uint8_t x, uint8_t y, uint8_t z)`
  (Stellte ein Daten-Rahmen mit Parkplatz-Nummer, aktuellem Zustand und den aktuellen Sensorwerten in x-, y- und z-Richtung zusammen)

Für eine genauere Beschreibung der Zusammensetzung eines MAC-Layer Daten-Rahmens siehe [01]

4.3.3.6 Rs232

Hier wird die Klasse „rs232“ genauer erklärt.

Beschrieb

Diese Klasse verwaltet die RS232 Kommunikation zwischen dem Mikrokontroller und einem externen seriellen Empfänger wie beispielsweise dem PC.

Anmerkung : Da die SPI Kommunikation schon über die serielle Kommunikationsschnittstelle USART1 Läuft, wurde USART0 für die RS232 Kommunikation gewählt. Es gilt aber zu beachten, dass der I2C Bus für die Kommunikation mit den digitalen Potentiometern auch auf dem USART0 läuft und somit diese Schnittstelle geteilt werden muss.

Diese Klasse wurde übernommen und nach unserem Gebrauch abgeändert.

Funktionen

- `void rs232_init(void)`
  (Initialisierung von RS232 auf der seriellen Kommunikationsschnittstelle USART0)
- `void rs232_putc(char c)`
  (Versenden eines Charakters – Setzen eines Charakters auf den TX Buffer)
- `void rs232_printf(const char * fmt, ...)`
  (Versenden einer Nachricht (Charakter-Array, String) in Anlehnung an die Methode „rs232_putc()“)
- *`static void rs232_getEvent(void)`*
  (Kalkulierung und Ausführung eines Events aufgerufen durch einen Interrupt (Erhalt von Daten über RS232))
- *`interrupt (USART0RX_VECTOR) rs232_isr(void)`*
  (Interrupt für Erhalt eines Charakter über RS232 (USART0))

Für eine genauere Beschreibung der einzelnen Register des USART0 siehe [03]

4.3.3.7 VtxPrintf

Hier wird die Klasse „vtxprintf“ genauer erklärt.

Beschrieb

Diese Klasse stellt eine abgespeckte Version der Funktion „printf()“ dar, welche in der Standard I/O (stdio.h) Bibliothek vorhanden ist. Sie ist speziell auf eingebettete Systeme zugeschnitten.

Anmerkung : *Diese Klasse wurde übernommen.*

Funktionen

- `void vtxprintf(void (* tx_char)(char), const char * fmt, va_list ap)`
  (Stellt das gewünschte Ausgabeformat des Charakter-Arrays bzw. Strings zusammen)

4.3.3.8 RavelFuncLib

Hier wird die Klasse „ravelfunclib“ genauer erklärt.

Beschrieb

Diese Klasse beinhaltet verschiedene Funktionen, die allgemein nützlich sind aber nicht einer konkreten Klasse zugehören.

Anmerkung : Die hier enthaltene Funktion ist für RS232-Ausgabetestzwecke erstellt worden.

Funktionen

- `uint8_t setAsciiToDecHexValue(uint8_t ascii)`
  (Berechnet aus einem Ascii-Hexadezimalzahlenwert den hexadezimalen Wert im Dezimalformat)

4.3.3.9 PWM

Hier wird die Klasse „pwm“ genauer erklärt.

Beschrieb

Diese Klasse verwaltet die Steuerung bzw. Erzeugung eines PWM Signals, welches für die Ladungspumpe (✋ *Kapitel 4.2.2 C*) verwendet wird. Hierfür wird der Timer A des Mikrokontrollers beansprucht.

Funktionen

- `void pwm_on(void)`
  (Startet die Erzeugung und Ausgabe eines PWM-Signals)
- `void pwm_off(void)`
  (Stoppt die Ausgabe des PWM-Signals)

🗗 *Für eine genauere Beschreibung der einzelnen Register des Timers A siehe [03]*

4.3.3.10 Sleep

Hier wird die Klasse „sleep“ genauer erklärt.

Beschrieb

Diese Klasse stellt den Mikroprozessor in einen „Schlafmodus“. D.h. wird der Mikroprozessor nur minimal benutzt kann er in einen Low-Power-Modus (☞ *Kapitel 4.3.1*) gesetzt werden. Des Weiteren werden die einzelnen Pins und Ports zurückgesetzt, d.h. soweit als möglich als Inputs definiert, da sie so weniger Energie verbrauchen.

Funktionen

- `void sleep(void)`
  (Setzt den Mikroprozessor in einen Modus wo er wenig Energie verbraucht)

4.3.3.11 I2C

Hier wird die Klasse „i2c“ genauer erklärt.

Beschrieb

Diese Klasse verwaltet die I2C Kommunikation zwischen dem Mikrokontroller und den digitalen Potentiometern und dem seriellen EEPROM.

Anmerkung : Eine I2C Kommunikation ist nur über die serielle Kommunikationsschnittstelle USART0 Möglich. Daher wurde USART0 gewählt.
Die RS232 Kommunikation benutzt aber ebenfalls den USART0, da der USART1 schon vom SPI besetzt wird. Diese Schnittstelle wird also zwischen I2C und RS232 geteilt.

Diese Klasse wurde übernommen und nach unserem Gebrauch abgeändert.

Funktionen

- `void i2c_init(void)`
  (Initialisierung von I2C auf der seriellen Kommunikationsschnittstelle USART0)
- `void i2c_write(uint8_t addr, uint8_t n, ...)`
  (Versenden von Charakteren über den I2C-Bus auf einen gewünschten Empfänger)

🗗 *Für eine genauere Beschreibung der einzelnen Register des USART0 siehe [03]*
🗗 *Die genaue Definition ist in [06] nachzulesen*

4.3.3.12 AMR_Autoconf

Hier wird die Klasse „amr_autoconf“ genauer erklärt.

Beschrieb

Diese Klasse ist für die Konfiguration des Offsets des Verstärkers für das Sensorsignal verantwortlich. Die digitalen Potentiometer werden mit verschiedenen Offsets versehen und so nach dem idealen Mittelwert der Ausgabe abgetastet. Einmal gefunden werden die Potentiometer mit diesem gefunden Wert programmiert.

Anmerkung : *Diese Klasse wurde übernommen.*

Funktionen

- `void amr_autoconf(void)`
  (Kofiguriert bzw. sucht und definiert das ideale Offset für die digitalen Potentiometer)
- *`static void amr_scan_offset(void)`*
  (Durchläuft den Offsetbereich der Potentiometer zum Suchen des idealen Offsets)

Für eine genauere Beschreibung der digitalen Potentiometer siehe [07]

4.3.3.13 AMR

Hier wird die Klasse „amr“ genauer erklärt.

Beschrieb

Diese Klasse verwaltet den Zugriff auf die Magnetsensoren bzw. die Analog-Digital Konverter.

Anmerkung : *Diese Klasse wurde übernommen.*

Funktionen

- `void amr_init(void)`
  (Initialisiert den 12-Bit Analog-Digital Konverter zur Benutzung der Magnetsensoren)
- `void amr_set_x_ofs(uint8_t i)`
  (Schreibt bzw. setzt einen Offsetwert auf das digitale Potentiometer für die x-Achse)
- `void amr_set_y_ofs(uint8_t i)`
  (Schreibt bzw. setzt einen Offsetwert auf das digitale Potentiometer für die y-Achse)
- `void amr_set_z_ofs(uint8_t i)`
  (Schreibt bzw. setzt einen Offsetwert auf das digitale Potentiometer für die z-Achse)
- `uint8_t amr_get_x_ofs(void)`
  (Gibt den aktuellen Offsetwert des Potentiometers für die x-Achse zurück)
- `uint8_t amr_get_y_ofs(void)`
  (Gibt den aktuellen Offsetwert des Potentiometers für die y-Achse zurück)
- `uint8_t amr_get_z_ofs(void)`
  (Gibt den aktuellen Offsetwert des Potentiometers für die z-Achse zurück)
- `void amr_degauss(void)`
  (Erzeugt eine Set/Reset-Impuls für die Magnetsensoren)

- `void amr_start(void)`
  (Startet die Abtastung des Analog-Digital Konverters zum Erhalt der aktuellen Werte in Anlehnung an den Timer B des Mikrokontrollers)
- `void amr_stop(void)`
  (Stoppt die Abtastung des Analog-Digital Konverters – Ausschaltung Timer B)
- `interrupt (TIMERB0_VECTOR) timerb_isr(void)`
  (Interrupt bei Überlauf des Timers B)
- `interrupt (ADC12_VECTOR) adc12_isr(void)`
  (Interrupt des Analog-Digital Konverters - Konvertierung des analogen Signals)

Für eine genauere Beschreibung des Analog-Digital Konverters siehe [03]

4.3.3.14 Czame

Hier wird die Klasse „czame“ genauer erklärt.

Beschrieb

Diese Klasse stellt den Algorithmus dar, der aus den Abtastwerten des Analog-Digital Konverters die Präsenz eines Fahrzeugs berechnet.

Oberflächlich und einfach erklärt, stützt sich dieser Algorithmus auf zwei Parameter. Der erste wird durch die Geschwindigkeit der Signaländerung definiert und der zweite stützt sich auf das Ingegral des Signals. Je steiler die Signaländerung und je grösser das Integral sind, desto schneller indiziert der Algorithmus dass ein Fahrzeug präsent ist.
Eine Callback-Funktion zeigt dann eine berechnete Zustandsänderung an.

Anmerkung : *Diese Klasse wurde übernommen.*

Funktionen

- `void czame_config(const char * str)`
  (Initialisiert die Parameter (Zeichenkette) des Algorithmus)
- `void czame_init(uint8_t i)`
  (Initialisation der Variablen des Algorithmus)
- `void czame_play(uint16_t x, uint16_t y, uint16_t z)`
  (Fütterung des Algorithmus mit Zustandswerten)
- *`static void clear(uint8_t * tgt)`*
  (Initialisierung bzw. Rücksetzung des Zustandvektors bzw. -arrays)
- *`static void copy(uint8_t * tgt, const uint8_t * src)`*
  (Führt eine Kopie das Zustandvektors bzw. -arrays durch)
- *`static void forward(uint8_t * s, const uint8_t * e, uint8_t obsrv)`*
  (Zustandsänderung vorwärts in Zustand „vielleicht ein Fahrzeug“)
- *`static void backward(const uint8_t *e, uint8_t best)`*
  (Zustandsänderung rückwärts in Zustand „vielleicht ein Fahrzeug“)
- *`static void viterbi(uint8_t obsrv)`*
  (Wiederherstellung des wahrscheinlichsten Zustands)
- *`static uint16_t isqrt(uint32_t x)`*
  (Berechnung der Quadratwurzel)
- *`static uint16_t length(uint16_t x, uint16_t y, uint16_t z)`*
  (Berechnung der Norm eines Vektors)

4.3.3.15 Variables

Hier wird die Klasse „variables“ genauer erklärt.

Beschrieb

Diese Klasse beinhaltet keine Funktionen. Hier werden alle globalen Variabeln definiert und initialisiert.

Funktionen

- `keine`
  (-)

4.3.3.16 Definitions

Hier wird die Klasse „definitions“ genauer erklärt.

Beschrieb

Diese Klasse beinhaltet keine Funktionen. Hier werden alle globalen Definitionen und Makros definiert.

Funktionen

- `keine`
  (-)

4.3.3.17 Main

Hier wird die Klasse „main“ genauer erklärt.

Beschrieb

Diese Klasse ist die Hauptklasse der gesamten Anwendung. Von hier aus wird alles initialisiert und gesteuert.

Funktionen

- `int main(void)`
  (Main-Funktion : Leitfaden der Anwendung)
- `void init_uc(void)`
  (Initialisierung des Mikrokontrollers)
- `void notify(uint16_t report)`
  (Callback-Funktion des Fahrzeugserkennungsalgorithmus)

Der gesamte Quellcode ist im Anhang 3 als Quellcode beigelegt.

4.3.4 Versionen

Es wurde eine Anwendung für die vorhandene Hardware für folgende drei Typen erstellt:

- Koordinator
- Router
- Endgerät

Endgerät (Sender)

Das Endgerät ist die eigentliche Hauptaufgabe.
Seine Aufgaben sind die Abtastung des Magnetfelds der Umgebung, die Berechnung der Präsenz eines Fahrzeugs mittels dieser Abtastwerte und schlussendlich der Versand des aktuellen Zustands via ZigBee-Rahmen an einen Koordinator/Router.

Folgende Klassen wurden für die Umsetzung dieser Version benutzt:

- amr (☞ *Kapitel 4.3.3.13*)
- amr_autoconf (☞ *Kapitel 4.3.3.12*)
- cc2420 (☞ *Kapitel 4.3.3.3*)
- clock (☞ *Kapitel 4.3.3.1*)
- czame (☞ *Kapitel 4.3.3.14*)
- definitions (☞ *Kapitel 4.3.3.16*)
- i2c (☞ *Kapitel 4.3.3.11*)
- mac_send (☞ *Kapitel 4.3.3.5*)
- main (☞ *Kapitel 4.3.3.17*)
- pwm (☞ *Kapitel 4.3.3.9*)
- rf2 (☞ *Kapitel 4.3.3.4*)
- sleep (☞ *Kapitel 4.3.3.10*)
- spi (☞ *Kapitel 4.3.3.2*)
- variables (☞ *Kapitel 4.3.3.15*)

Koordinator (Empfänger)

Der Koordinator wurde für Testzwecke erstellt und ersetzt/unterstützt das eingekaufte Modul XBee Pro von Maxstream Serie 1.
Seine Aufgabe ist der der Empfang der ZigBee-Rahmen der Endgeräte und Router und die anschliessende Ausgabe dieser Daten über die RS232-Schnittstelle an einen PC. Der Koordinator wurde aber nicht auf einen geringen Energieverbrauch konzipiert.

Folgende Klassen wurden für die Umsetzung dieser Version benutzt:

- cc2420 (☞ *Kapitel 4.3.3.3*)
- definitions (☞ *Kapitel 4.3.3.16*)
- main (☞ *Kapitel 4.3.3.17*)
- ravelfunclib (☞ *Kapitel 4.3.3.8*)
- rf2 (☞ *Kapitel 4.3.3.4*)
- rs232 (☞ *Kapitel 4.3.3.6*)
- spi (☞ *Kapitel 4.3.3.2*)
- variables (☞ *Kapitel 4.3.3.15*)
- vtxprintf (☞ *Kapitel 4.3.3.7*)

Router

Der Router wurde anschliessend für weitere Testzwecke erstellt.
Seine Aufgabe ist ebenfalls der Empfang der ZigBee-Rahmen der Endgeräte und Router, doch dieser gibt die empfangenen Daten anschliessend nicht per RS232 aus, sondern schickt sie in einem neuen ZigBee-Rahmen an den Koordinator weiter. Seine Aufgabe ist also die Weiterleitung von ZigBee-Rahmen. So wird die Reichweite zwischen Endgerät und Router erhöht. Der Router wurde ebenfalls nicht auf einen geringen Energieverbrauch konzipiert.

Folgende Klassen wurden für die Umsetzung dieser Version benutzt:

- cc2420 (☞ *Kapitel 4.3.3.3*)
- definitions (☞ *Kapitel 4.3.3.16*)
- mac_send (☞ *Kapitel 4.3.3.5*)
- main (☞ *Kapitel 4.3.3.17*)
- rf2 (☞ *Kapitel 4.3.3.4*)
- spi (☞ *Kapitel 4.3.3.2*)
- variables (☞ *Kapitel 4.3.3.15*)

5. Validierung und Tests

Dieser Abschnitt widmet sich der Validation der Anwendungen anhand einiger Tests und Versuchen.

5.1 Testprogramm

Eigens Testzwecken wurde mit dem Programm „C Sharp“ ein Testprogramm geschrieben, welches eine einfache Anzeige der aktuellen Sensor und Zustandswerte ausgibt.

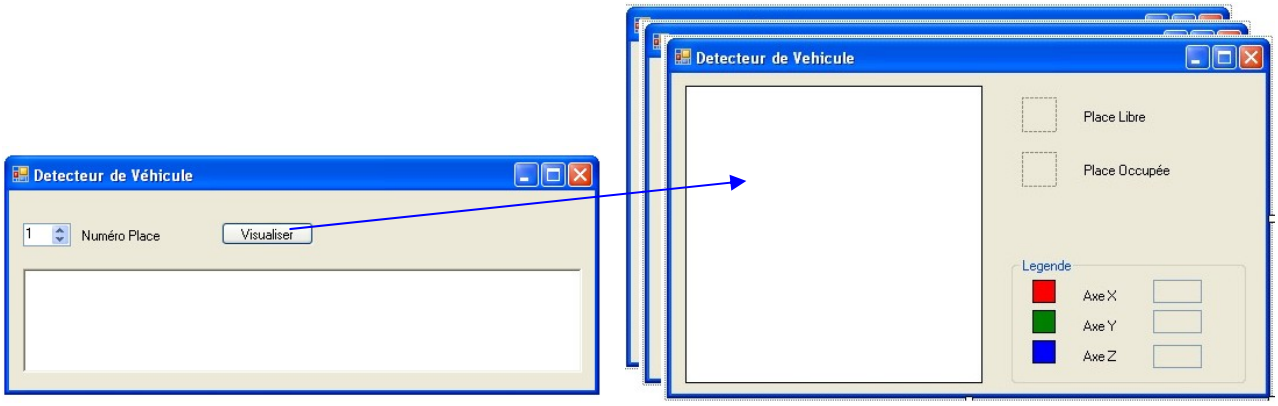


Abb. 35 : Grafische Oberfläche des Testprogramms (Version 3)

Dieses Testprogramm wird mit den Ausgabewerten des Koordinators mittels einer RS232 Verbindung gefüttert. Die RS232-Datenpakete wurden wie folgt definiert:

0xA1	0x01	0x00	0x80	0x80	0x80
Frame-format	parking-id	state	x-value	y-value	z-value

Beschreibung

Frame-format :	Zeigt den Start eines neuen Rahmens an	[konstanter Wert : 0xA1]	
Parking-id :	Parkplatz Nummer	[0 - 255]	
State :	Aktueller Zustand	[0 : frei - 1 : besetzt]	
x-value :	Sensorwert der X-Achse	[0 - 255]	Reset-Wert 128
y-value :	Sensorwert der Y-Achse	[0 - 255]	
z-value :	Sensorwert der Z-Achse	[0 - 255]	

Abb. 36 : RS232-Testprotokoll

Der grosse Vorteil dieses Testprogramms ist seine Reichweite. Es kann einerseits der Koordinator getestet werden, d.h. ob er alle Datenpaket erhält und diese dann auch richtig ausgibt. Es können die Sensoren getestet werden. Die aktuellen Sensorwerte werden grafisch angezeigt und man erkennt relativ schnell Kuriositäten.

Wie in Abbildung 35 ersichtlich ist, besteht das Testprogramm aus 2 Hauptfenstern. Das kleinere Fenster (links) zeigt in einer Liste die aufgetretenen Zustandsänderungen auf, wie ein Log-File. Hier kann für jeden Sensor ein separates Ausgabefenster (rechts) erzeugt werden, indem die Sensornummer gewählt wird und auf den Knopf „Visualiser“ gedrückt wird.

5.2 Praktische Tests

Die Tests in einer praktischen Umgebung wurden auf zwei verschiedene Arten ausgeführt.

- Simulation mittels magnetischer Komponenten im Labor
- Echtzeit-Simulation auf verschiedenen vorhandenen Parkplätzen

A) Simulation mittels magnetischer Komponenten im Labor

Bevor man sich an Echtzeit-Simulationen in wirklichen Einsatzgebieten vorwagt, bedarf es zuerst einer gründlichen Austestung der gesamten Anwendung im Labor bzw. beim Arbeitsplatz.

Schema

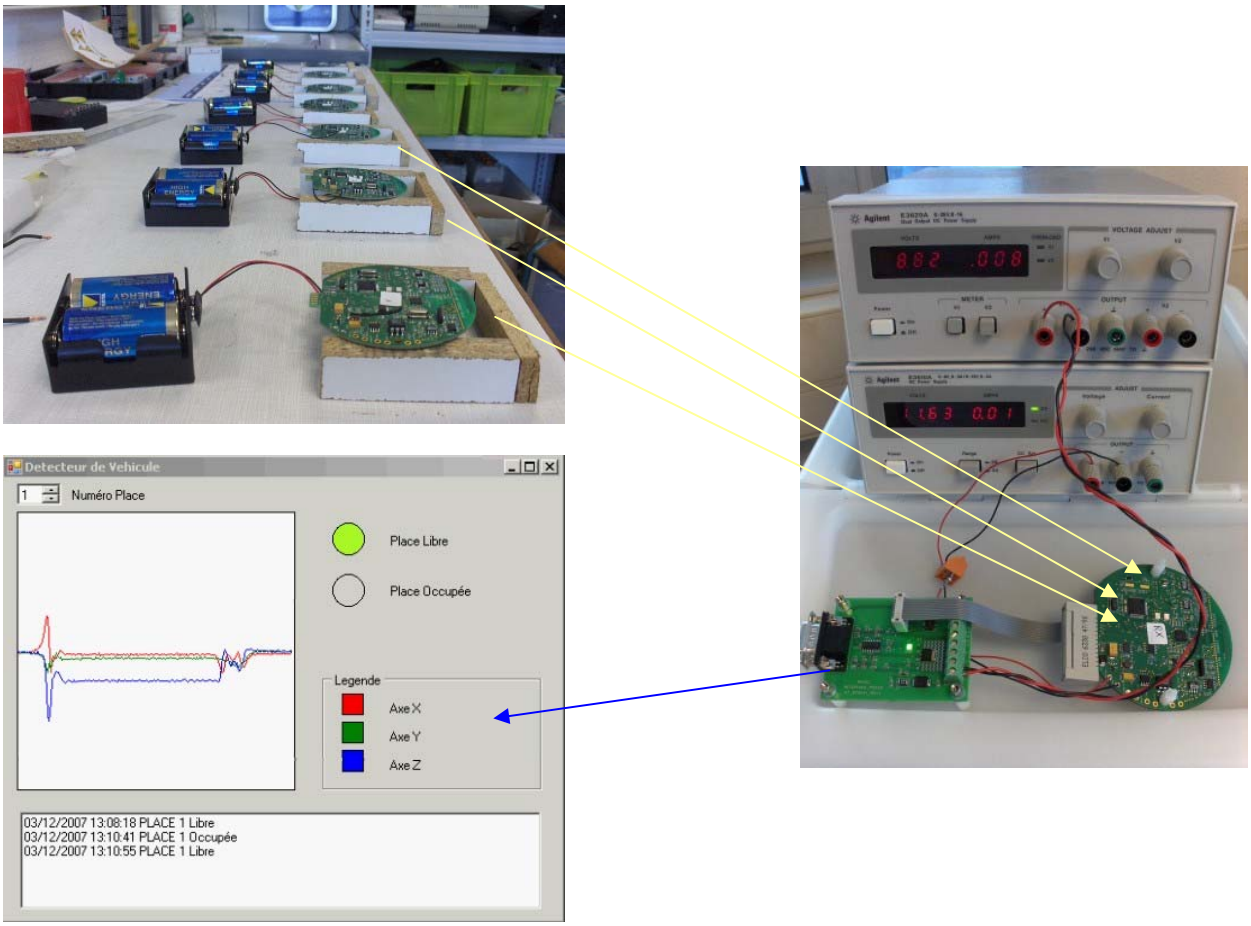


Abb. 37 : Aufbauschema Tests im Labor

Die einzelnen Endgeräte wurden im Labor aufgebaut und initialisiert. Damit die Initialisierung (Klasse amr_autoconf (☞ *Kapitel 4.3.3.12*)) korrekt durchgeführt werden kann, muss die Platine stabil stehen und es muss darauf geachtet werden, dass nichts in unmittelbarer Umgebung das System stören könnte.
Mit einer magnetischen Komponente wird nun die Umgebung der einzelnen Sensoren gestört, wie es bei einem Fahrzeug auch der Fall wäre. Vielleicht sogar noch ein wenig extremer, so dass das System näher an die Grenzen gebracht wird.

Ein Koordinator fängt alle (an ihn adressierten) ZigBee-Rahmen auf und gibt sie wie weiter oben erwähnt mittels RS232 an das Testprogramm (☞ *Kapitel 5.1*) weiter. Auf dem Bildschirm des PCs kann nun die Entwicklung der Sensormesswerte grafisch beobachtet werden.

B) Echtzeit-Simulation auf verschiedenen vorhandenen Parkplätzen

Nach einer gründlichen Austestung der gesamten Anwendung im Labor bzw. beim Arbeitsplatz ist man nun bereit für Echtzeit-Simulationen auf verschiedenen Parkplätzen.

Schema

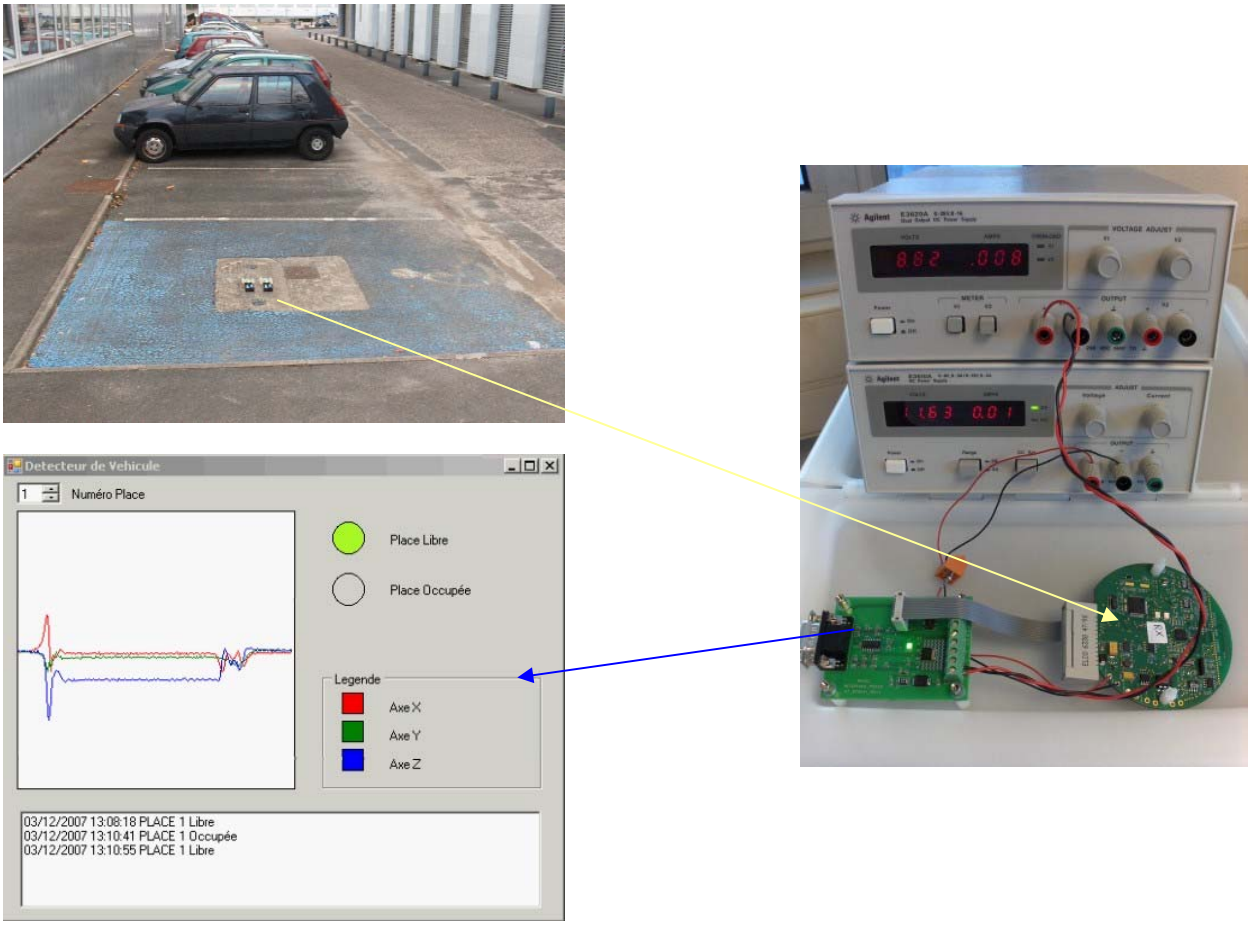


Abb. 38 : Aufbauschema auf Parkplatz

Vom Prinzip her läuft der Aufbau gleichermassen, nur wird dieses Mal mit echten Autos in einer echten Umgebung (Parkplätze) getestet. Damit die Messungen anschliessend analysiert werden konnten, wurde der Vorgang am Parkplatz mittels Kamera gefilmt und von den Auswertungskurven am PC wurde mittels des Programms „Desktop Activity Recorder“ ebenfalls ein Film erzeugt.

Folgend einige Beispiele von Testformationen, welche durchgeführt wurden:

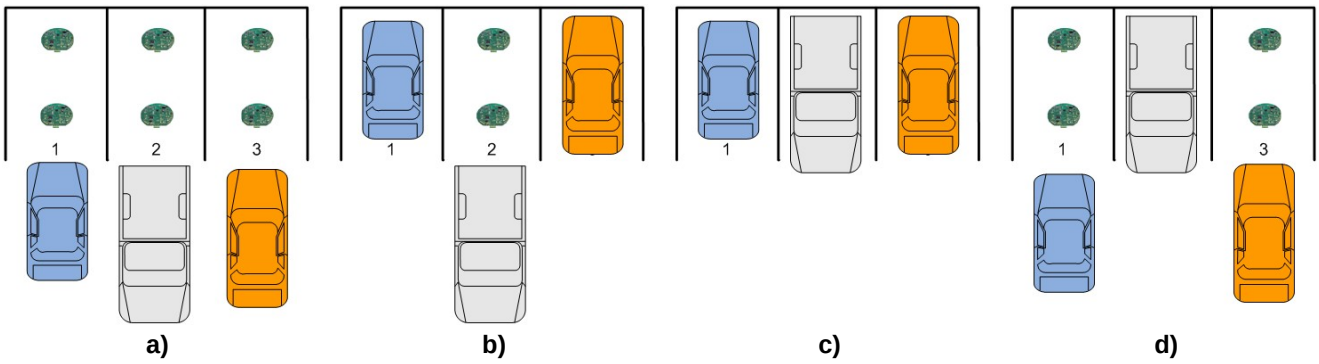


Abb. 39 : Testformationen

5.3 Verbesserungen

Durch die zahlreichen Tests und Versuche stiess man auf Probleme und Unstimmigkeiten allerlei Formen. Nach der Jagd nach Ampères bzw. Mikroampères wurde ebenfalls das ein oder andere Detail entdeckt, dass es zu Verbessern gibt.

Dieser Abschnitt zeigt die wichtigsten Verbesserungsvorschläge auf, die es zu machen gab und noch zu machen gibt.

5.3.1 RC-Filter des Verstärkerausgangs

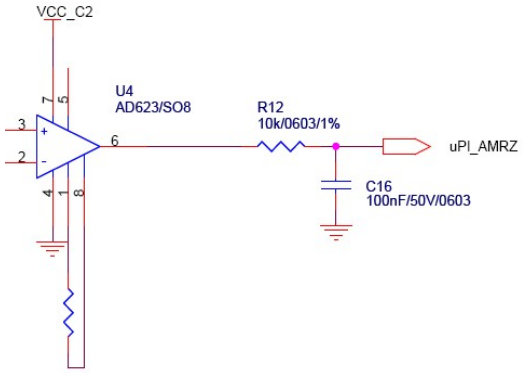


Abb. 40 : RC Filter des Verstärkerausgangs

Bei der Suche nach Verbesserungen in Fragen der Energiekonsumation ist man auf das Problem gestossen, dass der RC-Filter am Ausgang des Verstärkers viel zu langsam ist. D.h. die Ladung und Entladung des Kondensators braucht zu viel Zeit.

Diese Lade- und Entladezeit kann folgendermassen ausgerechnet werden:

Die Ladezeit des Kondensators ist proportional zur Grösse des Widerstands R und zur Kapazität C des Kondensators. Das Produkt von Widerstand und Kapazität nennt man die Zeitkonstante τ.

$$\tau = R \cdot C$$

Theoretisch dauert es unendlich lange, bis $U(t)=U_{max}$ ist. Für praktische Zwecke kann man als Ladezeit t_L verwenden, nach der der Kondensator näherungsweise als vollständig (mehr als 99%) geladen angesehen werden kann.

$$t_L = 5 \cdot \tau$$

Für die Entladung des Kondensators gelten dieselben zeitlichen Limiten.

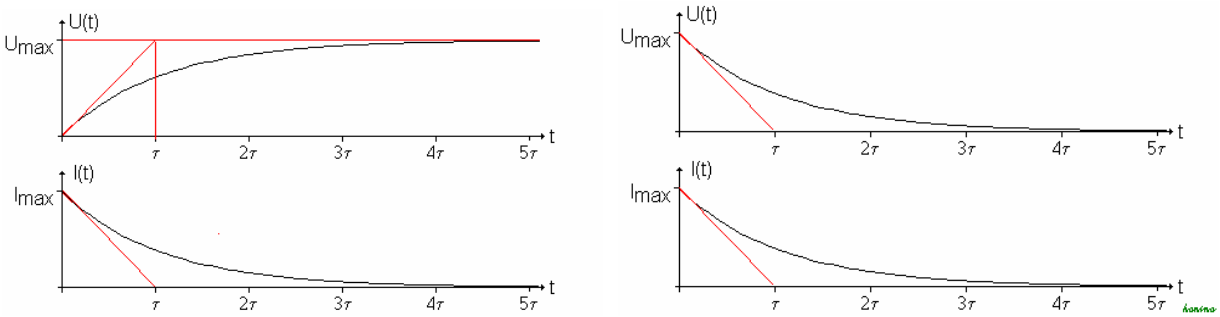


Abb. 41 : Lade- und Entladungskurve eines Kondensators in einem RC-Glied

In unserem Fall bedeutet dies also folgendes: $\tau = R \cdot C = 10\ k\Omega \cdot 100\ nF = 1\ ms$
$t_L = 5 \cdot \tau = 5 \cdot 1\ ms = 5\ ms$

Soll derselbe Tiefpassfilter beibehalten werden, so gibt es wohl keine Möglichkeit die Zeit zu senken. So wurde entschieden den Filter zu entfernen und eine Filterung per Software zu machen.

Der Softwarefilter wurde mittels gleitenden Mittelwerts erzeugt.

5.3.2 Initialisierung der digitalen Potentiometer

Der Algorithmus der Initialisierung der digitalen Potentiometer (Klasse „amr_autoconf“ (☞ *Kapitel 4.3.3.12*)) sieht so aus, dass alle möglichen Offsets des Potentiometer „durchgescannt“ werden und schlussendlich des Potentiometer mit dem Offset versetzt wird, mit welchem das verstärkte Sensorsignal in der Mitte des Spannungsbereichs des Analog-Digital Konverters (ADC) liegt.

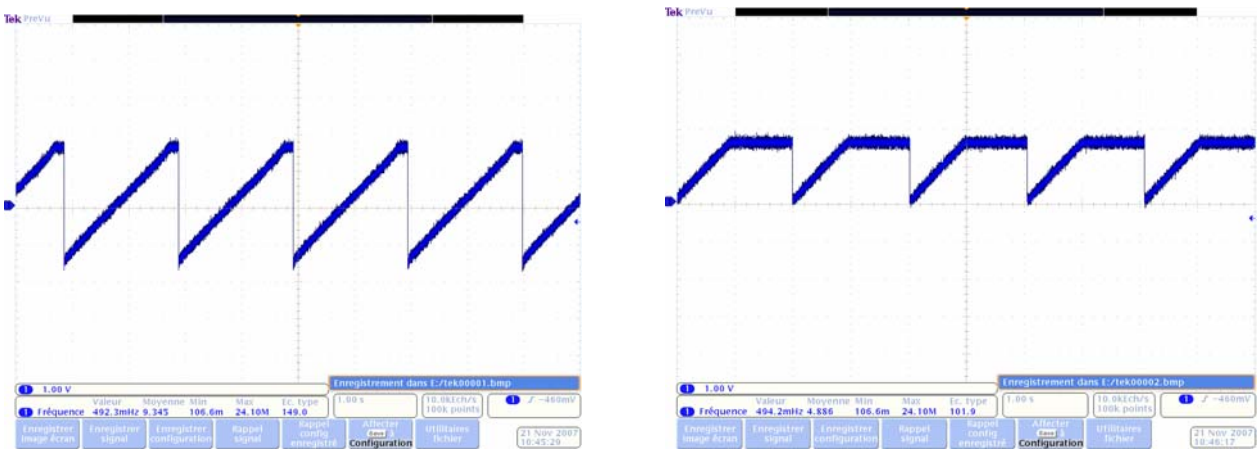


Abb. 42 : Ausgangsspannung bei Offsetdurchlauf des Potentiometers

Bei der Herstellung bzw. bei der Auslieferung der Magnetsensoren haben nicht alle Sensoren das gleiche Ausgangsoffset. Nun kann es vorkommen, dass bei keinem einzigen Offsetwert des digitalen Potentiometers das verstärkte Sensorsignal in der Mitte des Spannungsbereichs des Analog-Digital Konverters zu liegen kommt (siehe Abb. 42 Bild rechts).
In diesem Fall endet die Initialisierung der digitalen Potentiometer nie. D.h. der Initialisierungsalgorithmus muss neu überdacht werden.

Die neue Lösung soll nach dem Prinzip der Dichotomie funktionieren.
Das schlussendlich beste Offset wird dann im Potentiometer gesetzt und zusätzlich mit einem weiteren Offset in Form einer globalen Variabeln versehen.

Die Klasse „amr_autoconf“ verändert sich damit wie folgt:

Funktionen

- `void amr_init_calibration(void)`
  (Konfiguriert bzw. sucht und definiert das ideale Offset für die digitalen Potentiometer)
- *`void amr_average(void)`*
  (Erstellt verschiedene gleitende Mittelwerte der Sensorwerte)
- *`static void amr_capture(void)`*
  (Misst die Resultate mit dem aktuellen Offset um das ideale Offset zu finden)

5.3.3 Verstärkungsgrad der Verstärker

Der aktuelle Verstärkungsgrad liegt bei G = 667.

Die Sensoroffsets der Magnetsensoren liegen laut Datenblatt zwischen ± 10 mV. Bei einer Verstärkung von 667 ergeben sich Ausgangsspannungen von ± 6.67 V, was in Anbetracht unserer Betriebsspannung von 3.3 V bzw. 3 V viel zu hoch ist.

Somit muss die Verstärkung geändert werden. Die neue Verstärkung wird auf G = 166 festgesetzt, d.h. eine Verminderung der Verstärkung um den Faktor 4.

Um diese neue Verstärkung zu erreichen, müssen die Widerstände R13, R23 und R31 wie folgt verändert werden:

$$R = \frac{100\,k\Omega}{G-1} \quad \Rightarrow \quad R = 606\,\Omega$$

Somit ergeben sich Ausgangsspannungen von ± 1.66 V, was innerhalb unserer Betriebsspannung liegt.

5.3.4 Offsetverlust des digitalen Potentiometers

Durch die Ein- und Ausschaltung der Magnetsensorenkomponenten ist jeweils das aktuelle Offset des digitalen Potentiometers verloren gegangen.

Der Grund hierfür liegt darin, dass in den Funktionen `amr_set_x_ofs(offset)`, `amr_set_y_ofs(offset)` und `amr_set_z_ofs(offset)` der Klasse „amr“ (☞ *Kapitel 4.3.3.13*) das aktuell als Parameter übergebene Offset jeweils nur im RAM (RDACx Register) abgespeichert wurde. Schaltet man nun das digitale Potentiometer aus, indem man ihm die Betriebsspannung entzieht, so geht der Wert im RAM verloren.

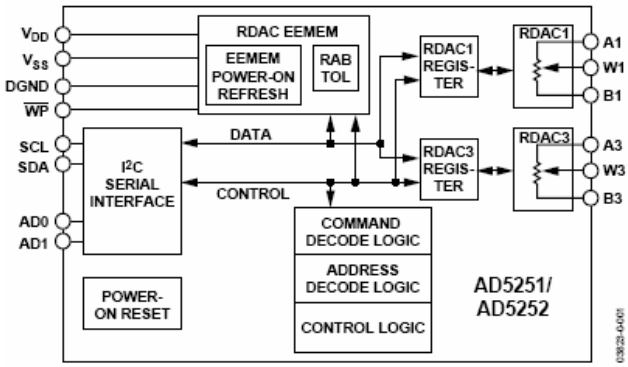


Abb. 43 : Blockdiagramm des digitalen Potentiometers

Wie in Abbildung 43 ersichtlich ist, verfügt das digitale Potentiometer auch über ein ROM (RDAC EEMEM). Beim Aufstart bzw. der Einschaltung des Potentiometers wird der Wert im ROM ins RAM geschrieben. Somit genügt es, wenn die Funktionen so abgeändert werden, damit der aktuelle Offsetwert auch jeweils ins ROM geschrieben wird.

5.3.5 Anpassung Ladungspumpe

Die Ladungspumpe gilt es zu verbessern:

- Die Kondensatoren C6, C7, C9 und C10 werden von 100 nF auf 1 nF herabgesetzt
- Ein weiterer Kondensator / Diodenblock wird angefügt

5.3.6 Dynamische Autokalibration

Es wurde häufig beobachtet, dass die Sensormesswerte bei der Rückkehrung in den Ausgangszustand nicht mehr ganz exakt zurück auf den Ursprungswert (ideal Mittelwert des Spannungsbereichs) kommen. Manchmal nicht einmal mehr nach einem Set/Reset-Impuls.
Der Grund hierbei liegt an der Sensibilität der Magnetsensoren, die durch eine zu starke Störung ein wenig verändert werden.

Um diesem Phänomen entgegenzuwirken wurde die Klasse „amr_autoconf“ (*Kapitel 4.3.3.12*) um eine weitere Funktion erweitert, die, falls kein Fahrzeug detektiert wurde, alle 10 Sekunden den aktuellen Sensorsignalwert um eine ADC-Einheit zum Mittelwert hin korrigiert.

Zusätzliche Funktionen

- `void amr_dyn_calib(void)`
  (Korrigiert den globalen Offsetwert der Magnetsensoren)

5.3.7 Verbesserung Set/Reset

Die vorgeschlagenen Schaltungen für den Set/Reset-Impuls sind so gedacht, dass eine Spannungsspitze von 5 Volt und mehr vorliegt. Die Betriebsspannung auf dieser Platine liegt bei 3.3 V bzw. 3 V was für den Set/Reset-Impuls nicht unbedingt ideal ist.

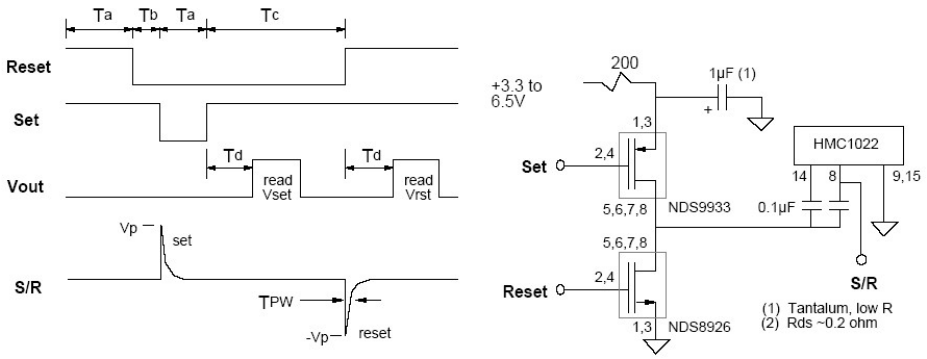


Abb. 44 : Angewandte Set/Reset Schaltung

Es gilt hier vielleicht eine Lösung zu finden den Set/Reset-Mechanismus zu Verbessern indem bei den Impulsen eine höhere Spannung erzeugt werden kann.
Zurzeit wird einmal pro Minute ein Set/Reset Impuls ausgeführt. Ein guter Set/Reset ist für diese Anwendung das A und O. Es bildet den Rückgrad für eine gut funktionierende Anwendung.

Für eine Revision 4 der Hardwareplatine könnte evtl. die folgende Schaltung interessant sein:

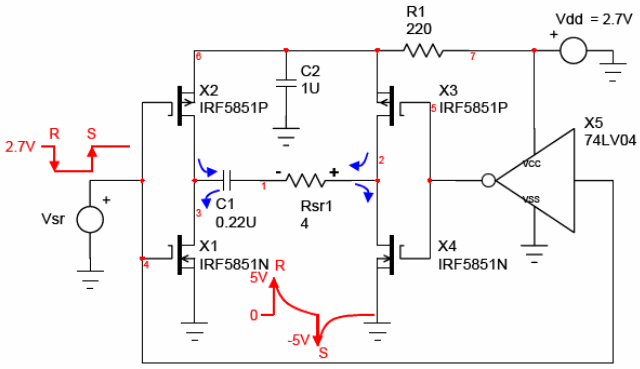


Abb. 45 : Verbesserte Set/Reset Schaltung (H-Brücke)

5.3.8 Änderung Fahrzeugserkennungsalgorithmus

Es konnte festgestellt werden, dass Störquellen mit einer Frequenz von 50 Hz (z.B. Signalgenerator, etc.) das System stören können. Dies macht sich auf den Signalwerten der Magnetsensoren bemerkbar indem sie in dieser Störfrequenz schwingen.

Mit Hilfe eines Filters in Form eines gleitenden Mittelwerts kann umgangen werden, doch wird somit der Algorithmus der Fahrzeugerkennung (Klasse „czame“ (☞ *Kapitel 4.3.3.14*)) unbrauchbar. Wendet man aber keinen Filter an, reicht diese Schwingung aus, dass der Algorithmus ein Fahrzeug identifiziert. Dies ist in dieser Form natürlich nicht brauchbar, denn können auch bei Parkplätzen solche Störquellen vorkommen. Die 50 Hz Form ist häufig, ist überall vorhanden.

Aus diesem Grund war man gezwungen den Algorithmus der Fahrzeugerkennung zu überdenken:

In einer ersten Lösung wurden nun einfache Schwellwerte implementiert. Werden diese überschritten, so bedeutet dies, dass ein Fahrzeug vorhanden ist. Diese Schwellwerte sind aus Erfahrungswerten einiger Tests mit verschiedenen Fahrzeugen zustande gekommen.

Der Klasse „amr_autoconf“ (*Kapitel 4.3.3.12*) wurde eine weitere Funktion hinzugefügt, die eben diesen neuen Algorithmus der Fahrzeugerkennung beherbergt.

Zusätzliche Funktionen

- `void amr_detection(void)`
  (Kalkuliert den Zustand des Parkplatzes)

5.3.9 Sonstiges

Da der eingebaute Spannungswandler, der die Batteriespannung von 3 V auf 3.3 V erhöhen sollte, zuviel Energie konsumiert, wird zurzeit einfach mit 3 V gearbeitet. Mit einem Kabel wird der Spannungswandler überbrückt, jedoch wäre für eine Revision 4 eine Brücke oder eine Komponentenänderung wünschenswert.

Die Dauer einer SPI Kommunikationsperiode konnte um einiges verringert werden indem eine unnötig als volatile deklarierte Variable als non-volatile umdeklariert wurde.

Mittlerweile gibt es einen 3-Achsen Magnetsensor von Honeywell auf dem Markt, bzw. er ist angekündigt. Ein solcher sollte für eine Revision 4 auch ins Auge gefasst werden.

6. Fazit und Aussichten

Nach vielen Tests und Versuchen läuft der Algorithmus nun mittlerweile ziemlich stabil.
Trotzdem ist die Revision 3 in dieser Form noch nicht bereit um einem Kunden zu übergeben, der diese bereits einsetzen oder weiterverkaufen möchte.

Als nächster Schritt sollte vielleicht die Umsetzung der Revision 4 der Hardwareplatine ins Auge gefasst werden.

Auch wurden nie Tests mit der schlussendlichen Installationsform gemacht. D.h. dass der Sensor auf die Batterien gelegt wird und mit einem Deckel versehen wird. Dies gilt es unbedingt nachzuholen.

Weiterhin bleiben aber noch einige Sachen zu definieren und implementieren. Beispielsweise blieb die Frage offen, wie auf die einzelnen Sensoren nach der Installation und bei Betrieb zugegriffen werden kann, um beispielsweise eine Adress- oder sonstige Parameteränderung vorzunehmen.

Auch das Thema Verschlüsselung der Kommunikation sollte nicht vergessen werden. Für einen Hacker wäre es im Moment wohl nicht allzu schwierig die Kommunikation zu stören.

Ein vielleicht noch ziemlich wichtiger Aspekt der auch noch nicht gross bedacht wurde, ist die Frage der Temperatur. Es bedarf noch einiger Tests und wohl auch Anpassungen um zu gewährleisten, dass das System sowie im Sommer bei 30°C als auch im Winter bei -5°C läuft.

In der Frage der Energiekonsumation steht man aber soweit ganz gut da:

Messungen an neun Sensoren mit folgender Charakteristik haben einen durchschnittlichen Verbrauch von ca. 350 uA aufgezeigt:

- Set/Reset-Impuls : 60 s
- Sendeimpuls : 1 s
- Sensorabtastfrequenz des ADC : 12 Hz

Mit den derzeitig eingesetzten Batterien, welche eine Elektrische Ladung von 16.5 Ah (Ampèrestunden) aufweisen, ergibt dies eine hochgerechnete Betriebsdauer von ca. 5.3 Jahren.

Kann der Set/Reset-Impuls verbessert werden und es bedarf nicht mehr in jeder Minute einen solchen Impuls, so kann der Verbrauch noch verringert werden.

7. Schlussfolgerung und Bemerkungen

Während dieser Diplomarbeit sind also folgende Arbeiten durchgeführt worden:

- Studium der Standards ZigBee und IEEE 802.15.4
- Einlesen und Studium der Hardwarekomponenten MSP430, CC2420, Magnetsensoren, digitale Potentiometer, Verstärker, etc. mittels Datenblätter und des Elektroschemas
- Verstehen vorhandener Software-Programmteile
- Kennenlernen der IAR Workbench Programmierumgebung und sonstigen Tools z.B. wie C Sharp
- Konzeption und Erstellung von Softwareteilen
- Aufbau und Durchführung von Tests und Versuchen
- Fehlersuche in Hard- und Software
- Dokumentation

Allgemein gilt zu sagen, dass eine Fahrzeugerkennung mittels Magnetsensoren nicht ganz leicht zu bewerkstelligen ist. Nach vielen Tests ist das Gefühl da, dass der ganze Erkennungsalgorithmus zwar ziemlich stabil läuft, d.h. zu vielleicht 97 % das Richtige anzeigt, doch darf man den Aspekt „Zukunft“ nicht vergessen. In der Zukunft gibt es evtl. vermehrt Technologien, die auf der Basis des Magnetismus aufbauen. Wie sich die vermehrten Magnetfelder auf dieses Produkt auswirken ist schwer abzuschätzen.

Vielerorts liest und hört man, dass ZigBee die Zukunft für Sensor- und Aktor-Netzwerke ist. Allmählich kommen nun auch Produkte auf den Markt, doch ist zum jetzigen Zeitpunkt die Auswahl noch ziemlich gering. Will man den Rahmen des Standards IEEE 802.15.4 übersteigen und auch die von der ZigBee Allianz vorgeschlagenen Normen nutzen, so braucht es doch noch einigen Zeitaufwand und Geld um sich hier zu Recht zu finden. Die Weichen für die ZigBee-Zukunft sind gelegt, doch wo sie hinführen bleibt noch abzuwarten.

Allen Bedenken zum Trotz war dies ein sehr interessantes Projekt und es bleibt zu hoffen, dass sich dieses nun auch bewähren kann.

Durch die Benutzung neuer Komponenten in Hard- und Software, wie dem IAR Workbench in Sachen Software oder dem MSP430 in Sachen Hardware und vielem mehr konnte viel neue Erfahrung im Bereich der eingebetteten Systeme gewonnen werden. Es war eine gute erste Einführung in die Arbeitswelt eines Ingenieurs.
Eine zusätzlich tolle Erfahrung war die Arbeit im Ausland zu schreiben, wo es doch manchmal vorkommt, dass man einander nicht richtig versteht und man nicht immer allen Erklärungen folgen kann. Doch schlussendlich nach vielleicht auch mehrmaligem Nachfragen kam man auf einen gemeinsamen Nenner.

Michel Gemmet

8. Quellenverzeichnis

[01] IEEE 802.15.4 Standard – 2003

[02] ZigBee Specifikation – 1. December 2006

[03] MSP430x1xx Family User's Guide (Rev. F)

[04] MSP430F15x/16x/161x Mixed Signal Microcontroller (Rev. E)

[05] CC2420 - 2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver (Rev. B)

[06] I2C-Bus Spezification and user manual (Rev. 03)

[07] Dual 64-and 256-Position I2C Nonvolatile Memory Digital Potentiometers

[08] Single Supply, Rail-to-Rail, Low Cost Instrumentation Amplifier

[09] Honeywell - 1 and 2 axis Magnetic Sensors

[10] Low Voltage 4 ohms Dual SPST Switches

[11] MaxStream - Product Manual (802.15.4)

[12] Honeywell – AN213 (Set/Reset function for magnetic sensors)

[13] Honeywell – AN218 (Vehicle detection using AMR sensors)

[14] Applications of magneto resistive sensors in navigation systems

[15] Vehicle detection and compass application using AMR magnetic sensors

[16] ZigBee – Protokollsoftware und Entwicklungsumgebung

[17] ZigBee – Grundlagen und Applikation

9. Abkürzungsverzeichnis

ADC – Analog Digital Converter
CCA – Clear Channel Assessment
CSMA/CA – Carries Sense Multiple Access with Collision Avoidance
FCS – Frame Check Sequence
FFD – Full Function Device
GUI – Graphical User Interface
I^2C – Inter-Integrated Circuit
LQI – Link Quality Indication
MAC – Media Access Control
NWK – Network
PAN – Personal Area Network
PHY – Physical
PWM – Pulsweitenmodulation (Pulse Width Modulation)
QoS – Quality of Service
RAM – Random Access Memory
RF – Radio Frequency
RFD – Reduced Function Device
SPI – Serial Peripheral Interface
USART – Universal Synchronous Asynchronous Receiver Transmitter

10. Anhang

- Anhang 1 : Projektvorschlag
  Begriffserklärungen
- Anhang 2 : Elektroschema
- Anhang 3 : Quellcodes

Inhaltsverzeichnis

1. Verfasser

Michel Gemmet
Laboratoire IMS - Bât. A31
351, Cours de la libération
33405 Talence Cedex - France

2. Projektvorschlag

TITRE

Capteur de détection de présence de véhicule avec interface Zigbee

CONTEXTE

Ce projet met en œuvre la technologie Zigbee :

Le ZigBee (IEEE 802.15.4) est une norme de transmission de données sans fil au même titre que le Bluetooth ou Wifi. Ses points forts son un faible coût (quelques dollars pour une puce ZigBee) et surtout sa faible consommation électrique.

SUJET

L'objectif principal de ce projet est le développement d'un logiciel gérant le protocole de communication Zigbee. Pour ce faire, vous devrez mettre en œuvre un démonstrateur complet en accord avec la norme IEE 802.15.4. Ce système contiendra un ensemble de capteurs de détection de présence de véhicule avec interface Zigbee.

COMPETENCES

- microinformatique (microprocesseurs Texas Instrument MSP430, outils IAR)
- réseau (Zigbee)
- conception électronique (CAO Cadence Orcad)

NIVEAU

Ecole d'ingénieur électronique

DUREE

4 mois

LIEU

Laboratoire IMS
Bâtiment A31 – 351, cours de la Libération – 33405 TALENCE Cedex

RESPONSABLE

M. Eric STEMPIN (Tél. : 05 40 00 27 73) – stempin@ixl.fr
M. Jean-Christophe MARTIN (05 40 00 27 70) – martinj@ixl.fr

SOCIETE

ADERA-RAVEL
162, avenue du Dr Schweitzer
33608 PESSAC CEDEX

Mme Valérie MOREAU (05 56 15 11 57) – moreau@adera.fr

3. Begriffserklärungen

Im folgenden Abschnitt werden einige Begriffe näher erklärt, die im Bericht auftauchten, jedoch nicht näher angeschaut wurden.

3.1 Balun

Ein Balun (von Balanced-unbalanced, n.) ist in der Elektrotechnik und Hochfrequenztechnik ein Impedanzwandler und kann gleichzeitig auch zur Potentialtrennung dienen oder Symmetrierglied sein. Der Begriff Balun ist jedoch umfassender.

Symmetrierglied (Balun) für das L-Band, bestehend aus einer λ/2-Umwegleitung aus Semi-Rigid-KoaxialkabelIn der Hochfrequenztechnik und besonders an Senderendstufen hat man oft unterschiedliche Wechselstromwiderstände (Impedanzen, Wellen- oder Quellwiderstände) aneinander anzupassen. Häufig wird hier ein Transformator eingesetzt, der auf die gewünschten Widerstände bzw. Impedanzen oder Wellenwiderstände transformiert.

Im Fall des Balun ist das o.g. jedoch nur eine Aufgabe, und die muss nicht immer die eigentliche Aufgabe sein. Der Transformator ist hier zusätzlich ein Wandler von symmetrischen Leitungssystemen (z.B. 240-Ohm-Flachbandkabel, Zweidrahtleitung) auf Koaxialkabel (Wellenwiderstand 50 oder 75 Ohm).

Flachbandkabel sind verlustärmer als Koaxialkabel, daher verwendet man sie teilweise auch heute noch bei Amateurfunk-Sendeanlagen.

Die Phasenlage auf den zwei Leitern des Flachbandkabels ist gegeneinander um 180° verschoben. Das Balun (Symmetrierglied) kehrt die Phasenlagen gleichsinnig zueinander um und senkt gleichzeitig die Impedanz auf 1/4.

In der Audiotechnik werden ebenfalls oft symmetrische Leitungsverbindungen (z.B. zwischen Gitarre, Mikrofon und Verstärker) genutzt, um Störungen zu verringern. Die Rückwandlung wird im Mischpult vorgenommen. Dort sitzt ebenfalls ein als Transformator ausgeführtes BALUN.

Es gibt auch Baluns mit einem Impedanzverhältnis von 1:1, dann werden lediglich die Potentialverhältnisse gegen Erde (symmetrisch/unsymmetrisch) transformiert.
Ziel ist hierbei meist, Mantelwellen auf koaxialen Kabeln und die damit verbundene Störeinstreuung oder -abstrahlung zu verhindern. Im einfachsten Fall besteht ein solches Balun aus einigen Windungen des Koaxialkabels, aus aufgeschobenen Ferrit-Ringkernen oder aus einer trifilar gewickelte Luftspule.

Baluns werden auch in der Messtechnik verwendet. Sie sind oft Bestandteil zur Impedanzanpassung von Antennenanlagen bei Kurzwelle, UKW und UHF.
Typische Hochfrequenz-Anwendung ist der Übergang mit einem Balun zwischen koaxialen Leitungen (in der Regel Anschlussleitungen) und Zweidrahtleitungen oder anderen symmetrischen Quellen (z. B. ein Dipol). Direkt im Anschlusskasten eines Halbwellen-Schleifendipols (240 Ohm, symmetrisch) ist oft ein Balun (Symmetrierglied) zum Anschluss an ein Koaxialkabel (50…75 Ohm, unsymmetrisch) untergebracht.

Als Balun werden beispielsweise auch impedanzangepasste (100/150 Ohm) Steck-Adapter zwischen RJ45- und IVS-Datenstecker (Typ 1) bezeichnet, die in Token-Ring- sowie Ethernet-Netzwerkwerken verwendet werden. Die Pinbelegung variiert je nach Netzwerktopologie.

3.2 SPI

SPI bedeutet Serial Peripheral Interface – zu Deutsch „serielle Peripherie Schnittstelle“ – und beschreibt einen synchronen seriellen Bus, welcher seines Zeichens von Motorola entwickelt wurde.
Dieser ermöglicht die Anbindung von Peripherien an einen Mikrocontroller. Auch ist die Verbindung mehrerer Mikrocontroller miteinander möglich. SPI erreicht hier sehr hohe Datenübertragungsraten, da das Taktsignal bis in den MHz Bereich reichen kann. Ausserdem werden die Daten in beide Richtungen gleichzeitig übertragen, also Voll-Duplex.

Der Hardware Aufwand bleibt dabei in Grenzen, da neben den Slave-Select- oder Chip-Select-Leitungen nur 1 Steuerleitung für den Takt sowie 2 Datenleitungen benötigt werden. In der Konfiguration (Polarität etc.) ist SPI ebenfalls sehr flexibel.

SPI ist als Master-Slave Bus ausgelegt, d.h. ein Master muss immer die Datenübertragung einleiten sowie die Slaves selektieren. Auch stellt der Master das Taktsignal bereit. Die Daten werden auf 2 Datenleitungen übertragen. Diese sind MISO (Master-In Slave-Out, Dateneingang Master) und MOSI (Master-Out Slave-In, Datenausgang Master). Jeder Slave besitzt ein Slave Select – Signal oder Chip Select – Signal. Diese sind meistens Low-Aktiv. Solange ein Slave nicht ausgewählt – selektiert – ist, sind Taktsignal und die Datensignale im sogenannten TriState-Modus und extrem hochohmig. Es werden also keine Daten / Takte zur SPI-Einheit durchgelassen.

Synchron zum Taktsignal vom Master werden die Daten an den Datenleitungen ausgegeben. Dieses wird über ein Schieberegister realisiert. Meist wird das höchstwertigste Bit (MSB) zuerst ausgegeben, die niederwertigen Bits folgend. Bei manchen Mikrocontrollern kann man einstellen, ob zuerst MSB oder LSB gesendet werden soll. Ein Datenwort beträgt bei der SPI immer 8Bit, sei es in 8, 16 oder 32-Bit Systemen. Die empfangenen Daten liegen nach dem Datentransfer im gleichen Register wie die Sendedaten. Es existiert also nur ein Register für Sende-/Empfangsdaten. Schreibt man ein Datenwort in dieses Register, so wird automatisch eine Datenübertragung eingeleitet. Um festzustellen ob die Übertragung abgeschlossen ist, gibt es Flags in Statusregistern oder man verwendet Interrupts.

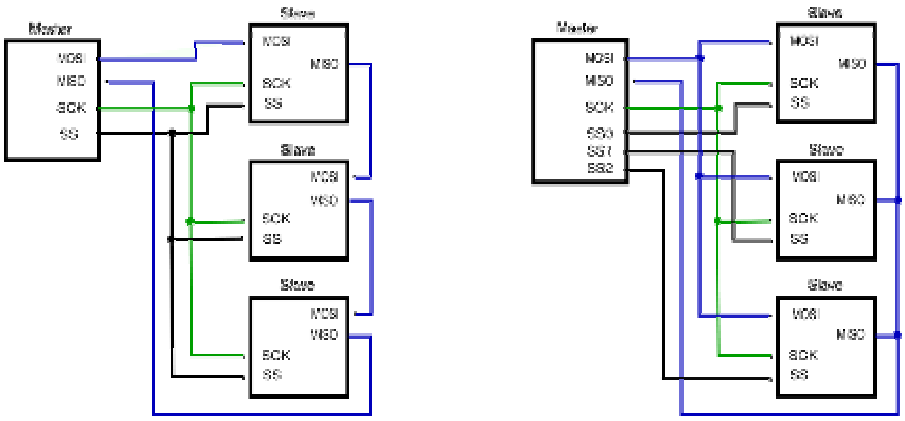


Abb. 1: SPI Verbindung durch Kaskadierung der Slaves (links) und Sternverbindung (rechts)

3.3 I²C

Die meisten 32-Bit Mikroprozessoren besitzt ein Two-Wire Serial Interface (TWI). Diese Schnittstelle ist äquivalent zum I²C-Bus, der, wie „Two Wire“ schon sagt, mit nur zwei Leitungen auskommt.
Dieser Bus ist ein serieller Datenbus, der um 1980 von Philips entwickelt wurde, um in Geräten wie Fernsehern mehrere ICs an Mikroprozessoren anschliessen zu können ohne breite Adress- und Datenleitungen zu benötigen. Daher kommt auch die Bezeichnung I²C, was eine Abkürzung für „Inter IC“ ist. Gemeint ist damit, dass der Bus für Verbindungen zwischen ICs gedacht ist.
Der Bus hat in der aktuellen I²C-Spezifikation 2.1 von 2001 eine maximale Datenübertragungsrate von 3,4 Mbit/s (High-Speed Modus).

Der grosse Vorteil des Busses ist, dass man mit nur zwei Leitungen viele ICs miteinander verbinden kann. Dazu kommt, dass während einer Datenübertragung der Takt automatisch von den beteiligten Geräten angepasst werden kann, falls eines der Geräte mehr Zeit zur Verarbeitung der Daten benötigt. Es ist ausserdem ein Arbitrationsverfahren vorgesehen. Dieses Verfahren sorgt dafür, dass nur ein Gerät erfolgreich Daten auf den Bus schreiben kann, wenn mehrere Geräte gleichzeitig versuchen eine Verbindung auf dem Bus zu initiieren.
Zur Adressierung der am Bus angeschlossenen Geräte stehen im Allgemeinen wahlweise 7-bit oder 10-bit Adressierung und damit 112 bzw. 1024 Adressen zur Verfügung.
Dadurch, dass es so wenige Adressen gibt, jedoch die Auswahl an ICs mit I²C-Ansteuerung gross ist, kommt es dazu, dass verschiedenen Geräten desselben Typs dieselbe Adresse zugewiesen wird. Problematisch kann dies werden, wenn man beispielsweise mehrere serielle EEPROMs am Bus betreiben möchte, zumal manche I²C-fähigen EEPROMs gleich 8 Adressen belegen, wodurch bei 7-bit Adressierung nur ein einziges EEPROM am Bus betrieben werden kann.
Generell können durch die in der Spezifikation festgelegte maximale Leitungskapazität von 400pF ca. 20 Geräte an einem Bus betrieben werden, alternativ können die Leitungen ca. 10 m lang sein. Durch I²C Repeater o. Ä. kann diese Beschränkung jedoch umgangen werden.

<u>Ablauf einer Verbindung</u>

Der I²C-Bus verwendet zwei Leitungen zur Datenübertragung, dabei ist eine der beiden Leitungen eine Datenleitung, während die andere Leitung eine Taktleitung ist. Entsprechend werden die Leitungen SDA für Serial DAta bzw SCL für Serial CLock genannt. Beide Leitungen sind laut Spezifikation mit Pullups auf High-Level zu halten; die SDA- und SCL-Pins der angeschlossenen ICs sind Open-Drain bzw. Open-Collector. Daher muss ein am Bus angeschlossener IC um eine 1 zu senden die Leitung loslassen und um eine 0 zu senden die Leitung auf Masse ziehen. Es ergibt sich dadurch eine Wired-And Funktion der Busleitungen.

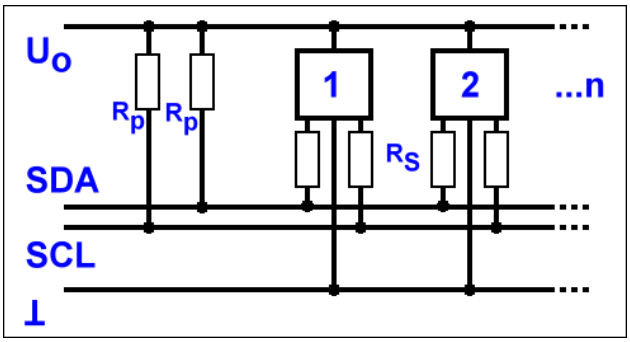


Abb. 2 : I²C Bus

Der Bus ist so ausgelegt, dass ein Busteilnehmer eine Verbindung zu einem anderen Busteilnehmer initiiert und nachher die Verbindung beendet oder eine weitere Verbindung aufbaut. Während einer Verbindung dürfen andere Busteilnehmer nicht auf den Bus zugreifen. Damit die am Bus angeschlossenen Geräte erkennen können, ob der Bus frei oder belegt ist, werden START- und STOP Conditions verwendet, die Anfang und Ende einer Verbindung kennzeichnen.

Eine START-Condition wird ausgelöst durch einen High-Low Übergang auf SDA, während SCL High ist und eine STOP-Condition wird umgekehrt ausgelöst durch einen Low-High Übergang auf SDA, während SCL High ist. Ausser beim Auslösen von START- bzw. STOP-Conditions darf sich SDA während der High-Phase von SCL jedoch nicht verändern.

Die Busteilnehmer können jeweils Master oder Slave, Transmitter oder Receiver sein. Bei einer Verbindung zwischen zwei Busteilnehmern ist jeweils eines der Geräte Master bzw. Slave; gleichermassen ist jeweils eines der Geräte Transmitter oder Receiver.
Dabei ist der Master dafür zuständig, die Verbindung zu einem Slave-Gerät zu initiieren und zu beenden, sowie den Takt auf der SCL-Leitung bereitzustellen. Ausserdem bestimmt der Master die Datenrichtung der Verbindung, d. h. welches der beiden Geräte Receiver und welches Transmitter ist.
Ein Gerät, das die Rolle eines Slave einnimmt, spricht auf eine Verbindungsanfrage eines Masters an, wenn es die eigene Geräteadresse empfängt. Es reagiert dann abhängig von der Datenrichtung mit dem Senden, bzw. Empfangen von Daten. Zusätzlich hat der Slave die Möglichkeit, die Taktrate auf der SCL-Leitung herabzusetzen, wenn diese zu schnell ist um die empfangenen bzw. zu sendenden Daten abzuarbeiten.
Der Transmitter ist das Gerät, das nach erfolgreichem Verbindungsaufbau Daten auf den Bus legt und der Receiver ist das Gerät, das die vom Transmitter gesendeten Daten empfängt.

3.4 CSMA/CA

Der englische Begriff Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) (zu Deutsch etwa: 'Mehrfachzugriff mit Trägerprüfung mit Kollisionsvermeidung') bezeichnet ein Prinzip für den Zugriff mehrerer Netzwerkstationen auf dasselbe Übertragungsmedium. Es wird häufig u.a. bei drahtlosen Netzwerken (Wireless LANs) eingesetzt, findet aber abgewandelt auch bei Technologien wie ISDN Anwendung, oder bei vielen Technologien, bei denen mehrere Clients Daten auf einen BUS legen und es nicht zu Kollisionen kommen darf.

Drahtlose Netze unterscheiden sich im Bezug auf den gemeinsamen Medienzugriff durch zwei wichtige Faktoren von drahtgebundenen Netzen:

- Der Netzadapter ist nicht notwendigerweise Voll-Duplex-fähig. Während einer eigenen Übertragung kann das Medium nicht überwacht werden. Der Einsatz eines "Collision Detection"-Mechanismus, wie er etwa von CSMA/CD vorgesehen ist und bei Ethernet verwendet wird, würde dann fehlschlagen. Deswegen wurde CSMA/CD zu einem Mechanismus weiterentwickelt, der konsequenter dem Prinzip "listen before talk" ("erst hören, dann sprechen") folgt. An die Stelle der Kollisionserkennung ("CD") sollte die (bestmögliche) Kollisionsvermeidung ("CA") treten. Dadurch lassen sich gleichzeitige Datenübertragungen zwar nicht völlig verhindern, aber doch minimieren.
- Die Reichweite des Signals ist stark begrenzt, da die Signalstärke quadratisch mit der Entfernung abnimmt. Deshalb kann es zu Effekten wie "versteckten" oder "ausgelieferten" Endgeräten kommen.

<u>Versteckte und ausgelieferte Endgeräte</u>

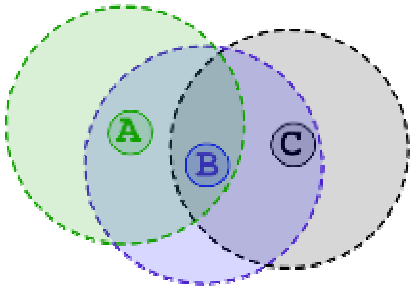


Abb. 3 : Verstecktes Endgerät

Zu einem versteckten Endgerät (engl. Hidden Station bzw. Hidden Terminal) kommt es zum Beispiel bei folgendem Szenario: Die zwei Funkteilnehmer A und C liegen räumlich so weit auseinander, dass sie ihre Funksignale gegenseitig nicht empfangen können. Zwischen ihnen liegt die Station B. A und C senden nun zeitgleich an B und erzeugen dort einen Konflikt, können diesen aber nicht erkennen, da sie die Funksignale des jeweils anderen ja nicht erreichen. A ist für C ein verstecktes Endgerät und umgekehrt.

Unter einem ausgelieferten Endgerät (engl. Exposed Station bzw. Exposed Terminal) versteht man, wenn in unserem vorliegenden Szenario die Station B an A sendet und nun C an irgendeine andere Station (nicht A oder B) senden möchte. C erkennt die Signale von B und wartet, bis die Übertragung zwischen B und A vorbei ist. Da die Funkwellen von C aber A gar nicht erreichen können, wäre es gar nicht nötig zu warten: bei A könnte gar kein Konflikt auftreten. Dennoch ist C den anderen beiden Stationen ausgeliefert.

Protokollablauf

Möchte ein Gerät Daten nach dem CSMA/CA Verfahren versenden, so ist u.a. folgender Ablauf möglich:

1. Zuerst wird das Medium abgehört "horch"("Carrier Sense")
2. Ist das Medium für die Dauer eines IFS frei, wird gesendet
3. Ist das Medium belegt, wird auf einen freien IFS gewartet und zur Kollisionsvermeidung zusätzlich um eine zufällige Backoff-Zeit verzögert
4. Wird das Medium während der Backoff-Zeit von einer anderen Station belegt, bleibt der Backoff-Timer so lange stehen und wird nach Freiwerden des Mediums weitergezählt

Um das Problem der "versteckten" oder "ausgelieferten" Endgeräte zu beseitigen, existiert eine "Request to Send / Clear to Send" (RTS/CTS)-Erweiterung für CSMA/CA. Dabei muss die sendewillige Station den Kanal durch ein RTS-Paket reservieren. Der Empfänger bestätigt diese Reservierung mit einem CTS-Paket. Anschließend ist ein sofortiges Senden der Daten möglich. Andere Stationen speichern die Belegungsdauer, die im RTS- und CTS-Paket gesendet wurde, und senden während dieser Zeit keine Daten

3.5 Dichotomie

Dichotomie bedeutet die Aufteilung in zwei Strukturen oder Begriffe, die nicht miteinander vereinbar sind.
In der Mathematik, der Philosophie, der Logik und der Sprachwissenschaft bezeichnet Dichotomie die Trennung eines Begriffs in zwei Unterbegriffe, die sich gegenseitig ausschliessen.

Das beste praktische Beispiel, welches die Dichotomie erklärt, ist das Spiel wo sich eine Person eine Zahl zwischen 1 und 100 merkt und die andere Person muss sie erraten.

Beispielsweise merkt sich Person A 81 so würde der Ablauf nach Dichotomie wie folgt ablaufen:

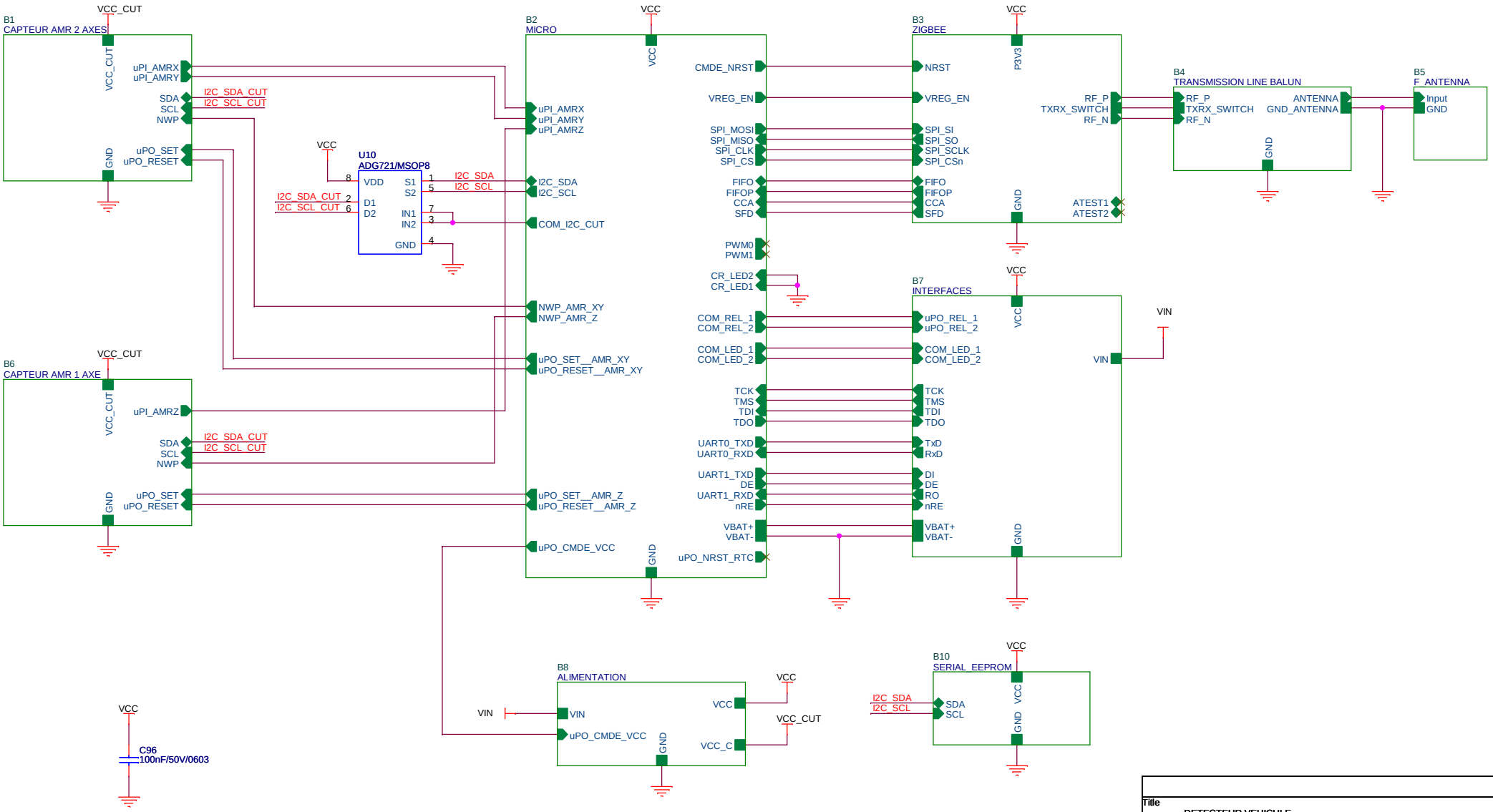
Person B rät 50.
Person A erklärt Person B, dass die Zahl grösser sei.

Person B rät 75.
Person A sagt zu Person B, dass die Zahl immer noch grösser sei.

Person B rät 87.
Person A sagt diesmal zu Person B, dass die Zahl kleiner sei.

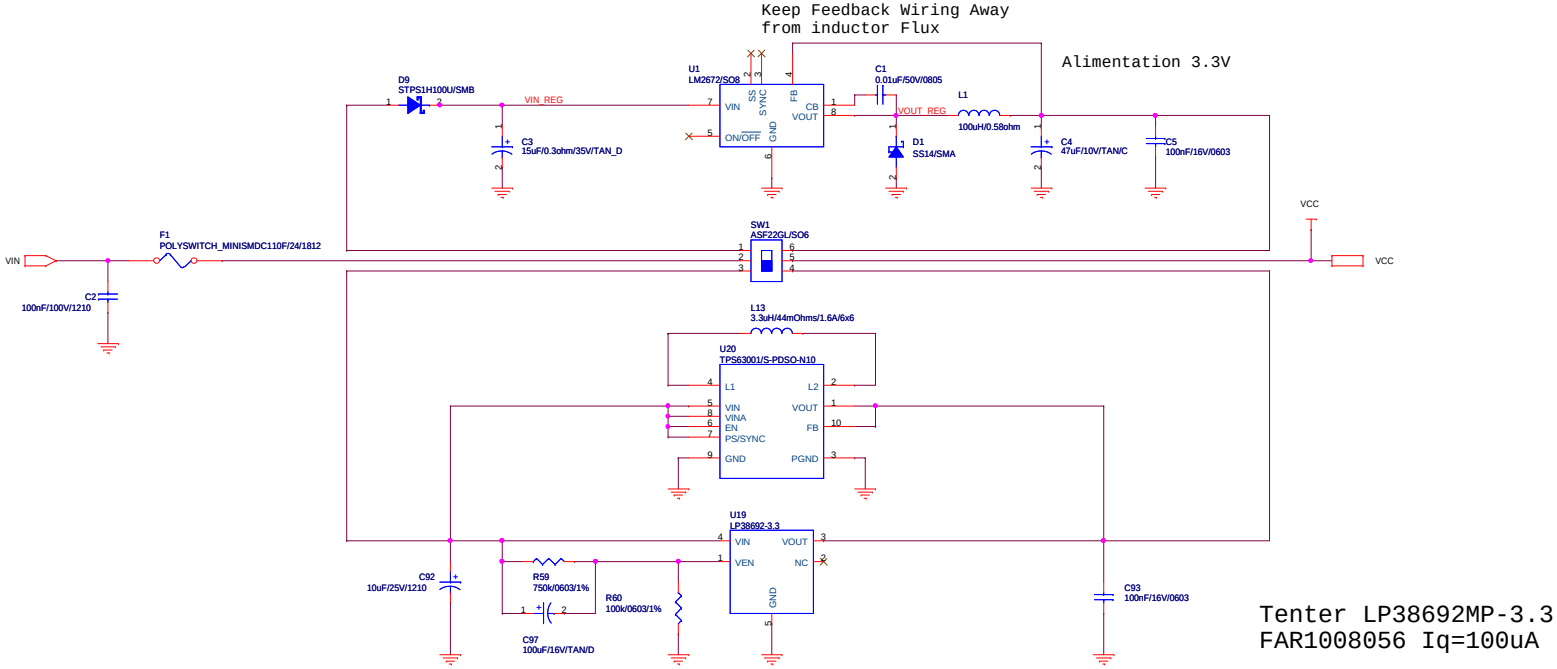
Person B rät 81.
Bingo!

DETECTEUR VEHICULE



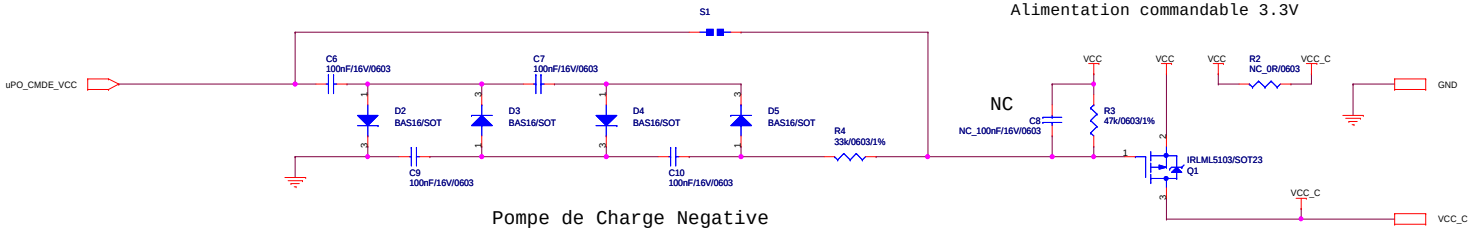
Title	DETECTEUR VEHICULE	
Size B	Document Number <Doc>	Rev <RevCode>
Date: Tuesday, November 20, 2007	Sheet 1 of 15	

Alimentation 3.3V/1000mA



Tenter LP38692MP-3.3
FAR1008056 Iq=100uA

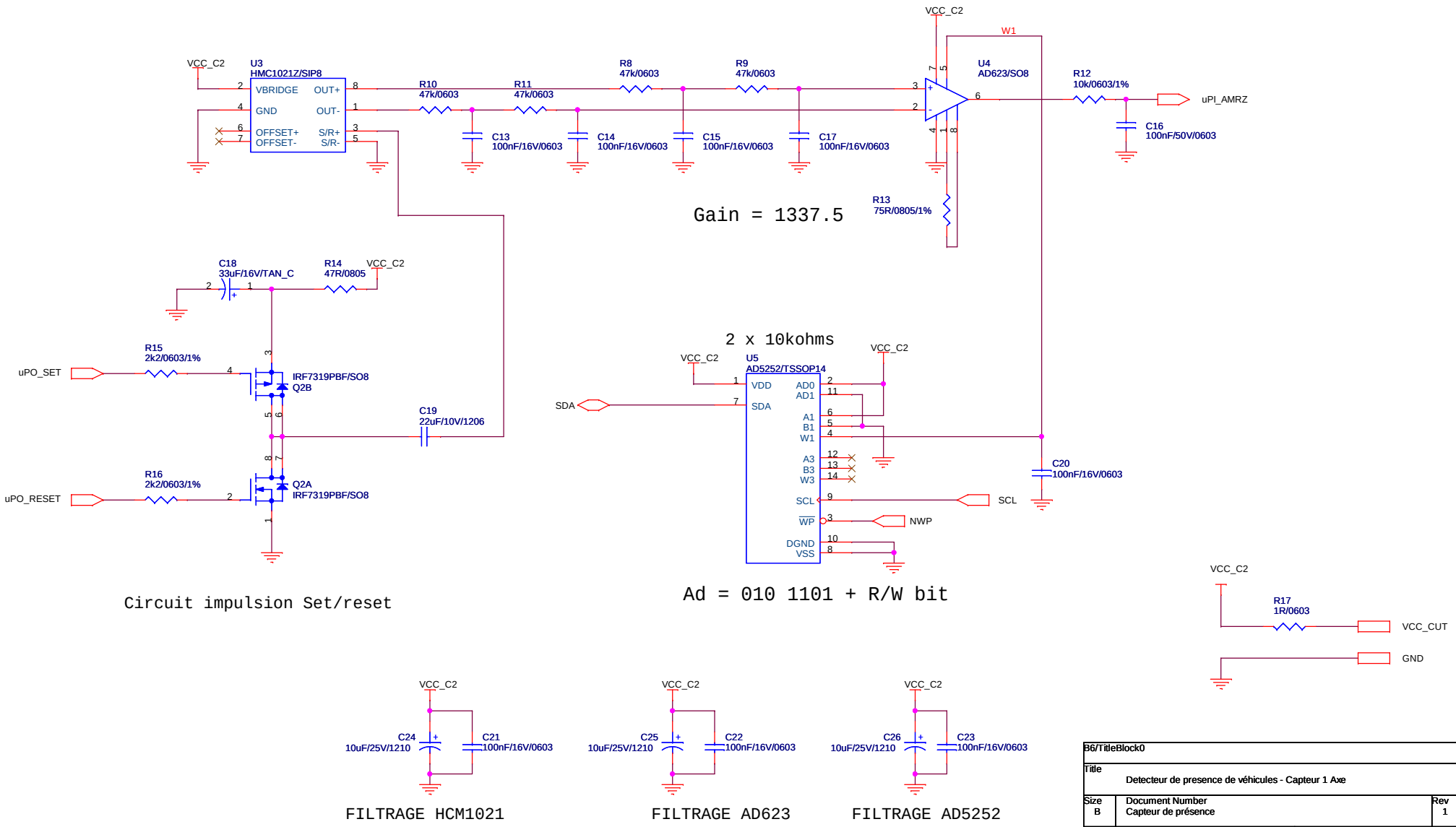
VENL=0.4V VENH=1.8V
Seuil = 3.4 V



Pompe de Charge Negative

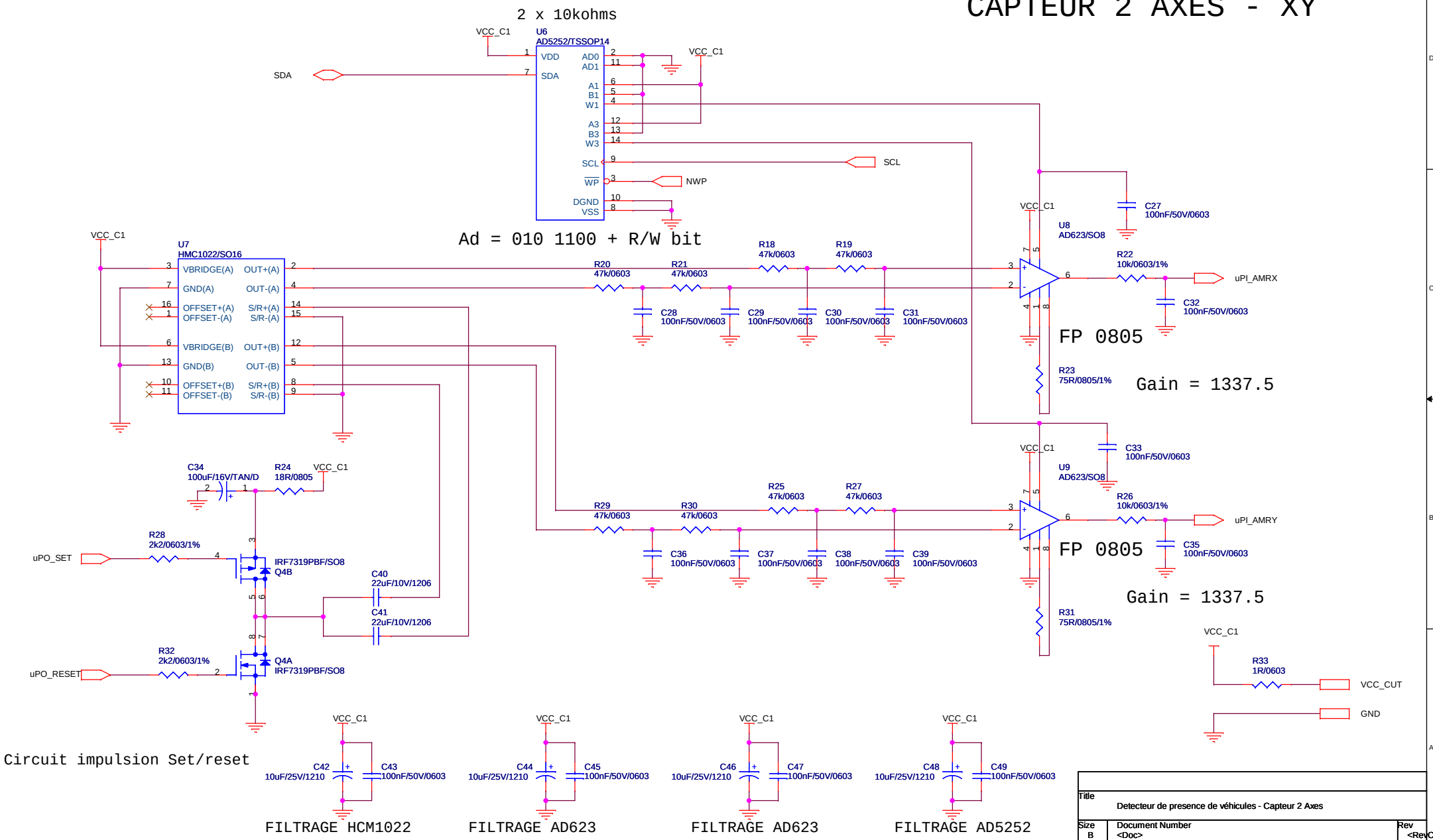
B8/TitleBlock0		
Title	Detecteur de presence de véhicules	
Size C	Document Number Alimentations	Rev 1
Date: Monday, November 26, 2007	Sheet 2 of 15	

DETECTEUR PRESENCE VEHICULE
CAPTEUR 1 AXE - Z



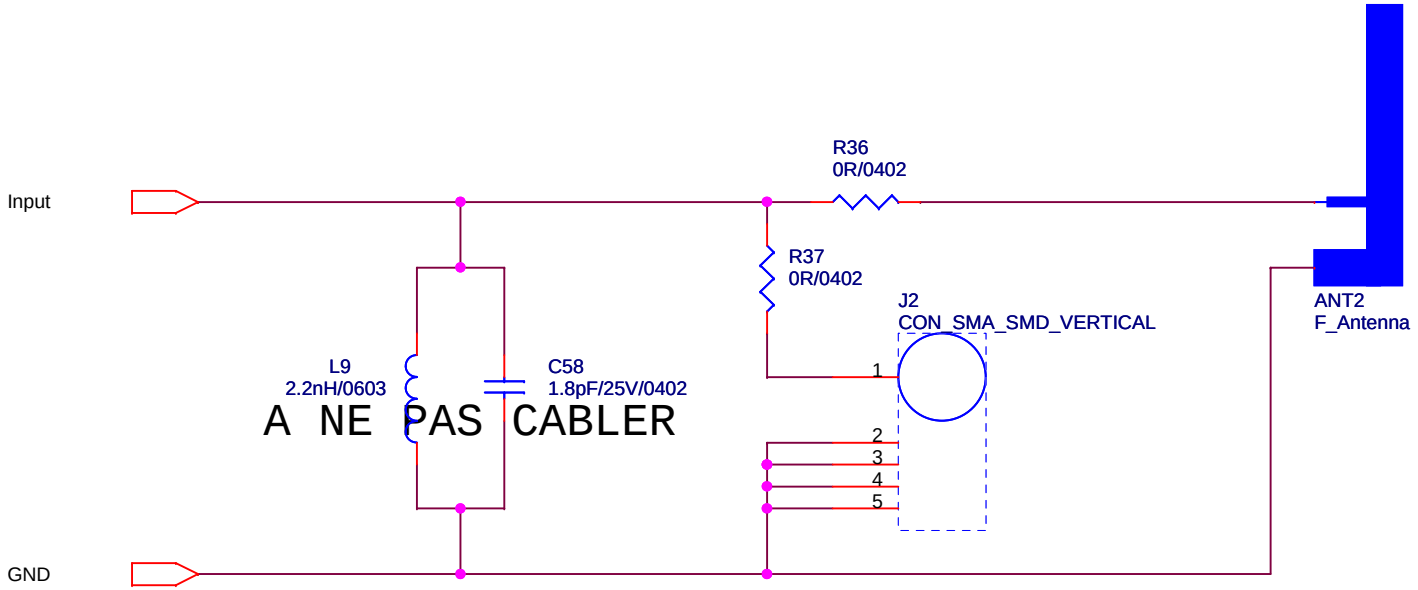
B6/TitleBlock0		
Title	Detecteur de presence de véhicules - Capteur 1 Axe	
Size B	Document Number Capteur de présence	Rev 1
Date: Tuesday, November 20, 2007	Sheet 4 of 15	

DETECTEUR PRESENCE VEHICULE
CAPTEUR 2 AXES - XY



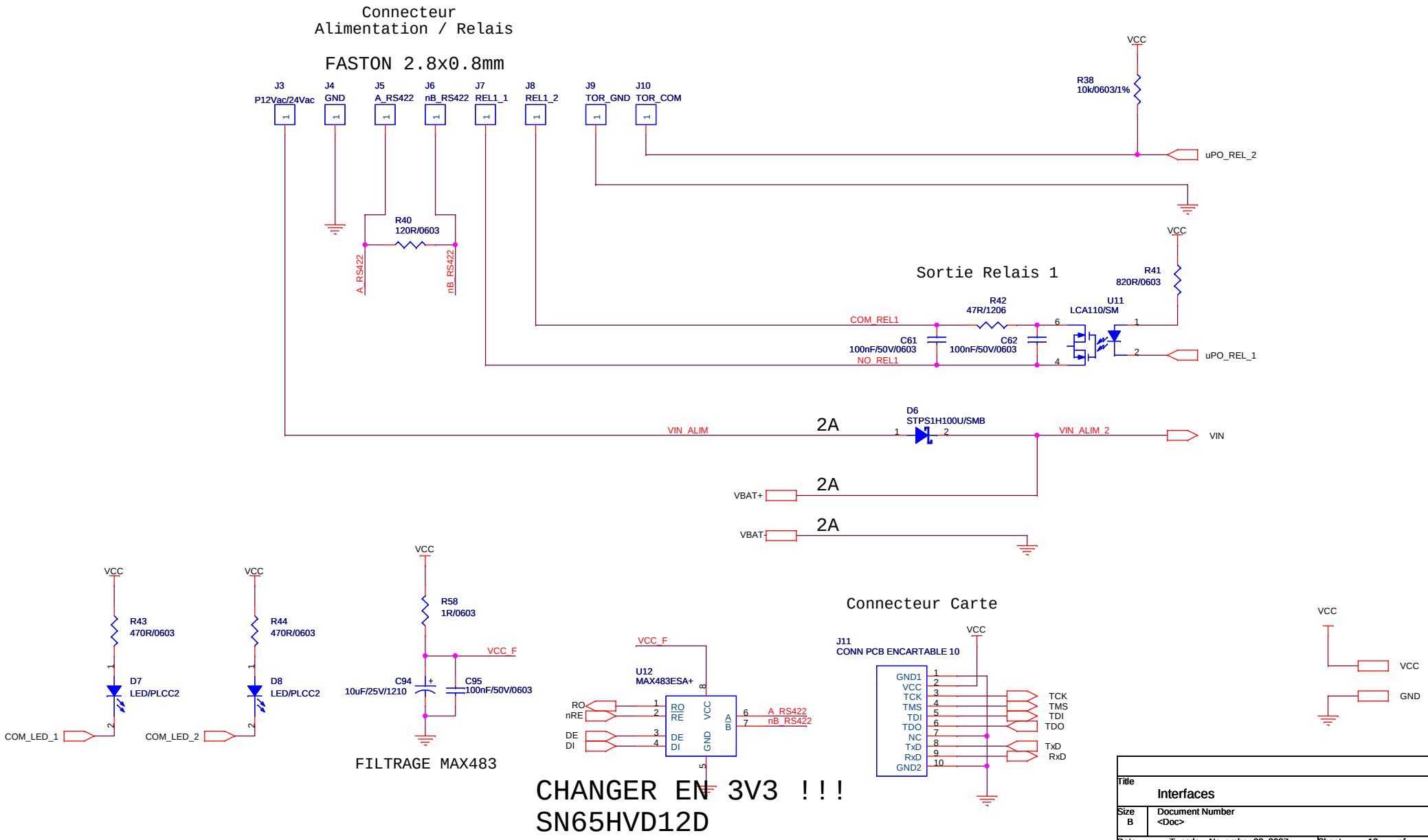
Title	Detecteur de presence de véhicules - Capteur 2 Axes	
Size B	Document Number <Doc>	Rev <RevCode>
Date: Friday, October 19, 2007	Sheet 5 of 15	

F ANTENNA



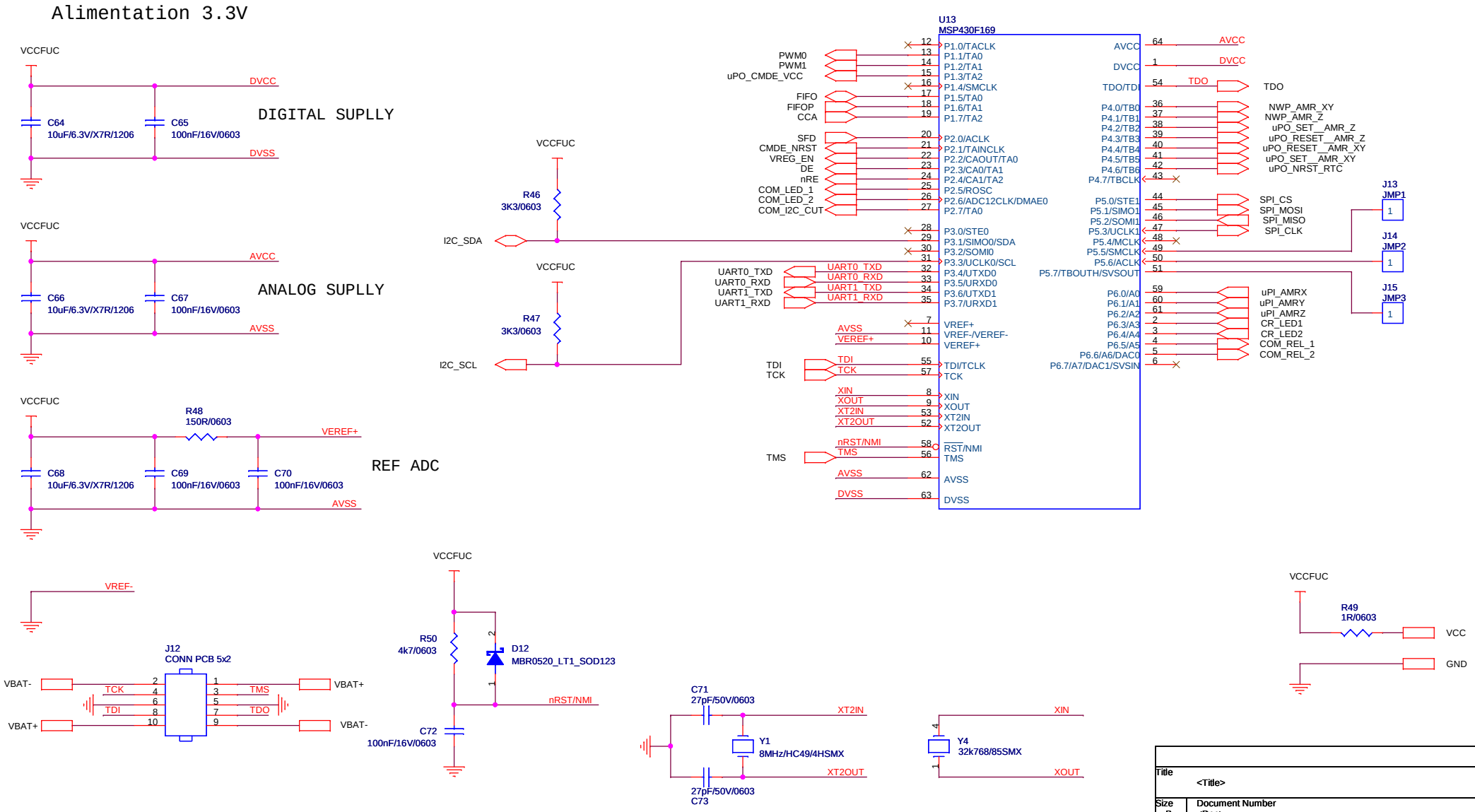
Title	F ANTENNA	
Size A	Document Number <Doc>	Rev <RevCode>
Date: Friday, October 19, 2007	Sheet 9 of 15	

INTERFACES



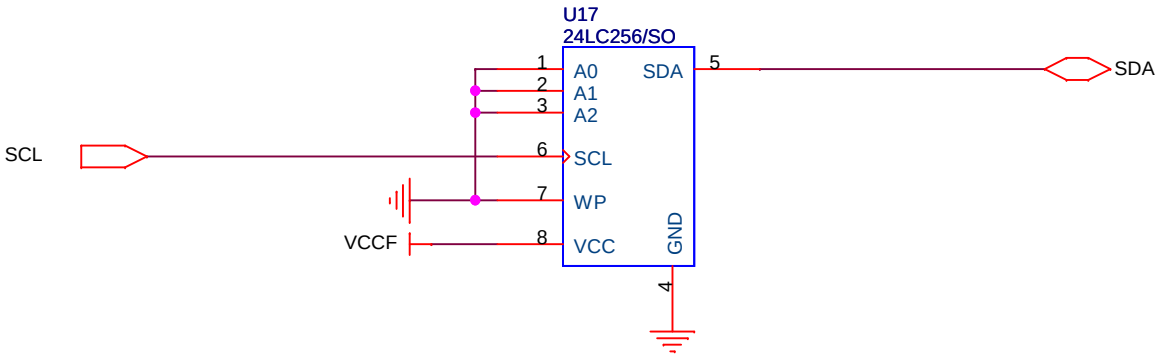
Title	Interfaces		
Size B	Document Number <Doc>		Rev <RevCode>
Date: Tuesday, November 20, 2007	Sheet 10 of 15		

MICROCONTROLEUR

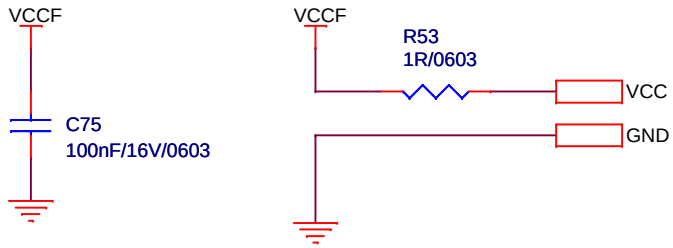


Title	<Title>	
Size B	Document Number <Doc>	Rev <RevCode>
Date: Friday, October 19, 2007	Sheet 11 of 15	

SERIAL EEPROM

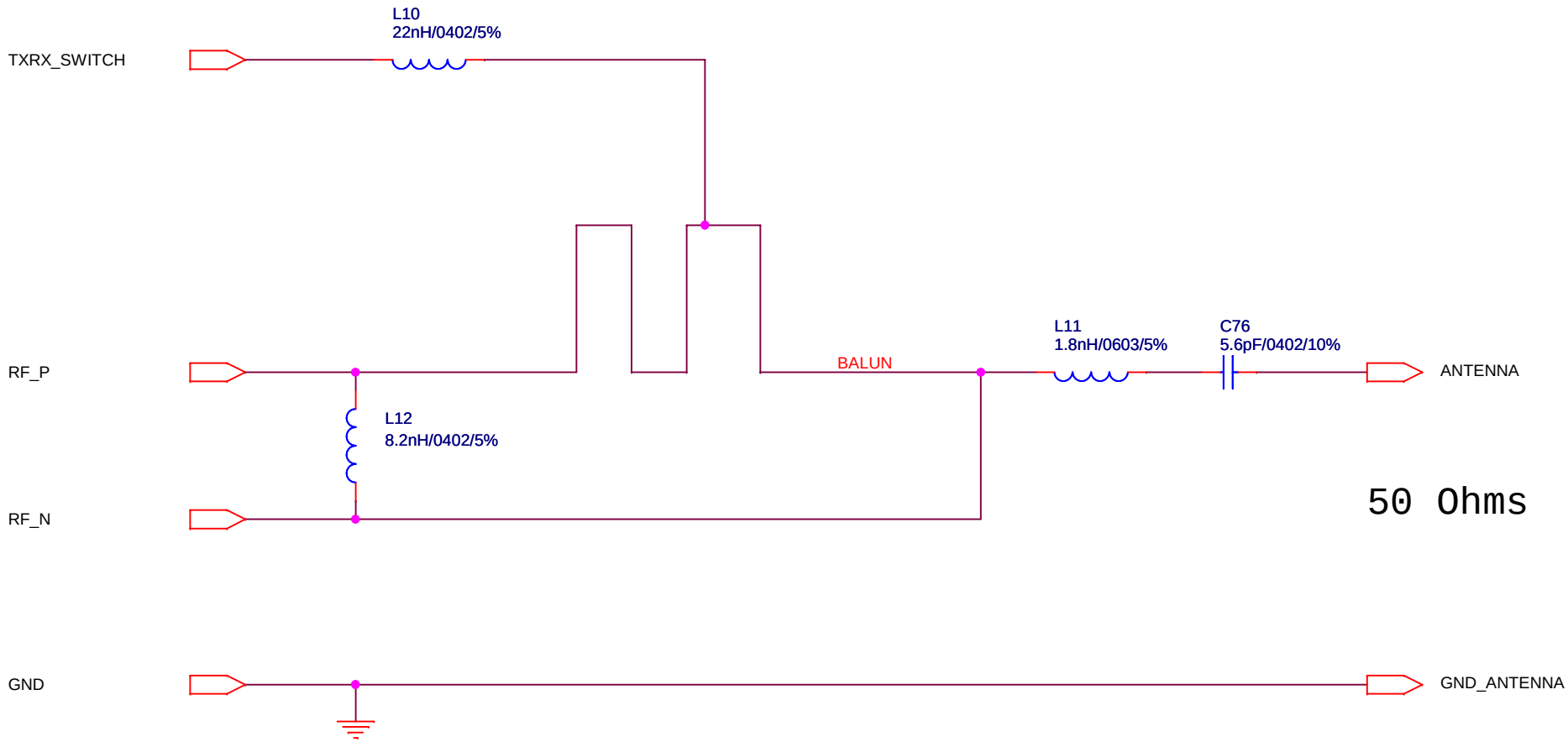


AD : 1010 000+R/W



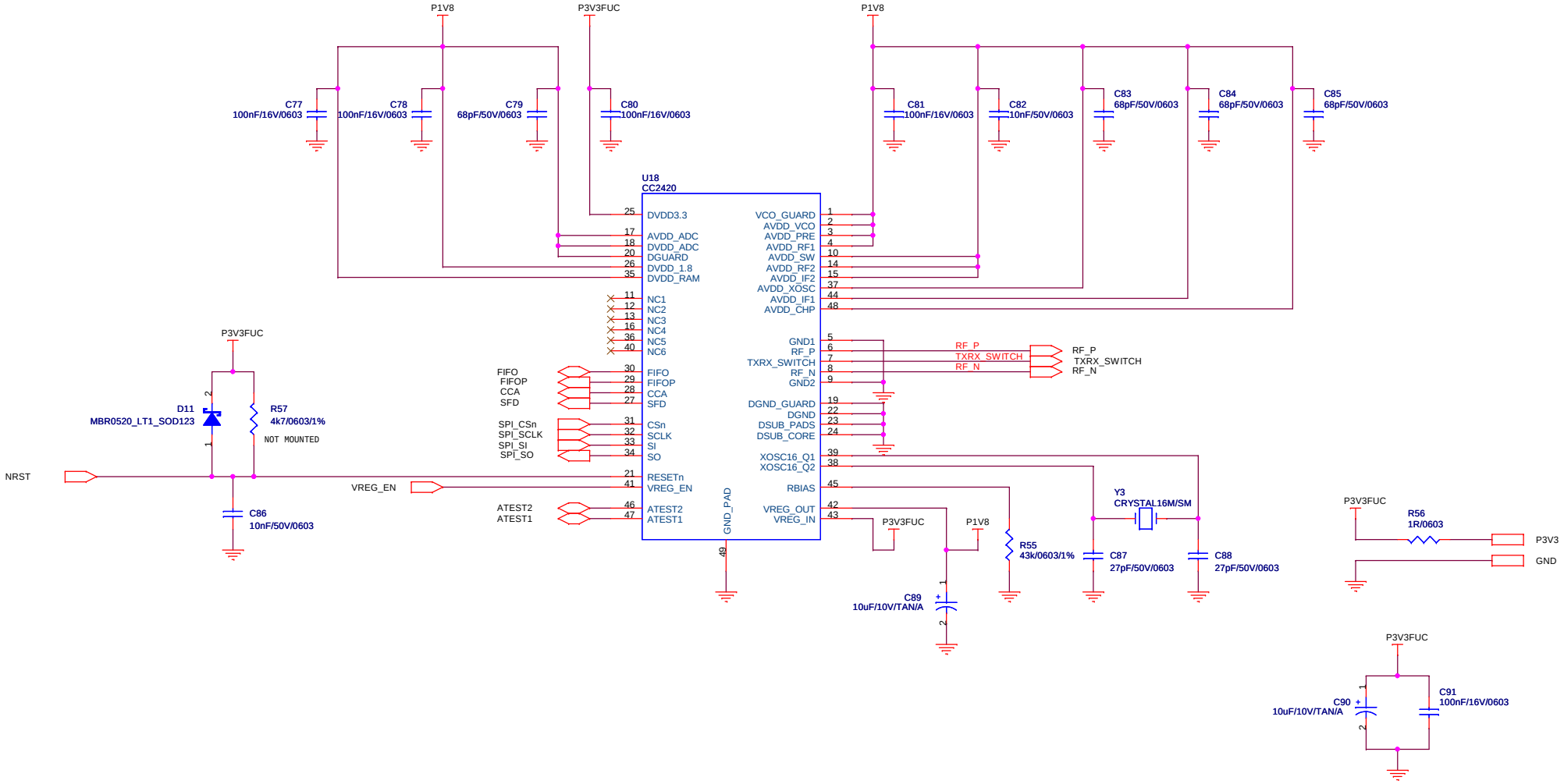
Title	SERIAL EEPROM	
Size A	Document Number <Doc>	Rev 1
Date: Tuesday, November 20, 2007	Sheet 13 of 15	

TRANSMISSION LINE BALUN



Title	Discrete Balun	
Size A	Document Number <Doc>	Rev <RevCode>
Date: Friday, October 19, 2007	Sheet 14 of 15	

ZIGBEE CC2420



Title	ZIGBEE	
Size A3	Document Number <Doc>	Rev <RevCode>
Date: Friday, October 19, 2007	Sheet 15 of 15	

```
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Headerfile amr_autoconf.h                                                        */
/*                                                                                   */
/*  Use the ADC to work with the sensors                                             */
/*-----------------------------------------------------------------------------------*/

#ifndef _AMR_AUTOCONF_H_
#define _AMR_AUTOCONF_H_

/******** FUNCTIONS ***************************************************************/
extern void amr_init_calibration(void);     // Configures the axis offsets.
extern void amr_average(void);              // Calculate the avarage value of any axis
extern void amr_detection(void);            // Calculate the parking place state
extern void amr_dyn_calib (void);           // Calculalate a constantly approximation to
                                            //  the mean value
#endif // _AMR_AUTOCONF_H_

```

```c
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Classfile amr_autoconf.c                                                         */
/*                                                                                   */
/*  Configures the sensors.                                                          */
/*-----------------------------------------------------------------------------------*/

/******** INCLUDES ***************************************************************/
#include "amr.h"
#include "czame.h"
#include "pwm.h"
#include "i2c.h"
#include "amr_autoconf.h"
#include "definitions.h"
#include "variables.h"
#include <io.h>
#include <stdbool.h>


/******** INTERN CONSTANTS *******************************************************/
#define SEC_AVR                 20

#define AUTOCONF_CYCLES         8
#define AUTOCONF_THRESHOLD_H    3584
#define AUTOCONF_THRESHOLD_L    512
#define AUTOCALIB_TIME          10
#define AUTOCALIB_OFFSET_CORR   1

#define TRIGGER_THRESHOLD_H     44
#define DETECTION_THRESHOLD_H   44
#define DETECTION_THRESHOLD_L   44


/******** STATIC FILE VARIABLES **************************************************/
static uint32_t x_avg_256_8 = 0;  //(512*256)
static uint32_t y_avg_256_8 = 0;  //(512*256)
static uint32_t z_avg_256_8 = 0;  //(512*256)
static uint32_t x_avg_256_32 = 0; //(512*256)
static uint32_t y_avg_256_32 = 0; //(512*256)
static uint32_t z_avg_256_32 = 0; //(512*256)

uint32_t x_avg_8;
uint32_t x_avg_32;
uint32_t x_avg_64;
uint32_t y_avg_8;
uint32_t y_avg_32;
uint32_t y_avg_64;
uint32_t z_avg_8;
uint32_t z_avg_32;
uint32_t z_avg_64;


/******** FUNCTIONS **************************************************************/

/***********************************************************************************/
/* Calculate the avarage value of any axis.                                        */
/*                                                                                 */
/* @DATE      : 26.11.2007                                                         */
/* @AUTHOR    : Ravel                                                              */
/* @INPUT     : -                                                                  */
/* @OUPUT     : -                                                                  */
/* @REVISION  : v1.0                                                               */
/***********************************************************************************/
void amr_average(void)
{
  //--- Define static variables

  //  static uint32_t x_avg_256_64 = 131072; //(512*256)
  //  static uint32_t y_avg_256_64 = 131072; //(512*256)
  //  static uint32_t z_avg_256_64 = 131072; //(512*256)

  //--- Calculate the avarages
  //.. Average 8
  /* x_avg_256_8 = (x_avg_256_8*7 + x*256) / 8 */
  /* x_avg_8 = x_avg_256_8 / 256 */
```

```c

  x_avg_256_8 = ((x_avg_256_8<<3) - x_avg_256_8) + (x<<8);
  x_avg_256_8 = x_avg_256_8>>3;
  x_avg_8     = (x_avg_256_8 >> 8);

  y_avg_256_8 = ((y_avg_256_8<<3) - y_avg_256_8) + (y<<8);
  y_avg_256_8 = y_avg_256_8>>3;
  y_avg_8     = (y_avg_256_8 >> 8);

  z_avg_256_8 = ((z_avg_256_8<<3) - z_avg_256_8) + (z<<8);
  z_avg_256_8 = z_avg_256_8>>3;
  z_avg_8     = (z_avg_256_8 >> 8);

  //.. Average 32  --> used in car detection
  /* x_avg_256_32 = (x_avg_256_32*31 + x*256) / 32 */
  /* x_avg_32 = x_avg_256_32 / 256 */

  x_avg_256_32 = ((x_avg_256_32<<5) - x_avg_256_32) + (x<<8);
  x_avg_256_32 = x_avg_256_32>>5;
  x_avg_32     = (x_avg_256_32 >> 8);

  y_avg_256_32 = ((y_avg_256_32<<5) - y_avg_256_32) + (y<<8);
  y_avg_256_32 = y_avg_256_32>>5;
  y_avg_32     = (y_avg_256_32 >> 8);

  z_avg_256_32 = ((z_avg_256_32<<5) - z_avg_256_32) + (z<<8);
  z_avg_256_32 = z_avg_256_32>>5;
  z_avg_32     = (z_avg_256_32 >> 8);

  //  //.. Average 256  --> used in dynamic auto calibration
  //  /* x_avg_256_64 = (x_avg_256_64*255 + x*256) / 256 */
  //  /* x_avg_256 = x_avg_256_64 / 256 */
  //
  //  x_avg_256_64 = ((x_avg_256_64<<8) - x_avg_256_64) + (x<<8);
  //  x_avg_256_64 = x_avg_256_64>>8;
  //  x_avg_64     = (x_avg_256_64 >> 8);
  //
  //  y_avg_256_64 = ((y_avg_256_64<<8) - y_avg_256_64) + (y<<8);
  //  y_avg_256_64 = y_avg_256_64>>8;
  //  y_avg_64     = (y_avg_256_64 >> 8);
  //
  //  z_avg_256_64 = ((z_avg_256_64<<8) - z_avg_256_64) + (z<<8);
  //  z_avg_256_64 = z_avg_256_64>>8;
  //  z_avg_64     = (z_avg_256_64 >> 8);
}

/*****************************************************************************************/
/* Calculate the parking place state.                                                    */
/*                                                                                       */
/* @DATE      : 26.11.2007                                                               */
/* @AUTHOR    : Ravel                                                                    */
/* @INPUT     : -                                                                        */
/* @OUPUT     : -                                                                        */
/* @REVISION  : v1.0                                                                     */
/*****************************************************************************************/
void amr_detection(void)
{
  static int16_t counter = 0;
  static uint8_t counter_thld = 0;
  static bool triggerState = false;
  static uint8_t detect_threshold = DETECTION_THRESHOLD_H;

  gState_s = (ABS((int)z_avg_32-512 + gZ_offset)
              +ABS((int)x_avg_32-512 + gX_offset)
                +ABS((int)y_avg_32-512 + gY_offset)) > detect_threshold;

  if(gState_s)
  {
    if(counter == (3*BPS))
      gState = true;
    else
      counter++;
  }
  else
  {
    counter = 0;
```

```
153     if(gState)
154     {
155       gState = false;
156       triggerState = false;
157       detect_threshold = DETECTION_THRESHOLD_H;
158     }
159
160     if(!triggerState)
161     {
162       triggerState=(ABS((int)z_avg_8-512 + gZ_offset)
163                     +ABS((int)x_avg_8-512 + gX_offset)
164                       +ABS((int)y_avg_8-512 + gY_offset)) > TRIGGER_THRESHOLD_H;
165
166       if(triggerState)
167       {
168         counter_thld = 5*BPS;
169         detect_threshold = DETECTION_THRESHOLD_L;
170       }
171     }
172
173     if(counter_thld > 0)
174     {
175       counter_thld--;
176     }
177     else
178     {
179       triggerState = false;
180       detect_threshold = DETECTION_THRESHOLD_H;
181     }
182   }
183
184   //.. Feed algorithm with ADC values to calculate the parking state
185   //   (--> If the parking state change the callback function 'notify' will be called)
186   //  czame_play(x+gX_offset,y+gY_offset,z+gZ_offset);
187 }
188
189 /*****************************************************************************************/
190 /* [static] - Make captures for the mean value initialisation -> calculate avarage.      */
191 /*                                                                                       */
192 /* @DATE      : 26.11.2007                                                               */
193 /* @AUTHOR    : Ravel                                                                    */
194 /* @INPUT     : -                                                                        */
195 /* @OUPUT     : -                                                                        */
196 /* @REVISION  : v1.0                                                                     */
197 /*****************************************************************************************/
198 static void amr_capture(void)
199 {
200   int16_t i;
201   amr_degauss();
202   pwm_on();
203   amr_start(BPS);
204   _BIS_SR(LPM3_bits+GIE);
205   amr_stop();
206   x_avg_256_32 = x<<8;
207   y_avg_256_32 = y<<8;
208   z_avg_256_32 = z<<8;
209   x_avg_256_8 = x_avg_256_32;
210   y_avg_256_8 = y_avg_256_32;
211   z_avg_256_8 = z_avg_256_32;
212   amr_start(BPS);
213     for(i = 0; i < (SEC_AVR*BPS); i++)
214     {
215       _BIS_SR(LPM3_bits+GIE);
216       amr_average();
217     }
218   amr_stop();
219   pwm_on();
220 }
221
222 /*****************************************************************************************/
223 /* Initialize the sensors of the best mean value (potentiometer offset).                 */
224 /*                                                                                       */
225 /* @DATE      : 26.11.2007                                                               */
226 /* @AUTHOR    : Ravel                                                                    */
227 /* @INPUT     : -                                                                        */
228 /* @OUPUT     : -                                                                        */
```

```
229 /* @REVISION  : v1.0                                                                    */
230 /*************************************************************************************/
231 void amr_init_calibration(void)
232 {
233   //--- Define variables
234   uint8_t i;                          // counter variable
235   int16_t ofs_x,ofs_y,ofs_z;              // axis offsets
236   struct { bool x, y, z; } found;
237   uint8_t ofsChange[AUTOCONF_CYCLES];
238
239   pwm_on();
240
241   amr_init();
242   i2c_init();
243
244   //--- Initialize variables
245   ofs_x = 128;                        // middle of range
246   ofs_y = 128;                        // middle of range
247   ofs_z = 128;                        // middle of range
248   found.x = false;
249   found.y = false;
250   found.z = false;
251   for(i = 0; i < AUTOCONF_CYCLES; i++)
252   {
253     ofsChange[i] = 128 >> i;
254   }
255
256
257   amr_degauss();
258   amr_degauss();
259   amr_degauss();
260   amr_degauss();
261   amr_degauss();
262   amr_degauss();
263
264   //--- Search best offset for potentiometers
265   i = 1;
266   led_2_on();
267   do
268   {
269     pwm_on();
270     amr_stop();
271     //.. Set offset
272     amr_set_x_ofs(ofs_x);
273     amr_set_y_ofs(ofs_y);
274     amr_set_z_ofs(ofs_z);
275
276     //.. Call amr_capture : search stability value
277     amr_capture();
278
279     //.. Check if stability is reached
280     if (!found.x)
281     {
282       if ((x_avg_32 <= AUTOCONF_THRESHOLD_H) && (x_avg_32 >= AUTOCONF_THRESHOLD_L))
283       {
284         found.x = true;
285         gX_offset = 512 - x_avg_32;
286       }
287       else
288       {
289         if (x_avg_32 < AUTOCONF_THRESHOLD_L)
290           ofs_x = ofs_x + ofsChange[i];
291         else
292           ofs_x = ofs_x - ofsChange[i];
293       }
294     }
295
296     if (!found.y)
297     {
298       if ((y_avg_32 <= AUTOCONF_THRESHOLD_H) && (y_avg_32 >= AUTOCONF_THRESHOLD_L))
299       {
300         found.y = true;
301         gY_offset = 512 - y_avg_32;
302       }
303       else
304       {
```

```
305         if (y_avg_32 < AUTOCONF_THRESHOLD_L)
306           ofs_y = ofs_y + ofsChange[i];
307         else
308           ofs_y = ofs_y - ofsChange[i];
309       }
310     }
311
312     if (!found.z)
313     {
314       if ((z_avg_32 <= AUTOCONF_THRESHOLD_H) && (z_avg_32 >= AUTOCONF_THRESHOLD_L))
315       {
316         found.z = true;
317         gZ_offset = 512 - z_avg_32;
318       }
319       else
320       {
321         if (z_avg_32 < AUTOCONF_THRESHOLD_L)
322           ofs_z = ofs_z + ofsChange[i];
323         else
324           ofs_z = ofs_z - ofsChange[i];
325       }
326     }
327
328     //.. Leave loop when all values are found
329     if ((found.x) && (found.y) && (found.z))
330       break;
331
332     //.. Increment counter
333     i++;
334   } while (i != AUTOCONF_CYCLES);
335
336   if (!((found.x) && (found.y) && (found.z)))
337   {
338     amr_capture();
339     gX_offset = 512 - x_avg_32;
340     gY_offset = 512 - y_avg_32;
341     gZ_offset = 512 - z_avg_32;
342   }
343
344   amr_start(BPS);
345
346   led_2_off();
347 }
348
349 /*****************************************************************************************/
350 /* Calculalate a constantly approximation to the mean value.                             */
351 /*                                                                                       */
352 /* @DATE      : 26.11.2007                                                               */
353 /* @AUTHOR    : Ravel                                                                    */
354 /* @INPUT     : -                                                                        */
355 /* @OUPUT     : -                                                                        */
356 /* @REVISION  : v1.0                                                                     */
357 /*****************************************************************************************/
358 void amr_dyn_calib (void)
359 {
360   //--- Definition and initialisation of STATIC variables
361   static uint16_t x_counter = BPS * AUTOCALIB_TIME;
362   static uint16_t y_counter = BPS * AUTOCALIB_TIME;
363   static uint16_t z_counter = BPS * AUTOCALIB_TIME;
364   static uint16_t x_pos = 512;
365   static uint16_t y_pos = 512;
366   static uint16_t z_pos = 512;
367
368   //--- If a car is detect leave this function and reset the counters
369   if(gState)
370   {
371     x_counter = BPS * AUTOCALIB_TIME;
372     y_counter = BPS * AUTOCALIB_TIME;
373     z_counter = BPS * AUTOCALIB_TIME;
374     return;
375   }
376
377   //--- Decrement the counters
378   x_counter--;
379   y_counter--;
380   z_counter--;
```

```c

  if (x_counter == 0)
  {
    x_pos = x_avg_32 + gX_offset;

    if (x_pos > 512)
      gX_offset = gX_offset - AUTOCALIB_OFFSET_CORR;
    else
      gX_offset = gX_offset + AUTOCALIB_OFFSET_CORR;

    x_counter = BPS * AUTOCALIB_TIME;
  }

  if (y_counter == 0)
  {
    y_pos = y_avg_32 + gY_offset;

    if (y_pos > 512)
      gY_offset = gY_offset - AUTOCALIB_OFFSET_CORR;
    else
      gY_offset = gY_offset + AUTOCALIB_OFFSET_CORR;

    y_counter = BPS * AUTOCALIB_TIME;
  }

  if (z_counter == 0)
  {
    z_pos = z_avg_32 + gZ_offset;

    if (z_pos > 512)
      gZ_offset = gZ_offset - AUTOCALIB_OFFSET_CORR;
    else
      gZ_offset = gZ_offset + AUTOCALIB_OFFSET_CORR;

    z_counter = BPS * AUTOCALIB_TIME;
  }
}

```

```c
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Headerfile amr.h                                                                 */
/*                                                                                   */
/*  Use the ADC to work with the sensors                                             */
/*-----------------------------------------------------------------------------------*/

#ifndef _AMR_H_
#define _AMR_H_

/******** INCLUDES ***************************************************************/
#include <stdint.h>


/******** INTERN GLOBAL VARIABLE *************************************************/
extern volatile uint32_t x, y, z;


/******** FUNCTIONS **************************************************************/
void    amr_init      (void);               // Initialize the ADC to use the sensors.
void    amr_start     (uint8_t bps);        // Start ADC sampling. [Using TimerB]
void    amr_stop      (void);               // Stop ADC sampling. [Using TimerB]
uint8_t amr_get_x_ofs (void);               // Get X axis offset.
void    amr_set_x_ofs (uint8_t x_ofs);      // Set X axis offset.
uint8_t amr_get_y_ofs (void);               // Get Y axis offset.
void    amr_set_y_ofs (uint8_t y_ofs);      // Set Y axis offset.
uint8_t amr_get_z_ofs (void);               // Get Z axis offset.
void    amr_set_z_ofs (uint8_t z_ofs);      // Set Z axis offset.
void    amr_degauss   (void);               // Sensors reconfiguration (degauss).

#endif //_AMR_H_
```

```c
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Classfile amr.c                                                                  */
/*                                                                                   */
/*  Use the ADC to work with the sensors                                             */
/*-----------------------------------------------------------------------------------*/

/******** INCLUDES ***************************************************************/
#include "amr.h"
#include "pwm.h"
#include "definitions.h"
#include "i2c.h"
#include "sleep.h"
#include <io.h>
#include <signal.h>


/******** INTERN CONSTANTS *******************************************************/
#define I2C_AMR_XY    0x2C                  // I2C-address of 2 the axis sensor
#define I2C_AMR_Z     0x2D                  // I2C-address of 1 the axis sonsor

#define AMR_XY_SET    BIT5                  // <P4.5>
#define AMR_XY_RESET  BIT4                  // <P4.4>
#define AMR_XY_NWP    BIT0                  // <P4.0>
#define AMR_Z_SET     BIT2                  // <P4.2>
#define AMR_Z_RESET   BIT3                  // <P4.3>
#define AMR_Z_NWP     BIT1                  // <P4.1>


/******** INTERN GLOBAL VARIABLES ************************************************/
volatile uint32_t x, y, z;                  // variables with values of the 3 axis


/******** INTERN STRUCTORGRAMS ***************************************************/
struct { uint8_t x, y, z; } ofs;            // sensor offset structogramm


/******** FUNCTIONS **************************************************************/

/***********************************************************************************/
/* Set X axis offset.                                                              */
/*                                                                                 */
/* @DATE      : 08.11.2007                                                         */
/* @AUTHOR    : Ravel                                                              */
/* @INPUT     : uint8_t i > offset value                                           */
/* @OUPUT     : -                                                                  */
/* @REVISION  : v1.0                                                               */
/***********************************************************************************/
void amr_set_x_ofs (uint8_t i)
{
  P4OUT |= AMR_XY_NWP;                       // Set sensor NWP as HIGH => 1

  i2c_write(I2C_AMR_XY, 2, 0x01, ofs.y=i);  // Write in RDAC  (potentiometer)
  i2c_write(I2C_AMR_XY, 2, 0x21, ofs.y=i);  // Write in EEMEM (potentiometer)
  DELAY(100*MSEC);

  P4OUT &= ~(AMR_XY_NWP);                    // Set sensor NWP as LOW => 0

}

/***********************************************************************************/
/* Set Y axis offset.                                                              */
/*                                                                                 */
/* @DATE      : 08.11.2007                                                         */
/* @AUTHOR    : Ravel                                                              */
/* @INPUT     : uint8_t i > offset value                                           */
/* @OUPUT     : -                                                                  */
/* @REVISION  : v1.0                                                               */
/***********************************************************************************/
void amr_set_y_ofs (uint8_t i)
{
  P4OUT |= AMR_XY_NWP;                       // set sensor NWP as HIGH => 1

  i2c_write(I2C_AMR_XY, 2, 0x03, ofs.y=i);  // Write in RDAC  (potentiometer)
  i2c_write(I2C_AMR_XY, 2, 0x23, ofs.y=i);  // Write in EEMEM (potentiometer)
```

```
  DELAY(100*MSEC);

  P4OUT &= ~(AMR_XY_NWP);                    // Set sensor NWP as LOW => 0
}

/*****************************************************************************/
/* Set Z axis offset.                                                        */
/*                                                                           */
/* @DATE      : 08.11.2007                                                   */
/* @AUTHOR    : Ravel                                                        */
/* @INPUT     : uint8_t i > offset value                                     */
/* @OUPUT     : -                                                            */
/* @REVISION  : v1.0                                                         */
/*****************************************************************************/
void amr_set_z_ofs (uint8_t i)
{
  P4OUT |= AMR_Z_NWP;                        // Set sensor NWP as HIGH => 1

  i2c_write(I2C_AMR_Z, 2, 0x01, ofs.z=i);    // Write in RDAC  (potentiometer)
  i2c_write(I2C_AMR_Z, 2, 0x21, ofs.z=i);    // Write in EEMEM (potentiometer)
  DELAY(100*MSEC);

  P4OUT &= ~(AMR_Z_NWP);                     // Set sensor NWP as LOW => 0
}

/*****************************************************************************/
/* Get X axis offset.                                                        */
/*                                                                           */
/* @DATE      : 08.11.2007                                                   */
/* @AUTHOR    : Ravel                                                        */
/* @INPUT     : -                                                            */
/* @OUPUT     : uint8_t < offset value                                       */
/* @REVISION  : v1.0                                                         */
/*****************************************************************************/
uint8_t amr_get_x_ofs (void)
{
  return ofs.x;
}

/*****************************************************************************/
/* Get Y axis offset.                                                        */
/*                                                                           */
/* @DATE      : 08.11.2007                                                   */
/* @AUTHOR    : Ravel                                                        */
/* @INPUT     : -                                                            */
/* @OUPUT     : uint8_t < offset value                                       */
/* @REVISION  : v1.0                                                         */
/*****************************************************************************/
uint8_t amr_get_y_ofs (void)
{
  return ofs.y;
}

/*****************************************************************************/
/* Get Z axis offset.                                                        */
/*                                                                           */
/* @DATE      : 08.11.2007                                                   */
/* @AUTHOR    : Ravel                                                        */
/* @INPUT     : -                                                            */
/* @OUPUT     : uint8_t < offset value                                       */
/* @REVISION  : v1.0                                                         */
/*****************************************************************************/
uint8_t amr_get_z_ofs (void)
{
  return ofs.z;
}

/*****************************************************************************/
/* Initialize the ADC to use the sensors.                                    */
/*                                                                           */
/* @DATE      : 08.11.2007                                                   */
/* @AUTHOR    : Ravel                                                        */
/* @INPUT     : -                                                            */
/* @OUPUT     : -                                                            */
/* @REVISION  : v1.0                                                         */
/*****************************************************************************/
```

```c
153 void amr_init (void)
154 {
155   //--- Configuration of write protect pin of the sensors
156   //    (if this pin is HIGH it's no more possible to write on the sensor)
157   P4DIR |= (AMR_XY_NWP + AMR_Z_NWP);        // set sensor NWP pins as output
158   P4OUT &= ~(AMR_XY_NWP + AMR_Z_NWP);       // set sensor NWP as low => 0
159
160   //--- Configuration of the Analog/Digital Converter (ADC)
161   P6SEL |= BIT0 + BIT1 + BIT2;              // Enable A/D inputs (A0=X, A1=Y, A2=Z)
162   ADC12CTL0 = ADC12ON + MSC + SHT0_2;
163   ADC12CTL1 = SHP + CONSEQ_1;
164   ADC12MCTL0 = INCH_0;                      // Channel=A0
165   ADC12MCTL1 = INCH_1;                      // Channel=A1
166   ADC12MCTL2 = INCH_2 + EOS;                // Channel=A2, end seq.
167   ADC12IE = BIT2;                           // Enable ADCIFG.2
168   ADC12CTL0 |= ENC;                         // Enable conversions
169 }
170
171 /*****************************************************************************************/
172 /* Sensors reconfiguration (degauss).                                                    */
173 /*                                                                                       */
174 /* @DATE      : 08.11.2007                                                               */
175 /* @AUTHOR    : Ravel                                                                    */
176 /* @INPUT     : -                                                                        */
177 /* @OUPUT     : -                                                                        */
178 /* @REVISION  : v1.0                                                                     */
179 /*****************************************************************************************/
180 void amr_degauss (void)
181 {
182   // ATTENTION: Do NOT set SET=0 et RESET=1 at the same time !
183
184   // Start         S=1 R=1, Wait 100ms     >>> T0
185   // Toggle-Reset  S=1 R=0, Wait  43us     >>> T1
186   // Toggle-Set    S=0 R=0, Wait 100ms     >>> T2
187   // Toggle-Set    S=1 R=0, Wait  43us     >>> T3
188   // Toggle-Reset  S=1 R=1, Wait 200ms     >>> T4
189   // Stop          S=1 R=1
190
191   //          ------------------                      ------------------------
192   //   RESET                   |                      |
193   //                           ------------------------
194   //
195   //          ... |     T0     | T1 |     T2     | T3 |        T4        | ...
196   //
197   //          -----------------------            ----------------------------
198   //   SET                          |            |
199   //                                --------------
200
201
202   amr_stop();                               // Stops the reading the sensors
203
204   _DINT();                                  // Disable the interrupts
205
206   pwm_on();                                 // Turn on the PWM
207
208   P4OUT |= (AMR_XY_SET + AMR_Z_SET + AMR_XY_RESET + AMR_Z_RESET); // S=1 R=1
209   P4DIR |=  (AMR_XY_RESET + AMR_Z_RESET + AMR_XY_SET + AMR_Z_SET );
210   DELAY(100*MSEC);                          // Wait 100 ms
211
212   P4OUT &= ~( AMR_XY_RESET + AMR_Z_RESET);  // S=1 R=0
213   //DELAY(10);                                // Wait 43 us
214
215   P4OUT &= ~(AMR_XY_SET + AMR_Z_SET);       // S=0 R=0
216   DELAY(100*MSEC);                          // Wait 100 ms
217
218   pwm_off();                                // Turn off the PWM
219
220   P4OUT |= (AMR_XY_SET + AMR_Z_SET);        // S=1 R=0
221   //DELAY(10);                                // Wait 43 us
222
223   P4OUT |= (AMR_XY_RESET + AMR_Z_RESET);    // S=1 R=1
224
225   _EINT();                                  // Enable the interrupts
226
227   amr_start(BPS);                           // Restarts the reading the sensors
228
```

```c
  DELAY(200*MSEC);                          // Wait 200 ms
}

/*****************************************************************************/
/* Start ADC sampling. [Using TimerB]                                        */
/*                                                                           */
/* @DATE      : 08.11.2007                                                   */
/* @AUTHOR    : Ravel                                                        */
/* @INPUT     : uint8_t bps > speed of sampling (samplings per second)       */
/* @OUPUT     : -                                                            */
/* @REVISION  : v1.0                                                         */
/*****************************************************************************/
void amr_start (uint8_t bps)
{
  TBCCTL0 &= ~CCIE;                         // CCR0 interrupt disabled
  TBCTL = TBSSEL_1 + ID_0 + MC_1;           // ACLK/1, Up mode (:= 32768 Hz)
  TBCCR0 = 32768/bps;
  TBCCTL0 |= CCIE;                          // CCR0 interrupt enabled
}

/*****************************************************************************/
/* Stop ADC sampling. [Using TimerB]                                         */
/*                                                                           */
/* @DATE      : 08.11.2007                                                   */
/* @AUTHOR    : Ravel                                                        */
/* @INPUT     : -                                                            */
/* @OUPUT     : -                                                            */
/* @REVISION  : v1.0                                                         */
/*****************************************************************************/
void amr_stop (void)
{
  TBCCTL0 &= ~CCIE;                         // CCR0 interrupt disabled
}

/*****************************************************************************/
/* [INTERRUPT] - TimerB interrupt                                            */
/*                                                                           */
/* @DATE      : 08.11.2007                                                   */
/* @AUTHOR    : Ravel                                                        */
/* @INPUT     : -                                                            */
/* @OUPUT     : -                                                            */
/* @REVISION  : v1.0                                                         */
/*****************************************************************************/
interrupt (TIMERB0_VECTOR) timerb_isr (void)
{
  pwm_on();
  ADC12CTL0 |= ADC12SC;
}

/*****************************************************************************/
/* [INTERRUPT] - ADC interrupt                                               */
/*                                                                           */
/* @DATE      : 08.11.2007                                                   */
/* @AUTHOR    : Ravel                                                        */
/* @INPUT     : -                                                            */
/* @OUPUT     : -                                                            */
/* @REVISION  : v1.0                                                         */
/*****************************************************************************/
interrupt (ADC12_VECTOR) adc12_isr (void)
{
  pwm_off();

  x = ADC12MEM0;                            // Assign ADC value to sensor variable x
  y = ADC12MEM1;                            // Assign ADC value to sensor variable y
  z = ADC12MEM2;                            // Assign ADC value to sensor variable z

  _BIC_SR_IRQ(LPM3_bits);                   // Clear SR_IRQ from low power mode 0  >>> AM
}

```

```c
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Headerfile cc2420.h                                                              */
/*                                                                                   */
/*  Class to work with the CC2420 Chipcon Transceiver                                */
/*-----------------------------------------------------------------------------------*/

#ifndef _CC2420_H_
#define _CC2420_H_

/******** INCLUDES ***************************************************************/
#include <stdint.h>


/******** FUNCTIONS **************************************************************/
void cc2420_init (void);                     // Initialition of the CC2420.

void cc2420_set_reg (uint8_t, uint16_t);    // Writes in a register.
uint16_t cc2420_get_reg (uint8_t);          // Reads from a register.
uint8_t cc2420_get_status (void);           // Read status of CC2420.
void cc2420_strobe (uint8_t);               // Calls a strobe register.

void cc2420_write_fifo (uint8_t, uint8_t*); // Write in CC2420 TX_FIFO to send a RF-Frame.
void cc2420_read_fifo (uint8_t, uint8_t*);  // Reads from CC2420 RX_FIFO

void cc2420_set_reg_ram (uint8_t, uint8_t, uint16_t*);  // Write to a RAM register.
void cc2420_get_reg_ram (uint8_t, uint8_t, uint16_t*);  // Reads from a RAM register.

extern void (*fifop_callback)(void);        // Callback function by flag FIFOP
extern void (*fifo_callback) (void);        // Callback function by flag FIFO

#endif  // endif _CC2420_H_
```

```c
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Classfile cc2420.c                                                               */
/*                                                                                   */
/*  Class to work with the CC2420 Chipcon Transceiver                                */
/*-----------------------------------------------------------------------------------*/

/******** INCLUDES ***************************************************************/
#include "cc2420.h"
#include "definitions.h"
#include "spi.h"
#include <io.h>
#include <signal.h>
#include <stdint.h>


/******** INTERN CONSTANTS *******************************************************/
#define CC2420_SELECT        CS0_ENABLE
#define CC2420_DESELECT      CS0_DISABLE

// Format for the access at registers
#define REG_WRITE_FORMAT(a)  a &= 0x3F      // Address
#define REG_READ_FORMAT(a)   a |= 0x40      // Address

// Format for the access at the RAM
#define RAM_WRITE_FORMAT(a)  a |= 0x80      // Address
#define RAM_READ_FORMAT(a)   a |= 0x80      // Address

// Format for the strobes
#define STROBE_FORMAT(a)     REG_WRITE_FORMAT(a)

// Diver definitions
#define REG_BANK_WRITE       0x80           // Bank to write access at register of RAM
#define REG_BANK_READ        0xA0           // Bank to read access at register of RAM


/******** CALLBACK FUNCTIONS *****************************************************/
void (*fifop_callback)(void) = 0;
void (*fifo_callback)(void) = 0;


/******** FUNCTIONS **************************************************************/

/***********************************************************************************/
/* Initialition of the CC2420.                                                     */
/*                                                                                 */
/* @DATE      : 09.11.2007                                                         */
/* @AUTHOR    : Ravel                                                              */
/* @INPUT     : -                                                                  */
/* @OUPUT     : -                                                                  */
/* @REVISION  : v1.0                                                               */
/***********************************************************************************/
void cc2420_init(void)
{
  //---Variables
  //uint8_t status;

  //---Pin configuration
  P2DIR |= CC2420_VREG + CC2420_RESET;      // Set VREG and RESET as output
  // P2DIR &= ~CC2420_SFD;                  // Set SFD (Start of Frame Delimiter) as input
  P2OUT &= ~(CC2420_VREG + CC2420_RESET);   // Set VREG and RESET at Low level
                                            //  (VREG = OFF and RESET = ACTIV)

  /*
  P1DIR &= ~(CC2420_FIFO + CC2420_FIFOP + CC2420_CCA);  // Set FIFO, FIFOP, CCA as inputs
  P1IES &= ~CC2420_FIFOP;                   // Interruption of FIFOP at rising edge
  P1IES |= CC2420_FIFO;                     // Interruption of FIFO at falling edge
  P1IFG &= ~(CC2420_FIFO + CC2420_FIFOP);   // Reset interruption flag of FIFOP and FIFO
  */

  CC2420_VREG_ON;                            // Set VREG at HIGH level

  spi_init();
  //DELAY(100);                             // Waiting for enable the regulator

```

```
  P2OUT &= ~CC2420_RESET;                    // Reset the 2420 -> Set Low level
  //DELAY(100);                              // Waiting

  P2OUT |= CC2420_RESET;                     // End of reset -> Reset High level

  cc2420_strobe(CC2420_SXOSCON);             // Turn on the crystal oscillator

// do
// {
//   status =  cc2420_get_status();          // Read status-bits of the CC2420
// }
// while(!(status & 0x40));                  // Wait as long as the oscillator is stabilized

// P1IE |= (CC2420_FIFO + CC2420_FIFOP);     // Release the interrupts FIFO and FIFOP
}

/*********************************************************************************/
/* Writes in a register.                                                         */
/*                                                                               */
/* @DATE      : 09.11.2007                                                       */
/* @AUTHOR    : Ravel                                                            */
/* @INPUT     : uint8_t  adr  > Address of register                              */
/*              uint16_t data > Data                                             */
/* @OUPUT     : -                                                                */
/* @REVISION  : v1.0                                                             */
/*********************************************************************************/
void cc2420_set_reg(uint8_t adr, uint16_t data)
{
  //---Variables
  uint8_t data_tmp = 0;

  //---Correct address convertion to write a register
  REG_WRITE_FORMAT(adr);

  //---Writing in register over SPI
  CC2420_SELECT;                             // Select CC2420 (SPI_CS)
  spi_send(adr);                             // Send writing address over SPI to CC2420
  data_tmp = (uint8_t) (data >> 8);          // Select the higher byte of data value
  spi_send(data_tmp);                        // Send the higher byte over SPI to CC2420
  data_tmp = (uint8_t) data;                 // Select the lower byte of data value
  spi_send(data_tmp);                        // Send the higher byte over SPI to CC2420
  CC2420_DESELECT;                           // De-select CC2420 (SPI_CS)
}

/*********************************************************************************/
/* Reads from a register.                                                        */
/*                                                                               */
/* @DATE      : 09.11.2007                                                       */
/* @AUTHOR    : Ravel                                                            */
/* @INPUT     : uint8_t  adr  > Address of register                              */
/* @OUPUT     : uint16_t      < Data                                             */
/* @REVISION  : v1.0                                                             */
/*********************************************************************************/
uint16_t cc2420_get_reg(uint8_t adr)
{
  //---Variables
  uint16_t data = 0;

  //---Correct address convertion to read a register
  REG_READ_FORMAT(adr);

  //---Reading from register over SPI
  CC2420_SELECT;                             // Select CC2420 (SPI_CS)
  spi_send(adr);                             // Send reading address over SPI to CC2420
  data = spi_receive();                      // Receiving the higher byte over SPI from CC2420
  data = data << 8;                          // Shift the received bloc to the higher range
  data = spi_receive();                      // Receiving the lower byte over SPI from CC2420
  CC2420_DESELECT;                           // De-select CC2420 (SPI_CS)
  return data;
}

/*********************************************************************************/
/* Read status of CC2420.                                                        */
/*                                                                               */
/* @DATE      : 09.11.2007                                                       */
/* @AUTHOR    : Ravel                                                            */
```

```
153 /* @INPUT      : -                                                                  */
154 /* @OUPUT      : uint8_t < Status                                                   */
155 /* @REVISION   : v1.0                                                               */
156 /*********************************************************************************/
157 uint8_t cc2420_get_status(void)
158 {
159   //--- Variables
160   uint8_t data;
161
162   //--- Send CC2420_SNOP to receive the status bits <see chipcon CC2420 manual p.29 table.5>
163   CC2420_SELECT;                          // Select CC2420 (SPI_CS)
164   spi_send(CC2420_SNOP);                  // Send CC2420_SNOP = No operation
165   data = U1RXBUF;                         // Receive status-bits
166   CC2420_DESELECT;                        // De-select CC2420 (SPI_CS)
167
168   return data;
169 }
170
171 /*********************************************************************************/
172 /* Write in CC2420 TX_FIFO to send a RF-Frame.                                      */
173 /*                                                                                  */
174 /* @DATE       : 09.11.2007                                                         */
175 /* @AUTHOR     : Ravel                                                              */
176 /* @INPUT      : uint8_t len   > Length of the frame to send                        */
177 /*               uint8_t *data > pointer with frame (data) to send                  */
178 /* @OUPUT      : -                                                                  */
179 /* @REVISION   : v1.0                                                               */
180 /*********************************************************************************/
181 void cc2420_write_fifo( uint8_t len, uint8_t *data)
182 {
183   //---Variables
184   uint8_t i,adr;
185
186   //---Assignment of the TX FIFO address and correct address convertion to write to it
187   adr = CC2420_TXFIFO;
188   REG_WRITE_FORMAT(adr);
189
190   //---Writing in TX FIFO buffer over SPI
191   CC2420_SELECT;                          // Select CC2420 (SPI_CS)
192   spi_send(adr);                          // Send writing address over SPI to CC2420
193   for(i = 0; i < len; i++)
194     spi_send(data[i]);                    // Send the frame over SPI to CC2420 TX buffer
195   CC2420_DESELECT;                        // De-select CC2420 (SPI_CS)
196 }
197
198 /*********************************************************************************/
199 /* Reads from CC2420 RX_FIFO -> when a RF-Frame is received.                         */
200 /*                                                                                  */
201 /* @DATE       : 09.11.2007                                                         */
202 /* @AUTHOR     : Ravel                                                              */
203 /* @INPUT      : uint8_t len   > Length of the data to read                         */
204 /*               uint8_t *data > pointer to save the readed data                    */
205 /* @OUPUT      : -                                                                  */
206 /* @REVISION   : v1.0                                                               */
207 /*********************************************************************************/
208 void cc2420_read_fifo(uint8_t len, uint8_t *data)
209 {
210   //---Variables
211   uint8_t i,adr;
212
213   //---Assignment of the RX FIFO address and correct address convertion to read from it
214   adr = CC2420_RXFIFO;
215   REG_READ_FORMAT(adr);
216
217   //---Reading in RX FIFO buffer over SPI
218   CC2420_SELECT;                          // Select CC2420 (SPI_CS)
219   spi_send(adr);                          // Send reading address over SPI to CC2420
220   for(i=0;i<len;i++)
221   {
222     data[i] = spi_receive();              // Receive the frame over SPI from the RX buffer
223   }
224   CC2420_DESELECT;                        // De-select CC2420 (SPI_CS)
225 }
226
227 /*********************************************************************************/
228 /* Calls a strobe register.                                                         */
```

```c
/*                                                                                    */
/* @DATE      : 09.11.2007                                                            */
/* @AUTHOR    : Ravel                                                                 */
/* @INPUT     : uint8_t cmd > Command of strobe                                       */
/* @OUPUT     : -                                                                     */
/* @REVISION  : v1.0                                                                  */
/*************************************************************************************/
void cc2420_strobe(uint8_t cmd)
{
  //---Correct address convertion to write a strobe register
  STROBE_FORMAT(cmd);

  //--- Write to strobe register over SPI
  CC2420_SELECT;                            // Select CC2420 (SPI_CS)
  spi_send(cmd);                            // Send reading address over SPI to CC2420
  CC2420_DESELECT;                          // De-select CC2420 (SPI_CS)
}

/*************************************************************************************/
/* Write to a RAM register.                                                           */
/*                                                                                    */
/* @DATE      : 09.11.2007                                                            */
/* @AUTHOR    : Ravel                                                                 */
/* @INPUT     : uint8_t  adr   > Address of RAM register.                             */
/*              uint8_t  len   > Length of the data to write                          */
/*              uint16_t *data > Pointer with data buffer                             */
/* @OUPUT     : -                                                                     */
/* @REVISION  : v1.0                                                                  */
/*************************************************************************************/
void cc2420_set_reg_ram(uint8_t adr, uint8_t len, uint16_t *data)
{
  //---Variables
  uint16_t i;
  uint8_t data_tmp;

  //---Correct address convertion to write to RAM
  RAM_WRITE_FORMAT(adr);

  //---Write to RAM over SPI
  CC2420_SELECT;                            // Select CC2420 (SPI_CS)
  spi_send(adr);                            // Send writing address over SPI to CC2420
  spi_send(REG_BANK_WRITE);                 // Send the writing bank address
  for(i=0;i<len;i++)
  {
    spi_send((uint8_t)data[i]);             // Send the lower byte over SPI to CC2420
    data_tmp = (uint8_t) (data[i] >> 8);    // Shifting the higher byte to lower byte
    spi_send(data_tmp);                     // Send the higher byte over SPI to CC2420
  }
  CC2420_DESELECT;                          // De-select CC2420 (SPI_CS)
}

/*************************************************************************************/
/* Reads from a RAM register.                                                         */
/*                                                                                    */
/* @DATE      : 09.11.2007                                                            */
/* @AUTHOR    : Ravel                                                                 */
/* @INPUT     : uint8_t  adr   > Address of RAM register.                             */
/*              uint8_t  len   > Length of the data to write                          */
/*              uint16_t *data > Pointer with data buffer to save the reading values  */
/* @OUPUT     : -                                                                     */
/* @REVISION  : v1.0                                                                  */
/*************************************************************************************/
void cc2420_get_reg_ram(uint8_t adr, uint8_t len, uint16_t *data)
{
  //---Variables
  uint8_t i;
  uint16_t data_tmp;

  //---Correct address convertion to reads from RAM
  RAM_READ_FORMAT(adr);

  //---Reads from RAM over SPI
  CC2420_SELECT;                            // Select CC2420 (SPI_CS)
  spi_send(adr);                            // Send reading address over SPI to CC2420
  spi_send(REG_BANK_READ);                  // Send the reading bank address
  for(i=0;i<len;i++)
```

```
305   {
306     data[i] = spi_receive();                // Receive the lower byte over SPI to CC2420
307     data_tmp = spi_receive();               // Receive the higher byte over SPI to CC2420
308     data[i] |= (data_tmp << 8);             // Shifting the received higher byte
309   }
310   CC2420_DESELECT;                          // De-select CC2420 (SPI_CS)
311 }
312
313 /*****************************************************************************************/
314 /* [INTERRUPT] - PORT1 interrupt                                                         */
315 /*                                                                                       */
316 /* @DATE      : 09.11.2007                                                               */
317 /* @AUTHOR    : Ravel                                                                    */
318 /* @INPUT     : -                                                                        */
319 /* @OUPUT     : -                                                                        */
320 /* @REVISION  : v1.0                                                                     */
321 /*****************************************************************************************/
322 interrupt(PORT1_VECTOR) port1_isr(void)
323 {
324   //---Call rf2 callback receive function
325   if (P1IFG & CC2420_FIFOP)
326   {
327     P1IFG &= ~CC2420_FIFOP;                 // Reset interrupt flag FIFOP
328     (*fifop_callback)();                    // Call rf2 callback function
329   }
330
331   //---Call rf2 callback overflow function
332   if (P1IFG & CC2420_FIFO)
333   {
334     P1IFG = 0;                              // Reset interrupt flag to 0
335     (*fifo_callback)();                     // Call rf2 callback function
336   }
337 }
338
```

```c
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Classfile cc2420.c                                                               */
/*                                                                                   */
/*  Class to work with the CC2420 Chipcon Transceiver                                */
/*-----------------------------------------------------------------------------------*/

/******** INCLUDES ***************************************************************/
#include "cc2420.h"
#include "definitions.h"
#include "spi.h"
#include <io.h>
#include <signal.h>
#include <stdint.h>


/******** INTERN CONSTANTS *******************************************************/

// Format for the access at registers
#define REG_WRITE_FORMAT(a)  a &= 0x3F      // Address
#define REG_READ_FORMAT(a)   a |= 0x40      // Address

// Format for the access at the RAM
#define RAM_WRITE_FORMAT(a)  a |= 0x80      // Address
#define RAM_READ_FORMAT(a)   a |= 0x80      // Address

// Format for the strobes
#define STROBE_FORMAT(a)     REG_WRITE_FORMAT(a) // Mise en forme de la commande

// Diver definitions
#define REG_BANK_WRITE       0x80           // Bank to write access at register of RAM
#define REG_BANK_READ        0xA0           // Bank to read access at register of RAM


/******** CALLBACK FUNCTIONS *****************************************************/
void (*fifop_callback)(void) = 0;
void (*fifo_callback)(void) = 0;

/******** STATIC INTERN CLASS VARIABLES ******************************************/
static uint8_t cc2420_buffer_rx[1024];
static int p_read = 0;
static int p_write = 0;
static volatile int lenght = 0;

/******** FUNCTIONS **************************************************************

/***********************************************************************************/
/* Initialition of the CC2420.                                                     */
/*                                                                                 */
/* @DATE      : 09.11.2007                                                         */
/* @AUTHOR    : Ravel                                                              */
/* @INPUT     : -                                                                  */
/* @OUPUT     : -                                                                  */
/* @REVISION  : v1.0                                                               */
/***********************************************************************************/
void cc2420_init(void)
{
  //---Variables
  uint8_t status;

  //---SPI initialisation
  spi_init();

  //---Pin configuration
  P2DIR |= CC2420_VREG + CC2420_RESET;     // Set VREG and RESET as output
  P2DIR &= ~CC2420_SFD;                    // Set SFD (Start of Frame Delimiter) as input
  P2OUT &= ~(CC2420_VREG + CC2420_RESET);  // Set VREG and RESET at Low level
                                        //  (VREG = OFF and RESET = ACTIV)

  P1DIR &= ~(CC2420_FIFO + CC2420_FIFOP + CC2420_CCA);  // Set FIFO, FIFOP, CCA as inputs
  P1IES &= ~CC2420_FIFOP;                  // Interruption of FIFOP at rising edge
  P1IES |= CC2420_FIFO;                    // Interruption of FIFO at falling edge
  P1IFG &= ~(CC2420_FIFO + CC2420_FIFOP);  // Reset interruption flag of FIFOP and FIFO

  CC2420_VREG_ON;                          // Set VREG at HIGH level
```

```c
  DELAY(0);                                 // Waiting for enable the regulator

  P2OUT &= ~CC2420_RESET;                   // Reset the 2420 -> Set Low level
  DELAY(0);                                 // Waiting

  P2OUT |= CC2420_RESET;                    // End of reset -> Reset High level

  cc2420_strobe(CC2420_SXOSCON);            // Turn on the crystal oscillator

  do
  {
    status =  cc2420_get_status();           // Read status-bits of the CC2420
  }
  while(!(status & 0x40));                   // Wait as long as the oscillator is stabilized

  P1IE |= (CC2420_FIFO + CC2420_FIFOP);      // Release the interrupts FIFO and FIFOP
}

/*****************************************************************************************/
/* Writes in a register.                                                                 */
/*                                                                                       */
/* @DATE      : 09.11.2007                                                               */
/* @AUTHOR    : Ravel                                                                    */
/* @INPUT     : uint8_t  adr  > Address of register                                      */
/*              uint16_t data > Data                                                     */
/* @OUPUT     : -                                                                        */
/* @REVISION  : v1.0                                                                     */
/*****************************************************************************************/
void cc2420_set_reg(uint8_t adr, uint16_t data)
{
  //---Variables
  uint8_t data_tmp = 0;

  //---Correct address convertion to write a register
  REG_WRITE_FORMAT(adr);

  //---Writing in register over SPI
  CC2420_SELECT;                            // Select CC2420 (SPI_CS)
  spi_send(adr);                            // Send writing address over SPI to CC2420
  data_tmp = (uint8_t) (data >> 8);         // Select the higher byte of data value
  spi_send(data_tmp);                       // Send the higher byte over SPI to CC2420
  data_tmp = (uint8_t) data;                // Select the lower byte of data value
  spi_send(data_tmp);                       // Send the higher byte over SPI to CC2420
  CC2420_DESELECT;                          // De-select CC2420 (SPI_CS)
}

/*****************************************************************************************/
/* Reads from a register.                                                                */
/*                                                                                       */
/* @DATE      : 09.11.2007                                                               */
/* @AUTHOR    : Ravel                                                                    */
/* @INPUT     : uint8_t  adr  > Address of register                                      */
/* @OUPUT     : uint16_t      < Data                                                     */
/* @REVISION  : v1.0                                                                     */
/*****************************************************************************************/
uint16_t cc2420_get_reg(uint8_t adr)
{
  //---Variables
  uint16_t data = 0;

  //---Correct address convertion to read a register
  REG_READ_FORMAT(adr);

  //---Reading from register over SPI
  CC2420_SELECT;                            // Select CC2420 (SPI_CS)
  spi_send(adr);                            // Send reading address over SPI to CC2420
  data = spi_receive();                     // Receiving the higher byte over SPI from CC2420
  data = data << 8;                         // Shift the received bloc to the higher range
  data = spi_receive();                     // Receiving the lower byte over SPI from CC2420
  CC2420_DESELECT;                          // De-select CC2420 (SPI_CS)
  return data;
}

/*****************************************************************************************/
/* Read status of CC2420.                                                                */
/*                                                                                       */
```

```c
/* @DATE      : 09.11.2007                                                          */
/* @AUTHOR    : Ravel                                                               */
/* @INPUT     : -                                                                   */
/* @OUPUT     : uint8_t < Status                                                    */
/* @REVISION  : v1.0                                                                */
/**********************************************************************************/
uint8_t cc2420_get_status(void)
{
  //--- Variables
  uint8_t data;

  //--- Send CC2420_SNOP to receive the status bits <see chipcon CC2420 manual p.29 table.5>
  CC2420_SELECT;                          // Select CC2420 (SPI_CS)
  spi_send(CC2420_SNOP);                  // Send CC2420_SNOP = No operation
  data = U1RXBUF;                         // Receive status-bits
  CC2420_DESELECT;                        // De-select CC2420 (SPI_CS)

  return data;
}

/**********************************************************************************/
/* Write in CC2420 TX_FIFO to send a RF-Frame.                                      */
/*                                                                                  */
/* @DATE      : 09.11.2007                                                          */
/* @AUTHOR    : Ravel                                                               */
/* @INPUT     : uint8_t len   > Length of the frame to send                         */
/*              uint8_t *data > pointer with frame (data) to send                   */
/* @OUPUT     : -                                                                   */
/* @REVISION  : v1.0                                                                */
/**********************************************************************************/
void cc2420_write_fifo( uint8_t len, uint8_t *data)
{
  //---Variables
  uint8_t i,adr;

  //---Assignment of the TX FIFO address and correct address convertion to write to it
  adr = CC2420_TXFIFO;
  REG_WRITE_FORMAT(adr);

  //---Writing in TX FIFO buffer over SPI
  CC2420_SELECT;                          // Select CC2420 (SPI_CS)
  spi_send(adr);                          // Send writing address over SPI to CC2420
  for(i = 0; i < len; i++)
    spi_send(data[i]);                    // Send the frame over SPI to CC2420 TX buffer
  CC2420_DESELECT;                        // De-select CC2420 (SPI_CS)
}

/**********************************************************************************/
/* Reads from CC2420 RX_FIFO -> when a RF-Frame is received.                        */
/*                                                                                  */
/* @DATE      : 09.11.2007                                                          */
/* @AUTHOR    : Ravel                                                               */
/* @INPUT     : uint8_t len   > Length of the data to read                          */
/*              uint8_t *data > pointer to save the readed data                     */
/* @OUPUT     : -                                                                   */
/* @REVISION  : v1.0                                                                */
/**********************************************************************************/
void cc2420_read_fifo(uint8_t len, uint8_t *data)
{
  //---Variables
  uint8_t i,adr;

  //---Assignment of the RX FIFO address and correct address convertion to read from it
  adr = CC2420_RXFIFO;
  REG_READ_FORMAT(adr);

  //---Reading in RX FIFO buffer over SPI
  CC2420_SELECT;                          // Select CC2420 (SPI_CS)
  spi_send(adr);                          // Send reading address over SPI to CC2420
  for(i=0;i<len;i++)
  {
    data[i] = spi_receive();              // Receive the frame over SPI from the RX buffer
  }
  CC2420_DESELECT;                        // De-select CC2420 (SPI_CS)
}
```

```c
229 /*****************************************************************************************/
230 /* Calls a strobe register.                                                              */
231 /*                                                                                       */
232 /* @DATE      : 09.11.2007                                                               */
233 /* @AUTHOR    : Ravel                                                                    */
234 /* @INPUT     : uint8_t cmd > Command of strobe                                          */
235 /* @OUPUT     : -                                                                        */
236 /* @REVISION  : v1.0                                                                     */
237 /*****************************************************************************************/
238 void cc2420_strobe(uint8_t cmd)
239 {
240   //---Correct address convertion to write a strobe register
241   STROBE_FORMAT(cmd);
242
243   //--- Write to strobe register over SPI
244   CC2420_SELECT;                            // Select CC2420 (SPI_CS)
245   spi_send(cmd);                            // Send reading address over SPI to CC2420
246   CC2420_DESELECT;                          // De-select CC2420 (SPI_CS)
247 }
248
249 /*****************************************************************************************/
250 /* Write to a RAM register.                                                              */
251 /*                                                                                       */
252 /* @DATE      : 09.11.2007                                                               */
253 /* @AUTHOR    : Ravel                                                                    */
254 /* @INPUT     : uint8_t  adr   > Address of RAM register.                                */
255 /*              uint8_t  len   > Length of the data to write                             */
256 /*              uint16_t *data > Pointer with data buffer                                */
257 /* @OUPUT     : -                                                                        */
258 /* @REVISION  : v1.0                                                                     */
259 /*****************************************************************************************/
260 void cc2420_set_reg_ram(uint8_t adr, uint8_t len, uint16_t *data)
261 {
262   //---Variables
263   uint16_t i;
264   uint8_t data_tmp;
265
266   //---Correct address convertion to write to RAM
267   RAM_WRITE_FORMAT(adr);
268
269   //---Write to RAM over SPI
270   CC2420_SELECT;                            // Select CC2420 (SPI_CS)
271   spi_send(adr);                            // Send writing address over SPI to CC2420
272   spi_send(REG_BANK_WRITE);                 // Send the writing bank address
273   for(i=0;i<len;i++)
274   {
275     spi_send((uint8_t)data[i]);             // Send the lower byte over SPI to CC2420
276     data_tmp = (uint8_t) (data[i] >> 8);    // Shifting the higher byte to lower byte
277     spi_send(data_tmp);                     // Send the higher byte over SPI to CC2420
278   }
279   CC2420_DESELECT;                          // De-select CC2420 (SPI_CS)
280 }
281
282 /*****************************************************************************************/
283 /* Reads from a RAM register.                                                            */
284 /*                                                                                       */
285 /* @DATE      : 09.11.2007                                                               */
286 /* @AUTHOR    : Ravel                                                                    */
287 /* @INPUT     : uint8_t  adr   > Address of RAM register.                                */
288 /*              uint8_t  len   > Length of the data to write                             */
289 /*              uint16_t *data > Pointer with data buffer to save the reading values     */
290 /* @OUPUT     : -                                                                        */
291 /* @REVISION  : v1.0                                                                     */
292 /*****************************************************************************************/
293 void cc2420_get_reg_ram(uint8_t adr, uint8_t len, uint16_t *data)
294 {
295   //---Variables
296   uint8_t i;
297   uint16_t data_tmp;
298
299   //---Correct address convertion to reads from RAM
300   RAM_READ_FORMAT(adr);
301
302   //---Reads from RAM over SPI
303   CC2420_SELECT;                            // Select CC2420 (SPI_CS)
304   spi_send(adr);                            // Envoi de l'adresse
```

```
305   spi_send(REG_BANK_READ);                  // Envoi du bank et de l'ordre de lecture
306   for(i=0;i<len;i++)
307   {
308     data[i] = spi_receive();                // Receive the lower byte over SPI to CC2420
309     data_tmp = spi_receive();               // Receive the higher byte over SPI to CC2420
310     data[i] |= (data_tmp << 8);             // Shifting the received higher byte
311   }
312   CC2420_DESELECT;                          // De-select CC2420 (SPI_CS)
313 }
314
315 /*****************************************************************************************/
316 /* Discharging of the RX buffer = discharging the received frames                        */
317 /*                                                                                       */
318 /* @DATE      : 20.11.2007                                                               */
319 /* @AUTHOR    : Ravel                                                                    */
320 /* @INPUT     : uint8_t *data > Data pointer.                                            */
321 /* @OUPUT     : uint8_t       < buffer state : If buffer empty or not                    */
322 /* @REVISION  : v1.0                                                                     */
323 /*****************************************************************************************/
324 uint8_t cc2420_receive(uint8_t * data)
325 {
326   P1IE &= ~( CC2420_FIFOP);
327   if(lenght!=0)
328   {
329     *data = cc2420_buffer_rx[p_read];
330     p_read = (p_read+1)%1024;
331     lenght--;
332     P1IE |= ( CC2420_FIFOP);
333     return 1;
334   }
335   P1IE |= ( CC2420_FIFOP);
336   return 0;
337
338 }
339
340 /*****************************************************************************************/
341 /* [INTERRUPT] - PORT1 interrupt                                                         */
342 /*                                                                                       */
343 /* @DATE      : 09.11.2007                                                               */
344 /* @AUTHOR    : Ravel                                                                    */
345 /* @INPUT     : -                                                                        */
346 /* @OUPUT     : -                                                                        */
347 /* @REVISION  : v1.0                                                                     */
348 /*****************************************************************************************/
349 interrupt(PORT1_VECTOR) port1_isr(void)
350 {
351   uint8_t adr;
352   uint8_t length_frame;
353   uint8_t i;
354
355   //---Assignment of the RX FIFO address and correct address convertion to read from it
356   adr = CC2420_RXFIFO;
357   REG_READ_FORMAT(adr);
358
359   led_1_on();
360
361   if (P1IFG & CC2420_FIFO)
362   {
363     if(P1IN & CC2420_FIFOP)
364     {
365       cc2420_strobe(CC2420_SFLUSHRX);
366       P1IFG &= ~(CC2420_FIFOP);
367     }
368     P1IFG &= ~(CC2420_FIFO);
369   }
370
371   if (P1IFG & CC2420_FIFOP)
372   {
373     cc2420_read_fifo(1, &length_frame);
374     if (lenght+length_frame+1<1024)
375     {
376       cc2420_buffer_rx [p_write] = length_frame;
377       p_write = (p_write+1)&1023;
378       lenght++;
379
380 //      for(i=0;i<length_frame;i++)
```

```
381 //      {
382 //        cc2420_read_fifo(1, cc2420_buffer_rx + p_write);
383 //        p_write = (p_write+1)%1024;
384 //        lenght++;
385 //      }
386       CC2420_SELECT;                          // Select CC2420 (SPI_CS)
387       spi_send(adr);                          // Send reading address over SPI to CC2420
388       for(i=0;i<length_frame;i++)
389       {
390         //---Reading in RX FIFO buffer over SPI
391         cc2420_buffer_rx[p_write] = spi_receive();
392         p_write = (p_write+1)&1023;
393         lenght++;
394       }
395       CC2420_DESELECT;                        // De-select CC2420 (SPI_CS)
396     }
397     else
398     {
399       cc2420_strobe(CC2420_SFLUSHRX);
400     }
401     P1IFG &= ~CC2420_FIFOP; //remise à zéro du flag concernant l'IT issue de FIFOP
402   }
403
404   led_1_off();
405 }
406
```

```c
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Headerfile czame.h                                                               */
/*                                                                                   */
/*  Car detection algorith.                                                          */
/*-----------------------------------------------------------------------------------*/

/*
  Algorithme stochastique de détection de véhicules par l'examen de
  la déviation du champ magnétique terrestre.

  Cet algorithme est prévu pour être le plus portable possible.
  Il peut donc être compilé avec les options: -ansi -Wall -pedantic.
  Il peut fonctionner sur toutes cibles aussi bien en embarqué que sur pc.
  Testé avec succès sur pc avec mingw, et sur msp430f169 avec iar et mspgcc.

  Pour une performance maximum on peut utiliser les options suivantes
  sur gcc ou mspgcc: -finline-functions -funroll-loops.
*/

#ifndef _CZAME_H_
#define _CZAME_H_

#include <stdint.h>

#define REPORT_FREE 0
#define REPORT_BUSY 1

/** Exemple de paramètres pour l'algorithme **/

#define CZ_BPS 12

#define CZ_CONF_XYZ "12/A28?A31C116/0C8540811C900A80/6"
#define CZ_CONF_XY  "12/A23?A27C109/0DB315601D701350/6"

/** Pointeur sur la fonction de diagnostique (callback).
    Par default c'est un pointeur sur NULL, et dans ce cas
    le diagnostique de l'algorithme est tous simplement ignoré. **/

extern void (*czame_report)(int16_t);

/** Initialise les paramètres de l'algorithme (ex: CZ_CONF_XYZ). **/

void czame_config (const char *str);

/** Initialisation des variables de l'algorithme.
    Pré-supose que le système demmare en l'abscence de véhicule. **/

void czame_init (void);

/** Soummet une mesure à l'algorithme.
    Cette fonction appelle "czame_report" toutes les "bps" mesures,
    si "czame_report" est différent du pointeur NULL.
    Cette fonction suppose que l'on utilise un ADC10, c'est à
    dire des mesures comprises entre 0 et 1023, avec 512 pour neutre. **/

void czame_play (uint16_t x, uint16_t y, uint16_t z);

#endif
```

```c
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Classfile czame.c                                                                */
/*                                                                                   */
/*  Car detection algorithm.                                                         */
/*-----------------------------------------------------------------------------------*/

/*
  Algorithme stochastique de détection de véhicules par l'examen de
  la déviation du champ magnétique terrestre.

  Cet algorithme est prévu pour être le plus portable possible.
  Il peut donc être compilé avec les options: -ansi -Wall -pedantic.
  Il peut fonctionner sur toutes cibles aussi bien en embarqué que sur pc.
  Testé avec succès sur pc avec mingw, et sur msp430f169 avec iar et mspgcc.

  Pour une performance maximum on peut utiliser les options suivantes
  sur gcc ou mspgcc: -finline-functions -funroll-loops.
*/

#include "czame.h"

#include <stdint.h>

/*
#include <stdio.h>
#define debug(...) fprintf(stderr, __VA_ARGS__)
*/

/* Nombre d'états de la chaîne de Markov (proche d'un anti-rebond) */
#define STATE 4

/* Pénalitée maximum pouvant être portée par un état */
#define METRIC 48

/* Nombre maximum de symboles de l'alphabet d'entrée (étage viterbi) */
#define MOSAIC 16

/* Nombre d'étapes en phase arrière (secondes) */
/*#define BACKWARD 6*/

/* Longueur de la matrices des pénalités */
/* La puissance de 2 immédiatement supérieure à BACKWARD+1 */
#define LENGTH 8

/* Numéro de l'observation qui fait tendre vers l'état stable (pas de véhicules) */
#define NEUTRAL 0

/** 1. Partie privée de l'algorithme (inlinées par -finline-functions) **/

/** 1.1. clear        : Initialise un vecteur d'état **/
/** 1.2. copy         : Copie un vecteur d'état **/
/** 1.3. forward      : Avance dans le treillis des possibles **/
/** 1.4. backward     : Recule dans le treillis des possibles **/
/** 1.5. viterbi      : Reconstitue l'explication la plus probable **/
/** 1.6. isqrt        : Calcul de la racine carrée **/
/** 1.7. length       : Calcul la norme d'un vecteur **/

/** 2. Partie publique de l'algorithme (interface) **/

/** 2.1. czame_config : Initialise les paramètres de l'algorithme **/
/** 2.2. czame_init   : Initialise les variables de l'algorithme **/
/** 2.3. czame_play   : Soumet une mesure à l'algorithme **/

/* Indicateurs pour l'introspection (optimisation) */
#ifdef SURVEY
uint16_t survey_gap;
uint16_t survey_disp;
uint8_t  survey_obsrv;
#endif

/** Outils divers **/

#define MIN(A,B) ((A)<(B)?(A):(B))
#define MAX(A,B) ((A)>(B)?(A):(B))
```

```c
77  #define ABS(X)   ((X)<0?-(X):(X))
78
79  /** Pointeur publique sur la fonction de signalisation (callback) **/
80  /** Elle indique le résultat de l'algorithme, c'est à dire la présence
81      ou l'abscence de véhicule au dessus du capteur, tous les BPS applels
82      à la fonction "czame_play". **/
83
84  void (*czame_report)(int16_t) = 0;
85
86  /** Paramètres de l'algorithme **/
87  /** Peuvent être, eventuellement, déterminés à la compilation et donc
88      passés dans un segment constant (ie: en flash).  Dans ce cas, la
89      fonction "czame_config" n'a plus de raison d'être. **/
90
91  /** NOTE: Si on fixe bps=12 et shift=72 (delay=6), alors :
92      Avec czame_config("12/A28?A31C116/0C8540811C900A80");
93      $ cat corpus.log | ./survey -m A28?A31C116 -s 0C8540811C900A80 -eval
94      on obtient max=611 moy=576.33
95      Avec czame_config("12/A23?A27C109/0DB315601D701350");
96      $ cat corpus.log | ./survey -noz -m A23?A27C109 -s 0DB315601D701350 -eval
97      on obtient max=2575 moy=2514.67 (on manque un véhicule) **/
98
99  /* EVAL [11/A27?A27C116/0C8530701C900A60] => MAX=694 MOY=600.27 */
100 /* EVAL [12/A28?A31C116/0C8540811C900A80] => MAX=611 MOY=576.33 */
101 /* EVAL [13/A31?A36C104/0353205212602051] => MAX=824 MOY=812.30 */
102 /* EVAL [14/A31?A37C117/0551304114605074] => MAX=677 MOY=659.71 */
103
104 /* EVAL [12/A28?A31C116/0C8540811C900A80] => MAX=611 MOY=576.33 */
105 /* EVAL [12/A28?A27C116B45?/0C8530811C900A803050] => MAX=587 MOY=560.50 */
106
107 /** On fixe toujours BPS=12 **/
108
109 /* EVAL [12/A28?A30C116/0B7430601C900A80/5] => MAX=611 MOY=574.17 */
110 /* EVAL [10/A30?A34C114/0F4412001F501620/7] => MAX=619 MOY=606.60 */
111
112 static struct {
113
114   uint8_t bps;     /* Nombre de mesures à agréger (chunk size) */
115   uint16_t magic;  /* Pour une division rapide par "bps" */
116
117   uint8_t card;  /* Nombre de symboles de l'alphabet d'entrée (étage viterbi) */
118
119   /* Description de la transformation d'espace (calcul de l'observation) */
120   struct { unsigned value:8, from:7, kind:1; } mosaic [MOSAIC];
121
122   uint8_t initial [STATE];           /* Vecteur des états initiaux */
123   uint8_t metrics [MOSAIC] [STATE];  /* Matrice des pénalités */
124
125   uint8_t delay;  /* Nombre d'étapes en phase arrière (secondes) */
126
127 } etc;
128
129 /** Etat du système **/
130 /** Represente l'état mémoire de l'algorithme, se trouvant
131     obligatoirement en mémoire vive (RAM). **/
132
133 static struct {
134
135   struct {  int16_t x, y, z; } prev;
136   struct {  int32_t x, y, z; } sum;
137   struct { uint32_t x, y, z; } delta_sum;
138   uint8_t count;
139
140   uint8_t matrix [LENGTH] [STATE];  /* Treilli (buffer circulaire) */
141   uint8_t pos;                      /* Position dans le treilli */
142
143 } var;
144
145 /** 1.1. Initialise un vecteur d'état **/
146
147 static void clear (uint8_t *tgt) {
148
149   uint8_t i;
150   for (i = 0; i < STATE; i++)
151     tgt[i] = 0;
152 }
```

```c

/** 1.2. Copie un vecteur d'état **/

static void copy (uint8_t *tgt, const uint8_t *src) {

  uint8_t i;
  for (i = 0; i < STATE; i++)
    tgt[i] = src[i];
}

/** Un changement du nombre d'états et/ou du treilli élémentaire,
    n'implique que la modification de "forward" et de "backward".
    Actuellement le treilli est proche de celui d'un anti-rebond. **/

/** 1.3. Avance dans le treillis des possibles **/

static void forward (uint8_t *s, const uint8_t *e, uint8_t obsrv) {

  uint8_t floor;
  uint8_t t0, t1, t2, t3;

  /* Confronte l'état à l'observation */
  t0 = MIN(e[0],e[2]) + etc.metrics[obsrv][0];
  t1 = e[0]           + etc.metrics[obsrv][1];
  t2 = e[3]           + etc.metrics[obsrv][2];
  t3 = MIN(e[1],e[3]) + etc.metrics[obsrv][3];

  /* Normalise le vecteur des états */
  floor = t0;
  if (t1 < floor) floor = t1;
  if (t2 < floor) floor = t2;
  if (t3 < floor) floor = t3;

  s[0] = MIN(METRIC -1, t0 -floor);
  s[1] = MIN(METRIC -1, t1 -floor);
  s[2] = MIN(METRIC -1, t2 -floor);
  s[3] = MIN(METRIC -1, t3 -floor);
}

/** 1.4. Recule dans le treillis des possibles **/

static uint8_t backward (const uint8_t *e, uint8_t best) {

  switch (best) {
  case 0 : if (e[2] < e[0]) best = 2; break;
  case 1 : best = 0; break;
  case 2 : best = 3; break;
  case 3 : if (e[1] < e[3]) best = 1; break;
  }

  return best;
}

/** 1.5. Reconstitue l'explication la plus probable **/

static void viterbi (uint8_t obsrv) {

  uint8_t curs;
  uint8_t i, j, best;

  /* Appliquer la nouvelle observation */
  curs = (var.pos +LENGTH -1) % LENGTH;
  forward(var.matrix[var.pos], var.matrix[curs], obsrv);

  /* Rechercher l'état de métrique le plus faible (le plus probable) */
  best = 0;
  for (i = 1; i < STATE; i++)
    if (var.matrix[var.pos][i] < var.matrix[var.pos][best])
      best = i;

  /* Remonter la chaine des survivants (chemin de viterbi) */
  for (j = 1; j <= etc.delay; j++) {
    curs = (var.pos +LENGTH -j) % LENGTH;
    best = backward(var.matrix[curs], best);
  }

```

```c
  /* Emettre le symbole le plus probable */
  if (czame_report) switch (best) {
  case 0:
  case 1: (*czame_report)(REPORT_FREE);
  break;
  case 2:
  case 3: (*czame_report)(REPORT_BUSY);
  break;
  }

  /* Passer à l'observation suivante */
  var.pos = (var.pos +1) % LENGTH;
}

/** 1.6. Calcul de la racine carrée **/

static uint16_t isqrt (uint32_t x) {

  uint32_t y, mask;
  y = 0;
  mask = 1UL << 30;
  while (x < mask) mask >>= 2;
  while (mask) {
    if (x >= (y | mask)) {
      x -= y | mask;
      y += mask << 1;
    }
    mask >>= 2;
    y >>= 1;
  }
  return y;
}

/** 1.7. Calcul la norme d'un vecteur **/

static uint16_t length (int16_t x, int16_t y, int16_t z) {

  uint32_t a;
  a  = (int32_t)x * x;
  a += (int32_t)y * y;
  a += (int32_t)z * z;
  return isqrt(a);
}

/** 2.1. Initialise les paramères de l'algorithme **/

void czame_config (const char *str) {

  uint8_t tmp[STATE];
  uint8_t i, j, n = 0;

  /* Initialisations relatives aux BPS */
  etc.bps = 0;
  while ('0' <= str[n] && str[n] <= '9') {
    etc.bps *= 10;
    etc.bps += str[n] - '0';
    n++;
  }
  etc.magic = (1UL << 16) / etc.bps;

  n++;

  /* Inititalisation de la segmentatation (mosaic) */
  for (i = 0; str[n] != '/'; i++) {
    etc.mosaic[i].from = str[n++] - 'A';
    etc.mosaic[i].value = 0;
    while ('0' <= str[n] && str[n] <= '9') {
      etc.mosaic[i].value *= 10;
      etc.mosaic[i].value += str[n] - '0';
      n++;
    }
    etc.mosaic[i].kind = 0;
    if (str[n] == '?') { etc.mosaic[i].kind = 1; n++; }
  }
  etc.card = i+1;

```

```c
  n++;

  /* Initialisation de la matrice des pénalités */
  clear(tmp);
  for (j = 0; j < etc.card; j++) {
    while (str[n] == '|') n++;
    if (str[n] == '+') n++; else
      for (i = 0; i < STATE; i++) {
        tmp[i] = str[n] -('9' < str[n] ? 'A'-10 : '0');
        n++;
      }
    copy(etc.metrics[j], tmp);
  }

  /* Calcul du vecteur état initial */
  clear(etc.initial);
  for (i = 0; i < LENGTH; i++) {
    forward(tmp, etc.initial, NEUTRAL);
    copy(etc.initial, tmp);
  }

  n++;

  /* Initialisation du nombre d'étapes en phase arrière */
  etc.delay = 0;
  while ('0' <= str[n] && str[n] <= '9') {
    etc.delay *= 10;
    etc.delay += str[n] - '0';
    n++;
  }
}

/** 2.2. Initialise les variables de l'algorithme **/

void czame_init (void) {

  uint8_t i, j;

  var.prev.x = var.prev.y = var.prev.z = 512;

  var.sum.x = var.sum.y = var.sum.z = 0;
  var.delta_sum.x = var.delta_sum.y = var.delta_sum.z = 0;
  var.count = etc.bps;

  var.pos = 0;
  for (j = 0; j < LENGTH; j++)
    for (i = 0; i < STATE; i++)
      var.matrix[j][i] = etc.initial[i];
}

/** 2.3. Soumet une mesure à l'algorithme **/

void czame_play (uint16_t x, uint16_t y, uint16_t z) {

  uint8_t n;
  uint8_t obsrv;
  uint16_t gap, disp;
  int16_t dx, dy, dz;

  var.count--;

  /* Agrégation des données */
  dx = x - 512;
  dy = y - 512;
  dz = z - 512;
  /* Pour le calcul de la moyenne (gap) */
  var.sum.x += dx;
  var.sum.y += dy;
  var.sum.z += dz;
  /* Pour le calcul de la dispersion (disp) */
  var.delta_sum.x += ABS(var.prev.x - dx);
  var.delta_sum.y += ABS(var.prev.y - dy);
  var.delta_sum.z += ABS(var.prev.z - dz);
  var.prev.x = dx;
  var.prev.y = dy;
  var.prev.z = dz;
```

```c

  if (0 < var.count) return;  /* Pour chaque agrégat */

  /* Calcul de l'écart moyen */
  gap = length(var.sum.x, var.sum.y, var.sum.z);
  gap = ((int32_t)etc.magic * gap) >> 16;

  /* Calcul de la dispersion */
  disp = var.delta_sum.x;
  if (disp < var.delta_sum.y)  disp = var.delta_sum.y;
  if (disp < var.delta_sum.z)  disp = var.delta_sum.z;
  disp = ((int32_t)etc.magic * disp) >> 14;

  /* Discrétisation des indicateurs (calcul de l'observation) */
  obsrv = 0;
  for (n = 0; n < etc.card-1; n++) {
    if ((obsrv == etc.mosaic[n].from) &&
    ((etc.mosaic[n].kind ? disp : gap) >= etc.mosaic[n].value))
      obsrv = n+1;
  }

  /* Prépare le calcul de l'observation suivante */
  var.sum.x = var.sum.y = var.sum.z = 0;
  var.delta_sum.x = var.delta_sum.y = var.delta_sum.z = 0;
  var.count = etc.bps;

  /* Transmet les indicateurs pour l'introspection */
  #ifdef SURVEY
  survey_gap   = gap;
  survey_disp  = disp;
  survey_obsrv = obsrv;
  #endif

  /* Emission d'une observation */
  viterbi(obsrv);
}

```

```
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Headerfile definitions.h                                                         */
/*                                                                                   */
/*  Global project definintions                                                      */
/*-----------------------------------------------------------------------------------*/

#ifndef _DEFINITIONS_H_
#define _DEFINITIONS_H_

/******** INCLUDES ***************************************************************/
#include <io.h>
#include <stdint.h>


/*************************************************************************************/
/*                                          LEDS                                     */
/*************************************************************************************/

#define led_init()          P2DIR |= BIT5, P2OUT |= BIT5, P2DIR |= BIT6, P2OUT |= BIT6
                                          // Set P2.5 and P2.6 as output and turn off leds

// LED1 <P2.5>
#define led_1_on()          P2OUT &= ~BIT5           // turn LED1 on
#define led_1_off()         P2OUT |=  BIT5           // turn LED1 off
#define led_1_toggle()      P2OUT ^=  BIT5           // toggle LED1 state

// LED2 <P2.6>
#define led_2_on()          P2OUT &= ~BIT6           // turn LED2 on
#define led_2_off()         P2OUT |=  BIT6           // turn LED2 off
#define led_2_toggle()      P2OUT ^=  BIT6           // toggle LED2 state

// Leds (LED1 <P2.5> and LED2 <P2.6>)
#define leds_on()           P2OUT &= ~(BIT5 + BIT6)  // turn LED2 on
#define leds_off()          P2OUT |=  (BIT5 + BIT6)  // turn LED2 off
#define leds_toggle()       P2OUT ^=  (BIT5 + BIT6)  // toggle LED2 state

/*************************************************************************************/
/*                                         CC2420                                    */
/*************************************************************************************/

// CC2420 Configurationn
#define CC2420_FIFO         BIT5            // FIFO  <P1.5>
#define CC2420_FIFOP        BIT6            // FIFOP <P1.6>
#define CC2420_CCA          BIT7            // CCA   <P1.7>
#define CC2420_SFD          BIT0            // SFD   <P2.0>
#define CC2420_RESET        BIT1            // RESET <P2.1>  (ACTIVE at LOW-Level)
#define CC2420_VREG         BIT2            // VREG  <P2.2>  (ACTIVE at HIGH-Level)

#define CC2420_VREG_ON      P2OUT |= CC2420_VREG
#define CC2420_VREG_OFF     P2OUT &= ~CC2420_VREG

// CC2420 Registers
#define CC2420_SNOP         0x00            // Strobe register
#define CC2420_SXOSCON      0x01            // Strobe register
#define CC2420_STXCAL       0x02            // Strobe register
#define CC2420_SRXON        0x03            // Strobe register
#define CC2420_STXON        0x04            // Strobe register
#define CC2420_STXONCCA     0x05            // Strobe register
#define CC2420_SRFOFF       0x06            // Strobe register
#define CC2420_SXOSCOFF     0x07            // Strobe register
#define CC2420_SFLUSHRX     0x08            // Strobe register
#define CC2420_SFLUSHTX     0x09            // Strobe register
#define CC2420_SACK         0x0A            // Strobe register
#define CC2420_SACKPEND     0x0B            // Strobe register
#define CC2420_SRXDEC       0x0C            // Strobe register
#define CC2420_STXENC       0x0D            // Strobe register
#define CC2420_SAES         0x0E            // Strobe register
#define CC2420_MAIN         0x10
#define CC2420_MDMCTRL0     0x11
#define CC2420_MDMCTRL1     0x12
#define CC2420_RSSI         0x13
#define CC2420_SYNCWORD     0x14
#define CC2420_TXCTRL       0x15
#define CC2420_RXCTRL0      0x16
```

```
77  #define CC2420_RXCTRL1      0x17
78  #define CC2420_FSCTRL       0x18
79  #define CC2420_SECCTRL0     0x19
80  #define CC2420_SECCTRL1     0x1A
81  #define CC2420_BATTMON      0x1B
82  #define CC2420_IOCFG0       0x1C
83  #define CC2420_IOCFG1       0x1D
84  #define CC2420_MANFIDL      0x1E
85  #define CC2420_MANFIDH      0x1F
86  #define CC2420_FSMTC        0x20
87  #define CC2420_MANAND       0x21
88  #define CC2420_MANOR        0x22
89  #define CC2420_AGCCTRL      0x23
90  #define CC2420_AGCTST0      0x24
91  #define CC2420_AGCTST1      0x25
92  #define CC2420_AGCTST2      0x26
93  #define CC2420_FSTST0       0x27
94  #define CC2420_FSTST1       0x28
95  #define CC2420_FSTST2       0x29
96  #define CC2420_FSTST3       0x2A
97  #define CC2420_RXBPFTST     0x2B
98  #define CC2420_FSMSTATE     0x2C
99  #define CC2420_ADCTST       0x2D
100 #define CC2420_DACTST       0x2E
101 #define CC2420_TOPTST       0x2F
102 #define CC2420_RESERVED     0x30
103 #define CC2420_TXFIFO       0x3E
104 #define CC2420_RXFIFO       0x3F
105
106
107 /*****************************************************************************************/
108 /*                                        MAC-FRAME                                      */
109 /*****************************************************************************************/
110
111 #define BEACON_FRAME_TYPE               0
112 #define DATA_FRAME_TYPE                 1
113 #define ACK_FRAME_TYPE                  2
114 #define CMD_FRAME_TYPE                  3
115
116 #define DATA_FRAMECTRL_LO               0x41
117 #define DATA_FRAMECTRL_HI               0xCC
118 #define ACK_FRAMECTRL_LO                0x02
119 #define ACK_FRAMECTRL_HI                0x00
120 #define CMD_FRAMECTRL_LO                0x63
121 #define CMD_FRAMECTRL_HI                0xC8
122
123 #define DST_PAN_ID_L0                   0x12
124 #define DST_PAN_ID_HI                   0x40
125
126 #define DST_ADDRESS_SHORT_LO            0xE1
127 #define DST_ADDRESS_SHORT_HI            0xBE
128
129 #define DST_ADDRESS_EXT_LL_LO           0x2D
130 #define DST_ADDRESS_EXT_LH_LO           0xF7
131 #define DST_ADDRESS_EXT_HL_LO           0x08
132 #define DST_ADDRESS_EXT_HH_LO           0x40
133 #define DST_ADDRESS_EXT_LL_HI           0x00
134 #define DST_ADDRESS_EXT_LH_HI           0xA2
135 #define DST_ADDRESS_EXT_HL_HI           0x13
136 #define DST_ADDRESS_EXT_HH_HI           0x00
137
138 #define SRC_PAN_ID_LO                   0xFF
139 #define SRC_PAN_ID_HI                   0xFF
140
141 #define SRC_ADDRESS_SHORT_LO            0xBE
142 #define SRC_ADDRESS_SHORT_HI            0xE1
143
144 #define SRC_ADDRESS_EXT_LL_LO           0x56
145 #define SRC_ADDRESS_EXT_LH_LO           0x34
146 #define SRC_ADDRESS_EXT_HL_LO           0x12
147 #define SRC_ADDRESS_EXT_HH_LO           0x90
148 #define SRC_ADDRESS_EXT_LL_HI           0x78
149 #define SRC_ADDRESS_EXT_LH_HI           0x56
150 #define SRC_ADDRESS_EXT_HL_HI           0x34
151 #define SRC_ADDRESS_EXT_HH_HI           0x12
152
```

```
153 #define CMD_ASSOCIATION_REQUEST         0x01
154 #define CMD_DISASSOCIATE_NOTIFICATION   0x03
155 #define CMD_DATA_REQUEST                0x04
156 #define CMD_ORPHAN_NOTIFICATION         0x06
157 #define CMD_BEACON_REQUEST              0x07
158
159 #define CRC_LO                          0x00
160 #define CRC_HI                          0x00
161
162 typedef struct
163 {
164     uint8_t frame_control[2];
165     uint8_t sequence_number;
166     uint8_t data[253];
167 } frame_t;
168
169 typedef struct
170 {
171     uint8_t frame_type;
172     uint8_t security_enabled;
173     uint8_t frame_pending;
174     uint8_t ack_request;
175     uint8_t intraPAN;
176     uint8_t dst_addr_mode;
177     uint8_t src_addr_mode;
178 } frame_control_t;
179
180 // rest at moment for class rf2.c
181 #define DEVICE_ADDRESS_H                0xFF
182 #define DEVICE_ADDRESS_L                0xFF
183
184 #define IEEE_ADDRESS_HH_H               0x12
185 #define IEEE_ADDRESS_HL_H               0x34
186 #define IEEE_ADDRESS_LH_H               0x56
187 #define IEEE_ADDRESS_LL_H               0x78
188
189 #define IEEE_ADDRESS_HH_L               0x90
190 #define IEEE_ADDRESS_HL_L               0x12
191 #define IEEE_ADDRESS_LH_L               0x34
192 #define IEEE_ADDRESS_LL_L               0x56
193
194 #define CHANNEL                         15      // chanel between 11 and 26
195
196
197 /*****************************************************************************/
198 /*                                    OTHERS                                 */
199 /*****************************************************************************/
200
201 #define BPS                     11
202
203 #define FALSE                   (0)
204 #define TRUE                    (!FALSE)
205 #define MIN(A,B)                ((A)<(B)?(A):(B))
206 #define MAX(A,B)                ((A)>(B)?(A):(B))
207 #define ABS(X)                  ((X)<0?-(X):(X))
208 #define CLAMP(X,LO,HI)          (((X)<(LO))?(LO):(((HI)<(X))?(HI):(X)))
209 #define SWAP(a,b)               (a^=b,b^=a,a^=b)
210
211 #define MSEC                    232UL
212 #define DELAY(N)                { volatile uint32_t _i = (N); while (0 < _i) _i--; }
213
214 #define SENSOR_RESET_TIME_SEC   1           // each 10 seconds
215 #define SENSOR_SIGN_OF_LIFE     300         // each 5 minutes
216
217
218 #endif // _DEFINITIONS_H_
219
```

```
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Headerfile i2c.h                                                                 */
/*                                                                                   */
/*  Class to use the I2C-Bus                                                         */
/*-----------------------------------------------------------------------------------*/

#ifndef _I2C_H_
#define _I2C_H_

/******** INCLUDES ***************************************************************/
#include <stdint.h>


/******** FUNCTIONS **************************************************************/
void i2c_init  (void);                          // Initialition of the I2C on USART0.
void i2c_write (uint8_t addr, uint8_t n, ...);   // Writes on the I2C Bus.

#endif

```

```c
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Classfile i2c.c                                                                  */
/*                                                                                   */
/*  Class to use the I2C-Bus                                                         */
/*-----------------------------------------------------------------------------------*/

/******** INCLUDES ***************************************************************/
#include "definitions.h"
#include "i2c.h"
#include <io.h>
#include <stdarg.h>


/******** FUNCTIONS **************************************************************/

/***********************************************************************************/
/* Initialition of the I2C on USART0.                                              */
/* (ATTENTION : RS232 use also the USART0 -> Possible conflict)                    */
/*                                                                                 */
/* @DATE      : 09.11.2007                                                         */
/* @AUTHOR    : Ravel                                                              */
/* @INPUT     : -                                                                  */
/* @OUPUT     : -                                                                  */
/* @REVISION  : v1.0                                                               */
/***********************************************************************************/
void i2c_init (void)
{
  P2OUT &= ~BIT7;                          // Invalidation of I2C (disconnection)
  P2DIR |= BIT7;                           // Set I2C connection pin as output
  P3SEL |= BIT1 + BIT3;                    // Choose I2C peripheral pins (SDA and SCL)
  P2OUT |= BIT7;                           // Activation of I2C (connection)s
}

/***********************************************************************************/
/* Writes on the I2C Bus.                                                          */
/*                                                                                 */
/* @DATE      : 09.11.2007                                                         */
/* @AUTHOR    : Ravel                                                              */
/* @INPUT     : uint8_t addr > I2C receiver address                                */
/*              uint8_t n    > Number of transmitted bytes                         */
/*              ...          > Data                                                */
/* @OUPUT     : -                                                                  */
/* @REVISION  : v1.0                                                               */
/***********************************************************************************/
void i2c_write (uint8_t addr, uint8_t n, ...)
{
  va_list ap;
  va_start(ap, n);

  //--- Activation of I2C
  P2OUT |= BIT7;

  //--- Choose UART in mode I2C
  U0CTL = I2C + SYNC;

  U0CTL &= ~I2CEN;
  I2CTCTL |= I2CSSEL_2 + I2CTRX;           // SMCLK + Transmit
  I2CNDAT = n;                             // Nbr. of bytes to transmit
  I2CSA = addr;                            // Addr. of slave destination
  // I2CPSC = 0x00;
  // I2CSCLH = 0x00;
  // I2CSCLL = 0x00;
  U0CTL |= I2CEN;

  U0CTL |= MST;                            // Master mode
  I2CTCTL |= I2CSTT + I2CSTP;              // Initiate transfer

  //--- Sending Data
  while (n--)
  {
    while ((I2CIFG & TXRDYIFG) == 0);      // Wait for transmit ready
    I2CDRB = (uint8_t)va_arg(ap, int);     // Send a byte
  }

```

```
  DELAY(MSEC/2);

  //--- Invalidation of I2C
  P2OUT &= ~BIT7;

  va_end(ap);
}

```

```c
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Headerfile mac_send.h                                                            */
/*                                                                                   */
/*  Composition and transmission of IEEE 802.15.4 frames.                            */
/*-----------------------------------------------------------------------------------*/

#ifndef _MAC_SEND_H_
#define _MAC_SEND_H_


/******** INCLUDES ***************************************************************/
#include <io.h>
#include <stdint.h>


/******** DEFINITIONS ************************************************************/
#define MAC_STATE      0
#define MAC_XYZ_STATE  1


/******** FUNCTIONS **************************************************************/
extern void mac_send(uint8_t event);        // Sends a composed frame (IEEE 802.15.4).

#endif //_MAC_SEND_H_

```

```c
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Classfile mac_send.c                                                             */
/*                                                                                   */
/*  Composition and transmission of IEEE 802.15.4 frames.                            */
/*-----------------------------------------------------------------------------------*/

/******** INCLUDES ***************************************************************/
#include "amr.h"
#include "definitions.h"
#include "mac_send.h"
#include "rf2.h"
#include "variables.h"
#include <io.h>
#include <stdint.h>


/******** FUNCTIONS **************************************************************/

/***********************************************************************************/
/* Composition of Data-Frame (IEEE 802.15.4) to indicate the parking state.         */
/*                                                                                  */
/* @DATE      : 09.11.2007                                                          */
/* @AUTHOR    : Ravel                                                               */
/* @INPUT     : const uint8_t * buffer > pointer with buffer for frame              */
/*              uint8_t state          > parking state (sensor state)              */
/* @OUPUT     : uint8_t                < length of frame                           */
/* @REVISION  : v1.0                                                                */
/***********************************************************************************/
static uint8_t mac_send_set_state(uint8_t * buffer)
{
  //--- Variables
  uint8_t length = 0;

  //--- Generate a MAC-Layer Data frame
  buffer[length++] = DATA_FRAMECTRL_LO;
  buffer[length++] = DATA_FRAMECTRL_HI;
  buffer[length++] = gSequenceNbr++;
  buffer[length++] = (uint8_t) gDstPANId;       // DST_PAN_ID_L0;
  buffer[length++] = gDstPANId >> 8;            // DST_PAN_ID_HI;
  buffer[length++] = (uint8_t) gDstIEEEAddr[3]; // DST_ADDRESS_EXT_LL_LO;
  buffer[length++] = gDstIEEEAddr[3] >> 8;      // DST_ADDRESS_EXT_LH_LO;
  buffer[length++] = (uint8_t) gDstIEEEAddr[2]; // DST_ADDRESS_EXT_HL_LO;
  buffer[length++] = gDstIEEEAddr[2] >> 8;      // DST_ADDRESS_EXT_HH_LO;
  buffer[length++] = (uint8_t) gDstIEEEAddr[1]; // DST_ADDRESS_EXT_LL_HI;
  buffer[length++] = gDstIEEEAddr[1] >> 8;      // DST_ADDRESS_EXT_LH_HI;
  buffer[length++] = (uint8_t) gDstIEEEAddr[0]; // DST_ADDRESS_EXT_HL_HI;
  buffer[length++] = gDstIEEEAddr[0] >> 8;      // DST_ADDRESS_EXT_HH_HI;
  buffer[length++] = (uint8_t) gSrcIEEEAddr[3]; // SRC_ADDRESS_EXT_LL_LO;
  buffer[length++] = gSrcIEEEAddr[3] >> 8;      // SRC_ADDRESS_EXT_LH_LO;
  buffer[length++] = (uint8_t) gSrcIEEEAddr[2]; // SRC_ADDRESS_EXT_HL_LO;
  buffer[length++] = gSrcIEEEAddr[2] >> 8;      // SRC_ADDRESS_EXT_HH_LO;
  buffer[length++] = (uint8_t) gSrcIEEEAddr[1]; // SRC_ADDRESS_EXT_LL_HI;
  buffer[length++] = gSrcIEEEAddr[1] >> 8;      // SRC_ADDRESS_EXT_LH_HI;
  buffer[length++] = (uint8_t) gSrcIEEEAddr[0]; // SRC_ADDRESS_EXT_HL_HI;
  buffer[length++] = gSrcIEEEAddr[0] >> 8;      // SRC_ADDRESS_EXT_HH_HI;
  buffer[length++] = gParkingNumber >> 8;
  buffer[length++] = (uint8_t) gParkingNumber;
  buffer[length++] = gState;
  buffer[length++] = CRC_LO;
  buffer[length++] = CRC_HI;

  //--- Return the lenght of the MAC-Layer Data frame
  return length - 1;
}

/***********************************************************************************/
/* Composition of Data-Frame (IEEE 802.15.4) to indicate the parking state.         */
/*                                                                                  */
/* @DATE      : 09.11.2007                                                          */
/* @AUTHOR    : Ravel                                                               */
/* @INPUT     : const uint8_t * buffer > pointer with buffer for frame              */
/*              uint8_t state          > parking state (sensor state)              */
/* @OUPUT     : uint8_t                < length of frame                           */
/* @REVISION  : v1.0                                                                */
```

```
77  /*****************************************************************************************/
78  static uint8_t mac_send_set_vizgraph(uint8_t * buffer, uint8_t x_state, uint8_t y_state, uint8
79  {
80
81    //--- Variables
82    uint8_t length = 0;
83
84    //--- Generate a MAC-Layer Data frame
85    buffer[length++] = DATA_FRAMECTRL_LO;
86    buffer[length++] = DATA_FRAMECTRL_HI;
87    buffer[length++] = gSequenceNbr++;             // Sequence Number
88    buffer[length++] = (uint8_t) gDstPANId;        // DST_PAN_ID_L0;
89    buffer[length++] = gDstPANId >> 8;             // DST_PAN_ID_HI;
90    buffer[length++] = (uint8_t) gDstIEEEAddr[3]; // DST_ADDRESS_EXT_LL_LO;
91    buffer[length++] = gDstIEEEAddr[3] >> 8;       // DST_ADDRESS_EXT_LH_LO;
92    buffer[length++] = (uint8_t) gDstIEEEAddr[2]; // DST_ADDRESS_EXT_HL_LO;
93    buffer[length++] = gDstIEEEAddr[2] >> 8;       // DST_ADDRESS_EXT_HH_LO;
94    buffer[length++] = (uint8_t) gDstIEEEAddr[1]; // DST_ADDRESS_EXT_LL_HI;
95    buffer[length++] = gDstIEEEAddr[1] >> 8;       // DST_ADDRESS_EXT_LH_HI;
96    buffer[length++] = (uint8_t) gDstIEEEAddr[0]; // DST_ADDRESS_EXT_HL_HI;
97    buffer[length++] = gDstIEEEAddr[0] >> 8;       // DST_ADDRESS_EXT_HH_HI;
98    buffer[length++] = (uint8_t) gSrcIEEEAddr[3]; // SRC_ADDRESS_EXT_LL_LO;
99    buffer[length++] = gSrcIEEEAddr[3] >> 8;       // SRC_ADDRESS_EXT_LH_LO;
100   buffer[length++] = (uint8_t) gSrcIEEEAddr[2]; // SRC_ADDRESS_EXT_HL_LO;
101   buffer[length++] = gSrcIEEEAddr[2] >> 8;       // SRC_ADDRESS_EXT_HH_LO;
102   buffer[length++] = (uint8_t) gSrcIEEEAddr[1]; // SRC_ADDRESS_EXT_LL_HI;
103   buffer[length++] = gSrcIEEEAddr[1] >> 8;       // SRC_ADDRESS_EXT_LH_HI;
104   buffer[length++] = (uint8_t) gSrcIEEEAddr[0]; // SRC_ADDRESS_EXT_HL_HI;
105   buffer[length++] = gSrcIEEEAddr[0] >> 8;       // SRC_ADDRESS_EXT_HH_HI;
106   buffer[length++] = gParkingNumber >> 8;        // Data - Parking nbr higher byte
107   buffer[length++] = (uint8_t) gParkingNumber;   // Data -      "      lower byte
108   buffer[length++] = gState ;                    // Data - parking state
109   buffer[length++] = (uint8_t) x_state;          // Data -    "    lower byte
110   buffer[length++] = (uint8_t) y_state;          // Data -    "    lower byte
111   buffer[length++] = (uint8_t) z_state;          // Data -    "    lower byte
112   buffer[length++] = CRC_LO;                     // FCS lower byte
113   buffer[length++] = CRC_HI;                     // FCS lower byte
114
115   //--- Return the lenght of the MAC-Layer Data frame
116   return length - 1;
117 }
118
119
120  extern uint32_t x_avg_8;
121  extern uint32_t x_avg_32;
122 // extern uint32_t x_avg_64;
123  extern uint32_t y_avg_8;
124  extern uint32_t y_avg_32;
125 // extern uint32_t y_avg_64;
126  extern uint32_t z_avg_8;
127  extern uint32_t z_avg_32;
128 // extern uint32_t z_avg_64;
129
130 /*****************************************************************************************/
131 /* Sends a composed frame (IEEE 802.15.4).                                              */
132 /*                                                                                      */
133 /* @DATE      : 09.11.2007                                                              */
134 /* @AUTHOR    : Ravel                                                                   */
135 /* @INPUT     : uint8_t length > length of the frame (used length in buffer)            */
136 /* @OUPUT     : -                                                                       */
137 /* @REVISION  : v1.0                                                                    */
138 /*****************************************************************************************/
139
140
141
142  void mac_send(uint8_t event)
143 {
144   static int delay_counter = 0;
145   static int delay,delay_level;
146   static int id_send = 0;
147   static int counter = 0;
148
149   int8_t cursor = 1;
150   int16_t x_state,y_state,z_state;
151
152   if (counter==0)
```

```
153     {
154       delay_counter = (delay_counter + gGlitch) % 787;
155
156       delay_level=72*BPS;
157       for(int i=BPS;i>0;i--)
158       {
159         delay_level -= 72;
160         if ( delay_counter >= delay_level)
161         {
162           delay = (delay_counter-delay_level);
163           id_send = i-1;
164           break;
165         }
166       }
167       counter = BPS;
168     }
169     else
170     {
171       counter--;
172     }
173
174     if(counter!=id_send) return;
175
176     DELAY(delay * MSEC);
177
178     if(event == MAC_STATE)
179     {
180       cursor += mac_send_set_state(rf2_buffer + cursor);
181                                                 // set parking state [IEEE802.15.4 Data-frame]
182     }
183     else if(event == MAC_XYZ_STATE)
184     {
185       x_state=x_avg_32+gX_offset;
186       y_state=y_avg_32+gY_offset;
187       z_state=z_avg_32+gZ_offset;
188       x_state = x_state>>2;
189       y_state = y_state>>2;
190       z_state = z_state>>2;
191       if(x_state>255) x_state=255;
192       if(y_state>255) y_state=255;
193       if(z_state>255) z_state=255;
194       if(x_state<0) x_state=0;
195       if(y_state<0) y_state=0;
196       if(z_state<0) z_state=0;
197
198
199       cursor += mac_send_set_vizgraph(rf2_buffer + cursor,x_state, y_state, z_state);
200                                                 // set sensor state [IEEE802.15.4 Data-frame]
201     }
202
203     rf2_buffer[0] = cursor;                     // first character content the frame length
204
205     rf2_send(cursor);
206   }
207
```

```c
/*-----------------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                       */
/*  RAVEL - 2007                                                                           */
/*  Classfile main.c                                                                       */
/*                                                                                         */
/*  Main class of the system.                                                              */
/*-----------------------------------------------------------------------------------------*/

/******** INCLUDES ***************************************************************************/
#include "amr.h"
#include "amr_autoconf.h"
#include "czame.h"
#include "definitions.h"
#include "mac_send.h"
#include "variables.h"
#include "sleep.h"
#include <io.h>
#include <stdint.h>

/******** FUNCTION PROTOTYPES ****************************************************************/
void notify (int16_t report);


/******** FUNCTIONS **************************************************************************/

/*****************************************************************************************/
/* Initialisation of the MSP430F169 microcontroller.                                     */
/*                                                                                       */
/* @DATE      : 09.11.2007                                                               */
/* @AUTHOR    : Ravel                                                                    */
/* @INPUT     : -                                                                        */
/* @OUPUT     : -                                                                        */
/* @REVISION  : v1.0                                                                     */
/*****************************************************************************************/
void init_uc(void)
{
  led_init();

  //--- Clock Init [Basic Clock Modul]
  //    ( GOAL : MCLK = SMCLK = 4 MHz, ACLK=32768 Hz )

  // DCOCTL : DCO Control Register
  // .DCOx = 5; Selection of DCO frequency <see page 116 [4-7] of msp430 manual>
  // .MODx = 20; Modulator Selection
  DCOCTL = (DCO2 + DCO0 + MOD4 + MOD2);

  // BCSCTL1 : Basic Clock System Control Register 1
  // .XT2OFF = 1; Disable the external oscillator for SCLK and SMCLK
  // .XTS    = 0; Set the low frequency mode for LXFT1
  // .DIVAx  = 0; Set the ACLK divisor to 1 [32768 Hz/1 = 32768 Hz]
  // .RSELx  = 7; Internal Resistor. Second register to select of DCO frequency.
  BCSCTL1 = (XT2OFF + RSEL2 + RSEL1 + RSEL0);

  // BCSCTL2 : Basic Clock System Control Register 2
  // .SELMx = 0; Select DCOCLK as source for MCLK
  // .DIVMx = 0; Set the MCLK divisor to 1 [4 MHz/1 = 4 MHz]
  // .SELS  = 0; Select DCOCLK as source for SMCLK
  // .DIVSx = 0; Set the SMCLK divisor to 1 [4 MHz/1 = 4 MHz]
  // .DCOR  = 0; Select internal resistor for DCO
  BCSCTL2 = 0x04;
}

/*******************************************************************************************/
/* [MAIN] - main function.                                                                 */
/*  Explications : - LED 1 :       RF communication -> ON=transmission, OFF=runtime        */
/*                 - LED 2 :       Indicate the parking state -> ON=taken, OFF=free        */
/*                 - LED 1 and 2 : Initialisation -> ON=CEP_initialisation, OFF=runtime    */
/*                                                                                         */
/* @DATE      : 09.11.2007                                                                 */
/* @AUTHOR    : Ravel                                                                      */
/* @INPUT     : -                                                                          */
/* @OUPUT     : -                                                                          */
/* @REVISION  : v1.0                                                                       */
/*******************************************************************************************/
int main (void)
{
```

```c
  uint16_t counter = BPS*60;

  //--- Initialisation
  WDTCTL = WDTPW + WDTHOLD; // Stop WatchDog Timer
  init_uc();                    // Initialisation du microcontroleur
  amr_init_calibration();       // Sensors configuration (search mean value)

  //--- Initialisation of the algorithm
//  czame_config(CZ_CONF_XYZ);    // Algorithm configuration
//  czame_report = &notify;       // Algorithm callback assignment
//  czame_init();                 // Algorithm initialisation

  amr_init();
  amr_start(BPS);

  //--- Endlos loop [Runtime mode]
  for(;;)
  {
    sleep();
    amr_average();
    amr_detection();
    amr_dyn_calib();

    // rf send xyz & state
    mac_send(MAC_XYZ_STATE);

    counter--;

    if(counter == 0)
    {
      amr_degauss();
      counter = BPS*60;
    }

  }
}

/*******************************************************************************/
/* Callback function by detecting a parking state change.                      */
/*                                                                             */
/* @DATE      : 09.11.2007                                                     */
/* @AUTHOR    : Ravel                                                          */
/* @INPUT     : -                                                              */
/* @OUPUT     : -                                                              */
/* @REVISION  : v1.0                                                           */
/*******************************************************************************/
void notify (int16_t report)
{
  if (report == REPORT_FREE)
  {
    if(gState == true)
      gState = false;
  }
  else
  {
    if(gState == false)
      gState = true;
  }
}
```

```c
/*-----------------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                       */
/*  RAVEL - 2007                                                                           */
/*  Classfile main.c                                                                       */
/*                                                                                         */
/*  Main class of the system.                                                              */
/*-----------------------------------------------------------------------------------------*/

/******** INCLUDES ***************************************************************************/
#include "cc2420.h"
#include "rf2.h"
#include "definitions.h"
#include "rs232.h"
#include "variables.h"
#include <io.h>
#include <stdint.h>
#include <signal.h>
#include <string.h>


/******** DEFINITIONS ************************************************************************/
#define VREG_EN     BIT2            // <P2.2> Pin: VREG_EN
#define RS422_TX_EN BIT3            // <P2.3> Pin: DE
#define RS422_RX_EN BIT4            // <P2.4> Pin: nRE
#define ENABLE_I2C  BIT7            // <P2.7> Pin: COM_I2C_CUT
#define CLK_MCLK        BIT4            // <P5.4>
#define CLK_SMCLK       BIT5            // <P5.5>
#define CLK_ACLK        BIT6            // <P5.6>
#define ENABLE_RELAY_1  BIT5            // <P6.5> Pin: uPO_REL_1
#define ENABLE_RELAY_2  BIT6            // <P6.6> Pin: uPO_REL_2


/******** FUNCTION PROTOTYPES ****************************************************************/
void init_uc (void);


/******** FUNCTIONS **************************************************************************/

/*****************************************************************************************/
/* [MAIN] - main function.                                                               */
/*  Explications : - LED 1 :       RF communication -> ON=transmission, OFF=runtime       */
/*                 - LED 1 and 2 : Initialisation -> ON=CEP_initialisation, OFF=runtime   */
/*                                                                                       */
/* @DATE      : 09.11.2007                                                               */
/* @AUTHOR    : Ravel                                                                    */
/* @INPUT     : -                                                                        */
/* @OUPUT     : -                                                                        */
/* @REVISION  : v1.0                                                                     */
/*****************************************************************************************/
int main (void)
{
  //--- Variables
  uint8_t data, akt_state, flag_frame, flag_crc = 0;
  uint16_t nbr, x, y, z = 0;
  int num_byte_frame = 0;
  flag_frame = 0;

  //--- Stop Watchdog
  WDTCTL = WDTPW + WDTHOLD;

  //--- Enable the interrupts
  _EINT();

  //--- Initialisation
  init_uc();                        // Initialisation of the MSP430F169 microcontroller
  led_init();
  rs232_init();                     // Initialisation of the RS232 on USART0
  rf2_init();

  //--- Enable RX
  rf2_receive_on();

  //--- Endlos loop [Runtime mode]
  for(;;)
  {

```

```c
    if (cc2420_receive(&data))
    {
      if (num_byte_frame==0)
      {
        num_byte_frame = data;
        //rs232_printf("[nbf:%u]\n",num_byte_frame);
        if(data == 26) flag_frame=1;
        if(data == 29) flag_frame=2;
      }
      else
      {
        //---Sampling values (only for debug)
        if (flag_frame == 2)
        {
          //---Resultat interpretation
          //if (num_byte_frame==11)         // parking address H
          //    nbr = data << 8;
          if (num_byte_frame==7)            // parking address L
            nbr = data;

          if (num_byte_frame==6)            // parking state
            akt_state = data;

          /*
          if (num_byte_frame==8)            // x value H
            x = data << 8;
          if (num_byte_frame==7)            // x value L
            x |= data;
          */
          if (num_byte_frame==5)            // x value L
            x = data;

          /*
          if (num_byte_frame==6)            // y value H
            y = data << 8;
          if (num_byte_frame==5)            // y value L
            y |= data;
          */
          if (num_byte_frame==4)            // y value L
            y = data;

          /*
          if (num_byte_frame==4)            // z value H
            z = data << 8;
          if (num_byte_frame==3)            // z value L
            z |= data;
          */
          if (num_byte_frame==3)            // z value L
            z = data;

          //---CRC check
          if (num_byte_frame==1)
          {
            flag_crc = data & 0x80;
            num_byte_frame--;
          }

          //---Send result over RS232
          if (num_byte_frame==0)
          {
            //rs232_printf("x:%u, y:%u, z:%u", x, y, z);
            if(flag_crc)
            {
                rs232_putc(0xA1);
                rs232_putc(nbr);
                rs232_putc(akt_state);
                rs232_putc(x);
                rs232_putc(y);
                rs232_putc(z);
                //rs232_printf("%x", x>>2);
                //rs232_printf("0\t%1u\t%1u\t%1u\t0\n",
                //  x,
                //  y,
                //  z);
                //rs232_printf("<%x %x %x %x %x %x>\n", 0xA1, nbr, akt_state, x>>2, y>>2, z>>2
            }
```

```c
153             flag_frame =0;
154             num_byte_frame++;
155           }
156         }
157         else if (flag_frame == 1)  //---State has changed
158         {
159           //---Resultat interpretation
160
161           //if (num_byte_frame==5)          // parking address H
162           //    data_rs = data;
163           if (num_byte_frame==4)            // parking address L
164             nbr = data;
165
166           if (num_byte_frame==3)            // parking state
167             akt_state = data;
168
169           //---CRC check
170           if (num_byte_frame==1)
171           {
172             flag_crc = data & 0x80;
173             num_byte_frame--;
174           }
175
176           //---Send result over RS232
177           if (num_byte_frame==0)
178           {
179             if(flag_crc)
180             {
181               rs232_putc(0xA2);
182               rs232_putc(nbr);
183               rs232_putc(akt_state);
184               //rs232_printf("<%x %x %x>\n", 0xA2, nbr, akt_state);
185             }
186             flag_frame =0;
187             num_byte_frame++;
188           }
189         }
190
191         num_byte_frame--;
192       }
193     }
194   }
195 }
196
197 /*****************************************************************************************/
198 /* Initialisation of the MSP430F169 microcontroller.                                     */
199 /*                                                                                       */
200 /* @DATE      : 09.11.2007                                                               */
201 /* @AUTHOR    : Ravel                                                                    */
202 /* @INPUT     : -                                                                        */
203 /* @OUPUT     : -                                                                        */
204 /* @REVISION  : v1.0                                                                     */
205 /*****************************************************************************************/
206 void init_uc(void)
207 {
208   //--- <Port 5> Clock outputs (to control the correct frequency on each pin)
209   P5DIR |= (CLK_ACLK + CLK_SMCLK + CLK_MCLK);     // Set Clock pins as output
210   P5SEL |= (CLK_ACLK + CLK_SMCLK + CLK_MCLK);     // Select Clock mode (periph.) of the pin
211
212   //--- <Port 6> Enable the relays
213   //P6DIR |= (ENABLE_RELAY_1 + ENABLE_RELAY_2);   // Set relay pins as output
214   //P6OUT |= (ENABLE_RELAY_1 + ENABLE_RELAY_2);   // HIGH level to not consume power
215
216   //--- <Port 2> Enable RS422_RX; Disable the Chipcon CC2420, the RS422_TX and the I2C
217   P2DIR |= (VREG_EN + RS422_TX_EN + RS422_RX_EN + ENABLE_I2C);
218                                                   // Set pins as output
219   P2OUT |= RS422_RX_EN;                           // Enable RS422_RX
220   P2OUT &= ~(VREG_EN + RS422_TX_EN + ENABLE_I2C); // Disable CC2420, RS422_TX and I2C
221
222   //--- Clock Init [Basic Clock Modul]
223   //    ( GOAL : MCLK = SMCLK = 4 MHz, ACLK=32768 Hz )
224
225   // DCOCTL : DCO Control Register
226   // .DCOx = 5; Selection of DCO frequency <see page 116 [4-7] of msp430 manual>
227   // .MODx = 20; Modulator Selection
228   DCOCTL = (DCO2 + DCO0 + MOD4 + MOD2);
```

```c

  // BCSCTL1 : Basic Clock System Control Register 1
  // .XT2OFF = 1; Disable the external oscillator for SCLK and SMCLK
  // .XTS    = 0; Set the low frequency mode for LXFT1
  // .DIVAx  = 0; Set the ACLK divisor to 1 [32768 Hz/1 = 32768 Hz]
  // .RSELx  = 7; Internal Resistor. Second register to select of DCO frequency.
  BCSCTL1 = (XT2OFF + RSEL2 + RSEL1 + RSEL0);

  // BCSCTL2 : Basic Clock System Control Register 2
  // .SELMx = 0; Select DCOCLK as source for MCLK
  // .DIVMx = 0; Set the MCLK divisor to 1 [4 MHz/1 = 4 MHz]
  // .SELS  = 0; Select DCOCLK as source for SMCLK
  // .DIVSx = 0; Set the SMCLK divisor to 1 [4 MHz/1 = 4 MHz]
  // .DCOR  = 0; Select internal resistor for DCO
  BCSCTL2 = 0x00;
}

```

```c
/*-----------------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                       */
/*  RAVEL - 2007                                                                           */
/*  Classfile main.c                                                                       */
/*                                                                                         */
/*  Main class of the system.                                                              */
/*-----------------------------------------------------------------------------------------*/

/******** INCLUDES ***************************************************************************/
#include "cc2420.h"
#include "rf2.h"
#include "definitions.h"
#include "mac_send.h"
#include "variables.h"
#include <io.h>
#include <stdint.h>
#include <signal.h>
#include <string.h>


/******** DEFINITIONS ************************************************************************/
#define VREG_EN     BIT2            // <P2.2> Pin: VREG_EN
#define RS422_TX_EN BIT3            // <P2.3> Pin: DE
#define RS422_RX_EN BIT4            // <P2.4> Pin: nRE
#define ENABLE_I2C  BIT7            // <P2.7> Pin: COM_I2C_CUT
#define CLK_MCLK        BIT4            // <P5.4>
#define CLK_SMCLK       BIT5            // <P5.5>
#define CLK_ACLK        BIT6            // <P5.6>
#define ENABLE_RELAY_1  BIT5            // <P6.5> Pin: uPO_REL_1
#define ENABLE_RELAY_2  BIT6            // <P6.6> Pin: uPO_REL_2


/******** FUNCTION PROTOTYPES ****************************************************************/
void init_uc (void);


/******** FUNCTIONS **************************************************************************/

/*****************************************************************************************/
/* [MAIN] - main function.                                                               */
/*  Explications : - LED 1 :       RF communication -> ON=transmission, OFF=runtime       */
/*                 - LED 1 and 2 : Initialisation -> ON=CEP_initialisation, OFF=runtime   */
/*                                                                                       */
/* @DATE      : 09.11.2007                                                               */
/* @AUTHOR    : Ravel                                                                    */
/* @INPUT     : -                                                                        */
/* @OUPUT     : -                                                                        */
/* @REVISION  : v1.0                                                                     */
/*****************************************************************************************/
int main (void)
{
  //--- Variables
  uint8_t data, flag_frame, flag_crc = 0;
  int16_t num_byte_frame = 0;

  //--- Stop Watchdog
  WDTCTL = WDTPW + WDTHOLD;

  //--- Enable the interrupts
  _EINT();

  //--- Initialisation
  init_uc();                        // Initialisation of the MSP430F169 microcontroller
  led_init();
  rf2_init();

  //--- Enable RX
  rf2_receive_on();

  //--- Endlos loop [Runtime mode]
  for(;;)
  {

    if (cc2420_receive(&data))
    {
      if (num_byte_frame==0)
```

```
77          {
78            num_byte_frame = data;
79            //rs232_printf("[nbf:%u]\n",num_byte_frame);
80            if(data == 26) flag_frame=1;
81            if(data == 29) flag_frame=2;
82          }
83          else
84          {
85            //---Sampling values (only for debug)
86            if (flag_frame == 2)
87            {
88              //---Resultat interpretation
89              if (num_byte_frame==8)            // parking address H
90                gNbr_rec = data << 8;
91              if (num_byte_frame==7)            // parking address L
92                gNbr_rec |= data;
93
94              if (num_byte_frame==6)            // parking state
95                gState_rec = data;
96
97              /*
98              if (num_byte_frame==8)            // x value H
99                gX_rec = data << 8;
100             if (num_byte_frame==7)            // x value L
101               gX_rec |= data;
102             */
103             if (num_byte_frame==5)            // x value L
104               gX_rec = data;
105
106             /*
107             if (num_byte_frame==6)            // y value H
108               gY_rec = data << 8;
109             if (num_byte_frame==5)            // y value L
110               gY_rec |= data;
111             */
112             if (num_byte_frame==4)            // y value L
113               gY_rec = data;
114
115             /*
116             if (num_byte_frame==4)            // z value H
117               gZ_rec = data << 8;
118             if (num_byte_frame==3)            // z value L
119               gZ_rec |= data;
120             */
121             if (num_byte_frame==3)            // z value L
122               gZ_rec = data;
123
124             //---CRC check
125             if (num_byte_frame==1)
126             {
127               flag_crc = data & 0x80;
128               num_byte_frame--;
129             }
130
131             //---Send result over RS232
132             if (num_byte_frame==0)
133             {
134               //rs232_printf("x:%u, y:%u, z:%u", x, y, z);
135               if(flag_crc)
136               {
137                 // rf send xyz & state
138                 mac_send(REPEAT_XYZ_STATE);
139               }
140               flag_frame =0;
141               num_byte_frame++;
142             }
143           }
144           else if (flag_frame == 1)  //---State has changed
145           {
146             //---Resultat interpretation
147
148             if (num_byte_frame==5)            // parking address H
149               gNbr_rec = data << 8;
150             if (num_byte_frame==4)            // parking address L
151               gNbr_rec |= data;
152
```

```c
153           if (num_byte_frame==3)            // parking state
154             gState_rec = data;
155
156           //---CRC check
157           if (num_byte_frame==1)
158           {
159             flag_crc = data & 0x80;
160             num_byte_frame--;
161           }
162
163           //---Send result over RS232
164           if (num_byte_frame==0)
165           {
166             if(flag_crc)
167             {
168               // rf send xyz & state
169               mac_send(REPEAT_STATE);
170             }
171             flag_frame =0;
172             num_byte_frame++;
173           }
174         }
175
176         num_byte_frame--;
177       }
178     }
179   }
180 }
181
182 /*****************************************************************************************/
183 /* Initialisation of the MSP430F169 microcontroller.                                     */
184 /*                                                                                       */
185 /* @DATE      : 09.11.2007                                                               */
186 /* @AUTHOR    : Ravel                                                                    */
187 /* @INPUT     : -                                                                        */
188 /* @OUPUT     : -                                                                        */
189 /* @REVISION  : v1.0                                                                     */
190 /*****************************************************************************************/
191 void init_uc(void)
192 {
193   //--- <Port 5> Clock outputs (to control the correct frequency on each pin)
194   P5DIR |= (CLK_ACLK + CLK_SMCLK + CLK_MCLK);     // Set Clock pins as output
195   P5SEL |= (CLK_ACLK + CLK_SMCLK + CLK_MCLK);     // Select Clock mode (periph.) of the pin
196
197   //--- <Port 6> Enable the relays
198   //P6DIR |= (ENABLE_RELAY_1 + ENABLE_RELAY_2);   // Set relay pins as output
199   //P6OUT |= (ENABLE_RELAY_1 + ENABLE_RELAY_2);   // HIGH level to not consume power
200
201   //--- <Port 2> Enable RS422_RX; Disable the Chipcon CC2420, the RS422_TX and the I2C
202   P2DIR |= (VREG_EN + RS422_TX_EN + RS422_RX_EN + ENABLE_I2C);
203                                                   // Set pins as output
204   P2OUT |= RS422_RX_EN;                           // Enable RS422_RX
205   P2OUT &= ~(VREG_EN + RS422_TX_EN + ENABLE_I2C); // Disable CC2420, RS422_TX and I2C
206
207   //--- Clock Init [Basic Clock Modul]
208   //    ( GOAL : MCLK = SMCLK = 4 MHz, ACLK=32768 Hz )
209
210   // DCOCTL : DCO Control Register
211   // .DCOx = 5; Selection of DCO frequency <see page 116 [4-7] of msp430 manual>
212   // .MODx = 20; Modulator Selection
213   DCOCTL = (DCO2 + DCO0 + MOD4 + MOD2);
214
215   // BCSCTL1 : Basic Clock System Control Register 1
216   // .XT2OFF = 1; Disable the external oscillator for SCLK and SMCLK
217   // .XTS    = 0; Set the low frequency mode for LXFT1
218   // .DIVAx  = 0; Set the ACLK divisor to 1 [32768 Hz/1 = 32768 Hz]
219   // .RSELx  = 7; Internal Resistor. Second register to select of DCO frequency.
220   BCSCTL1 = (XT2OFF + RSEL2 + RSEL1 + RSEL0);
221
222   // BCSCTL2 : Basic Clock System Control Register 2
223   // .SELMx = 0; Select DCOCLK as source for MCLK
224   // .DIVMx = 0; Set the MCLK divisor to 1 [4 MHz/1 = 4 MHz]
225   // .SELS  = 0; Select DCOCLK as source for SMCLK
226   // .DIVSx = 0; Set the SMCLK divisor to 1 [4 MHz/1 = 4 MHz]
227   // .DCOR  = 0; Select internal resistor for DCO
228   BCSCTL2 = 0x00;
```

```
}

```

```c
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Headerfile pwm.h                                                                 */
/*                                                                                   */
/*  Class to use the PWM                                                             */
/*-----------------------------------------------------------------------------------*/

#ifndef _PWM_
#define _PWM_

/******** FUNCTIONS ***************************************************************/
extern void pwm_on (void);
extern void pwm_off (void);


#endif

```

```c
/*-----------------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                       */
/*  RAVEL - 2007                                                                           */
/*  Classfile pwm.c                                                                        */
/*                                                                                         */
/*  Class to use the PWM                                                                   */
/*-----------------------------------------------------------------------------------------*/

/******** INCLUDES ***************************************************************************/
#include <io.h>
#include "definitions.h"


/******** FUNCTIONS **************************************************************************/

/*****************************************************************************************/
/* Turns ON the PWM.                                                                     */
/*                                                                                       */
/* @DATE      : 27.11.2007                                                               */
/* @AUTHOR    : Ravel                                                                    */
/* @INPUT     : -                                                                        */
/* @OUPUT     : -                                                                        */
/* @REVISION  : v1.0                                                                     */
/*****************************************************************************************/
void pwm_on(void)
{
  /*
  P1OUT &= ~BIT3;
  */
  P1DIR |= BIT3;
  P1SEL |= BIT3;                            // Select peripheral mode of pin

  //---Configuration of PWM
  TACCR0 = 1;                               // PWM Period
  TACCTL2 = OUTMOD_7;                       // CCR2 reset/set
  TACCR2 = 1;                               // CCR2 PWM duty cycle

  //---Configuration of timer A
  TACTL = TASSEL_2 + MC_1;                  // ACLK, Up mode -> Timer counts to CCR0
  DELAY(1*MSEC);
}

/*****************************************************************************************/
/* Turns OFF the PWM.                                                                    */
/*                                                                                       */
/* @DATE      : 27.11.2007                                                               */
/* @AUTHOR    : Ravel                                                                    */
/* @INPUT     : -                                                                        */
/* @OUPUT     : -                                                                        */
/* @REVISION  : v1.0                                                                     */
/*****************************************************************************************/
void pwm_off(void)
{
  // P1OUT |= BIT3;
  // P1DIR |= BIT3;
  // ACTL = 0;                              // Reset timer A
  P1SEL &= ~BIT3;                           // Select PIO mode of pin
  P1OUT &= ~BIT3;                           // Set Low level
}
```

```
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Headerfile ravelfunclib.h                                                        */
/*                                                                                   */
/*  Library of helpfully functions.                                                  */
/*-----------------------------------------------------------------------------------*/

#ifndef _RAVELFUNCLIB_H_
#define _RAVELFUNCLIB_H_

/******** INCLUDES ***************************************************************/
#include <stdint.h>


/******** FUNCTIONS **************************************************************/
uint8_t setAscii2DecHexValue(uint8_t);          // calculate decimal HEX value from ascii char.

#endif  // endif _RAVELFUNCLIB_H_

```

```c
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Classfile ravelfunclib.c                                                         */
/*                                                                                   */
/*  Library of helpfully functions.                                                  */
/*-----------------------------------------------------------------------------------*/

/******** INCLUDES ***************************************************************/
#include "definitions.h"
#include "ravelfunclib.h"
#include <stdint.h>


/******** FUNCTIONS **************************************************************/

/***********************************************************************************/
/* Returns the decimal Hex value of an ascii character.                             */
/* (range 0-9, A-F, a-f)                                                            */
/*                                                                                  */
/* @DATE      : 09.11.2007                                                          */
/* @AUTHOR    : Ravel                                                               */
/* @INPUT     : -                                                                   */
/* @OUPUT     : -                                                                   */
/* @REVISION  : v1.0                                                                */
/***********************************************************************************/
uint8_t setAscii2DecHexValue(uint8_t ascii)
{
  uint8_t hexValue;

  // caluculate hexadecimal value in decimal format
  if(ascii >= 65 && ascii <= 70)            // ASCII 65=A --> 70=F
  {
    hexValue = ascii - 65 + 10;
  }
  else if (ascii >= 97 && ascii <= 102)     // ASCII 97=a --> 102=f
  {
    hexValue = ascii - 97 + 10;
  }
  else if (ascii >= 48 && ascii <= 57)      // ASCII 48=0 --> 57=9
  {
    hexValue = ascii - 48;
  }
  else
  {
    hexValue = 0;
  }

  // return calculated value
  return hexValue;
}

```

```c
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Headerfile rf2.h                                                                 */
/*                                                                                   */
/*  Class who administred the RF functions of the CC2420.                            */
/*-----------------------------------------------------------------------------------*/

#ifndef _RF2_H
#define _RF2_H


/******** INCLUDES ***************************************************************/
#include <stdint.h>


/******** FUNCTIONS **************************************************************/
void rf2_init (void);                // Initialition of the CC2420.
                                     // Settings : power_on(), config, PAN ID, addresse

void rf2_power_on (void);            // Starting of the regulator of the CC2420.
                                     // (POWER : ON, XTAL : OFF, RF : OFF).
                                     // - No register is configured but the CC2420 is
                                     //   found in mode wakes up after this function.

void rf2_power_off (void);           // Stopping of the regulator of the CC2420.
                                     // (-> POWER : OFF, XTAL : OFF, RF : OFF).
                                     // - Consumption = 0.02 uA
                                     // - The registers of configurations are not safeguarded
                                     //   in this operating mode.

void rf2_sleep (void);               // Deactivation of the CC2420.
                                     // (-> POWER : ON, XTAL : OFF, RF : OFF).
                                     // - Consumption = 20 uA.
                                     // - The registers of configuration are safeguarded in
                                     //   this operating mode.

void rf2_idle (void);                // CC2420 in waked up mode.
                                     // (-> POWER : ON, XTAL : ON, RF : OFF).
                                     // - Consumption = 256 uA.
                                     // - After this operations the CC2420 is neither in
                                     //   reception mode nor in emission mode.

uint8_t rf2_send (uint8_t);          // Sends a frame of raw data <param : length>.
                                     // - After a sending, the CC2420 is found by default in
                                     //   reception mode.
                                     //   return 0 if success

void rf2_receive_on (void);          // Setting in reception mode of the CC2420.
                                     // - Consumption = 18 mA.
                                     //   to put a callback!

void rf2_receive_off (void);         // Stop of the reception mode.
                                     // - At the exit of this function, the cc2420 is in the

// extern void (*rf2_recv_callback) (uint8_t size);
                                     // callback function at the time of the reception of a
                                     // frame.

#endif //_RF2_H
```

```c
/*-----------------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                       */
/*  RAVEL - 2007                                                                           */
/*  Classfile rf2.c                                                                        */
/*                                                                                         */
/*  Class who administred the RF functions of the CC2420.                                  */
/*-----------------------------------------------------------------------------------------*/

/******** INCLUDES ***************************************************************************/
#include "cc2420.h"
#include "definitions.h"
#include "rf2.h"
#include "variables.h"
#include <io.h>
#include <stdint.h>


/******** FUNCTION - PREDECLARATIONS OF STATIC FUNCTIONS ***********************************/
static void rf2_set_channel(uint8_t);       // sets channel number <param between 11 and 26>


/******** CALLBACK FUNCTIONS *****************************************************************/
void (*rf2_recv_callback) (uint8_t size) = 0;


/******** FUNCTIONS **************************************************************************/

/*****************************************************************************************/
/* Initialition of the CC2420.                                                           */
/* Settings : power_on(), config, PAN ID, address                                        */
/*                                                                                       */
/* @DATE      : 09.11.2007                                                               */
/* @AUTHOR    : Ravel                                                                    */
/* @INPUT     : -                                                                        */
/* @OUPUT     : -                                                                        */
/* @REVISION  : v1.0                                                                     */
/*****************************************************************************************/
void rf2_init (void)
{
  //--- Variables
  uint16_t data[6];

  //--- Assignment of the callback functions
  //fifop_callback = &rf2_receive;          // Received frame
  //fifo_callback = &rf2_overflow;          // Overflow

  //--- Initialisation of CC2420
  cc2420_init();                            // Initialisation of CC2420

  //--- Set Register
  cc2420_set_reg(CC2420_MDMCTRL0,0x22E2);   // Set Modem Control Register 0
                                            // .0x00-0x03 : PREAMBLE_LENGTH      0010
                                            // .0x04      : AUTOACK              0
                                            // .0x05      : AUTOCRC              1
                                            // .0x06-0x07 : CCA_MODE             11
                                            // .0x08-0x0A : CCA_HYST             010
                                            // .0x0B      : ADR_DECODE           0
                                            // .0x0C      : PAN_COORDINATOR      0
                                            // .0x0D      : RESERVED_FRAME_MODE  1
                                            // .0x0E-0x0F : - (reserved)         00

  cc2420_set_reg(CC2420_MDMCTRL1,0x0500);   // Set Modem Control Register 1
                                            // .0x00-0x01 : RX_MODE              00
                                            // .0x02-0x03 : TX_MODE              00
                                            // .0x04      : MODULATION_MODE      0
                                            // .0x05      : DEMOD_AVG_MODE       0
                                            // .0x06-0x0A : CORR_THR             10100
                                            // .0x0B-0x0F : - (reserved)         00000

  rf2_set_channel(CHANNEL);                 // Set RF channel <2.4 GHz between 11-26>

  cc2420_set_reg(CC2420_IOCFG0,0x0805);     // Set I/O Control Register 0
                                            // .0x00-0x06 : FIFOP_THR            0000101
                                            // .0x07      : CCA_POLARITY         0
                                            // .0x08      : SFD_POLARITY         0
                                            // .0x09      : FIFOP_POLARITY       0
```

```
                                          // .0x0A      : FIFO_POLARITY        0
                                          // .0x0B      : BCN_ACCEPT           1
                                          // .0x0C-0x0F : - (reserved)         0000
                                          // --> authorization of the "beacon" which have
                                          //     a good PAN. All the pins are active at
                                          //     the high level and FIFOP threshold = F

  cc2420_set_reg(CC2420_SECCTRL0,0x01C4);   // Set Security Control Register 0
                                          // .0x00-0x01 : SEC_MODE             00
                                          // .0x02-0x04 : SEC_M                001
                                          // .0x05      : SEC_RXKEYSEL         0
                                          // .0x06      : SEC_TXKEYSEL         1
                                          // .0x07      : SEC_SAKEYSEL         1
                                          // .0x08      : SEC_CBC_HEAD         1
                                          // .0x09      : RXFIFO_PROTECTION    0
                                          // .0x0A-0x0F : - (reserved)         000000
                                          // --> invalidation of safety (see for TX and RX)

  //--- Set Chipcon CC2420 network parameter <see chipcon manual p.31 table.6>
  data[0] = gSrcIEEEAddr[3];              // set 64-bit extended address LL
  data[1] = gSrcIEEEAddr[2];              //              "              LH
  data[2] = gSrcIEEEAddr[1];              //              "              HL
  data[3] = gSrcIEEEAddr[0];              //              "              HH
  data[4] = gDstPANId;                    // set 16-bit PAN identifier
  data[5] = 0xFFFF;                       // set 16-bit Short address
  cc2420_set_reg_ram(0x60,6,data);        // writing the address of the card and the PAN

}

/*********************************************************************************/
/* Starting of the regulator of the CC2420 (-> POWER : ON, XTAL : OFF, RF : OFF).        */
/* - No register is configured but the CC2420 is found in mode "wakes up" after this func.*/
/*                                                                                       */
/* @DATE      : 09.11.2007                                                               */
/* @AUTHOR    : Ravel                                                                    */
/* @INPUT     : -                                                                        */
/* @OUPUT     : -                                                                        */
/* @REVISION  : v1.0                                                                     */
/*********************************************************************************/
void rf2_power_on (void)
{
  CC2420_VREG_ON;                         // Startup of the CC2420 supply
}

/*********************************************************************************/
/* Stopping of the regulator of the CC2420 (-> POWER : OFF, XTAL : OFF, RF : OFF).       */
/* - Consumption = 0.02 uA                                                               */
/* - The registers of configurations are not safeguarded in this operating mode.         */
/*                                                                                       */
/* @DATE      : 09.11.2007                                                               */
/* @AUTHOR    : Ravel                                                                    */
/* @INPUT     : -                                                                        */
/* @OUPUT     : -                                                                        */
/* @REVISION  : v1.0                                                                     */
/*********************************************************************************/
void rf2_power_off (void)
{
  CC2420_VREG_OFF;                        // Stop-over of the CC2420 supply
}

/*********************************************************************************/
/* Deactivation of the CC2420 (-> POWER : ON, XTAL : OFF, RF : OFF).                     */
/* - Consumption = 20 uA                                                                 */
/* - The registers of configuration are safeguarded in this operating mode.              */
/*                                                                                       */
/* @DATE      : 09.11.2007                                                               */
/* @AUTHOR    : Ravel                                                                    */
/* @INPUT     : -                                                                        */
/* @OUPUT     : -                                                                        */
/* @REVISION  : v1.0                                                                     */
/*********************************************************************************/
void rf2_sleep (void)
{
  cc2420_strobe(CC2420_SXOSCOFF);         // Stop-over of the oscillator with quartz
}
```

```
153 /*****************************************************************************************/
154 /* CC2420 in waked up mode (-> POWER : ON, XTAL : ON, RF : OFF).                         */
155 /* - Consumption = 256 uA                                                                */
156 /* - After this operations the CC2420 is neither in reception mode nor in emission mode.  */
157 /*                                                                                       */
158 /* @DATE      : 09.11.2007                                                               */
159 /* @AUTHOR    : Ravel                                                                    */
160 /* @INPUT     : -                                                                        */
161 /* @OUPUT     : -                                                                        */
162 /* @REVISION  : v1.0                                                                     */
163 /*****************************************************************************************/
164 void rf2_idle (void)
165 {
166   //--- Variables
167   uint8_t status;
168
169   //--- Check quartz state
170   status = cc2420_get_status();             // Read status register of the CC2420
171
172   if (!(status & 0x40))                     // If the crystal oscillator isn't running...
173   {
174     cc2420_strobe(CC2420_SXOSCON);          // ...turn it on
175     do
176     {
177       status = cc2420_get_status();
178     }
179     while(!(status & 0x40));                // ... and make an attempt that it's stabilized
180   }
181   else                                      // Otherwise...
182   {
183     cc2420_strobe(CC2420_SRFOFF);           // ...disable RX/TX and frequency synthesizer
184   }
185 }
186
187 /*****************************************************************************************/
188 /* Sends a frame of raw data.                                                            */
189 /* - After a sending, the CC2420 is found by default in reception mode.                  */
190 /*                                                                                       */
191 /* @DATE      : 09.11.2007                                                               */
192 /* @AUTHOR    : Ravel                                                                    */
193 /* @INPUT     : uint8_t length > Number of bytes to transmit                             */
194 /* @OUPUT     : uint8_t        < result of the transmission                              */
195 /* @REVISION  : v1.0                                                                     */
196 /*****************************************************************************************/
197 uint8_t rf2_send (uint8_t length)
198 {
199   rf2_init();
200
201   //--- Variables
202   uint8_t status;
203
204   //--- Preparation to send the frame
205   cc2420_strobe(CC2420_SFLUSHTX);           // Flush the TX FIFO buffer
206   cc2420_strobe(CC2420_SRXON);              // Enable RX
207
208   cc2420_write_fifo(length, rf2_buffer);    // Set frame into TX FIFO buffer
209   cc2420_strobe(CC2420_STXONCCA);           // Send frame according to the channel state CCA
210   // cc2420_strobe(CC2420_STXON);           // Send frame without according to the channel
211                                             //  state CCA
212   do
213   {
214     status = cc2420_get_status();           // Get status-bits
215   }
216   while (status & 0x08);                    // Wait as long as bit TX_ACTIVE is idle
217
218   rf2_power_off();
219
220   return 0;                                                          // ...todo...
221 }
222
223 /*****************************************************************************************/
224 /* Setting in reception mode of the CC2420.                                              */
225 /* - Consumption = 18 mA                                                                 */
226 /* - After this operations the CC2420 is neither in reception mode nor in emission mode.  */
227 /*                                                                                       */
228 /* @DATE      : 09.11.2007                                                               */
```

```c
/* @AUTHOR    : Ravel                                                          */
/* @INPUT     : -                                                              */
/* @OUPUT     : -                                                              */
/* @REVISION  : v1.0                                                           */
/*******************************************************************************/
void rf2_receive_on (void)
{
  //--- Variables
  uint8_t status;

  //--- Turn on RF receiption
  cc2420_strobe(CC2420_SRXON);              // Enable RX
  do                                        // to make it more proper (not necessary)
  {
    status = cc2420_get_status();           // Get status-bits
  }
  while(!(status & 0x44));                  // Wait as long as the quartz is stabilized and
                                            //  the PLL is locked
  cc2420_strobe(CC2420_SFLUSHRX);           // Flush the RX FIFO buffer
  cc2420_strobe(CC2420_SFLUSHRX);           //  --> 2 times <see chipcon CC2420 manual p.33>
}

/*******************************************************************************/
/* Stop of the reception mode.                                                 */
/* - At the exit of this function, the CC2420 is in the "waked up" mode.       */
/*                                                                             */
/* @DATE      : 09.11.2007                                                     */
/* @AUTHOR    : Ravel                                                          */
/* @INPUT     : -                                                              */
/* @OUPUT     : -                                                              */
/* @REVISION  : v1.0                                                           */
/*******************************************************************************/
void rf2_receive_off (void)
{
  cc2420_strobe(CC2420_SRFOFF);             // Disable RX/TX and frequency synthesizer
}


/*******************************************************************************/
/* [static] - Sets channel number <param between 11 and 26 for 2.4 GHz>        */
/*                                                                             */
/* @DATE      : 09.11.2007                                                     */
/* @AUTHOR    : Ravel                                                          */
/* @INPUT     : uint8_t channel > Channel number                               */
/* @OUPUT     : -                                                              */
/* @REVISION  : v1.0                                                           */
/*******************************************************************************/
static void rf2_set_channel (uint8_t channel)
{
  //---Variables
  uint16_t freq = 0;

  // fc = 2048 + FREQ MHz
  //      --> fc = 2405 + 5 * (k-11) MHz  with k = channel [11->26]
  //               2048 + 357 + 5 * (k-11)
  //                     |----------------|
  //                            FREQ

  //--- Calculate the channel frequency
  freq = channel - 11;                      // (k-11)
  freq = (uint16_t) (freq + (freq << 2));   // 5 * (k-11)
  freq = freq + 357 + 0x4000;               // 357 + 5 * (k-11) + other register definitions

  cc2420_set_reg(CC2420_FSCTRL, freq);      // Frequency Synthesizer Control and Status
                                            // .0x00-0x09 : FREQ                0101111001
                                            // .0x0A      : LOCK_STATUS         0
                                            // .0x0B      : LOCK_LENGTH         0
                                            // .0x0C      : CAL_RUNNING         0
                                            // .0x0D      : CAL_DONE            0
                                            // .0x0E-0x0F : LOCK_THR            01
}
```

```
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Headerfile rs232.h                                                               */
/*                                                                                   */
/*  Class to use the RS232                                                           */
/*-----------------------------------------------------------------------------------*/

#ifndef __RS232__
#define __RS232__


/******** FUNCTIONS ***************************************************************/
void rs232_init   (void);
void rs232_putc   (char);
void rs232_printf (const char *, ...);

#endif

```

```c
/*-----------------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                       */
/*  RAVEL - 2007                                                                           */
/*  Classfile rs232.c                                                                      */
/*                                                                                         */
/*  Class to use the RS232                                                                 */
/*-----------------------------------------------------------------------------------------*/

/******** INCLUDES ***************************************************************************/
#include "definitions.h"
#include "cc2420.h"
#include "rs232.h"
#include "variables.h"
#include "ravelfunclib.h"
#include "vtxprintf.h"
#include <io.h>
#include <signal.h>
#include <string.h>


/******** FUNCTION PROTOTYPES ****************************************************************/
static void rs232_getEvent(void);


/******** FUNCTIONS **************************************************************************/

/*****************************************************************************************/
/* Initialisation of the USART0 [for mode RS232].                                        */
/*                                                                                       */
/* @DATE      : 09.11.2007                                                               */
/* @AUTHOR    : Ravel                                                                    */
/* @INPUT     : -                                                                        */
/* @OUPUT     : -                                                                        */
/* @REVISION  : v1.0                                                                     */
/*****************************************************************************************/
void rs232_init (void)
{
  U0CTL = 0;
  U0CTL = SWRST;                          // Stop USART0 --> SET init state of U0CTL
  P3SEL |= BIT4 + BIT5;                   // Sel periph. func. of P3.4,5 - USART0 TXD,RXD
  P3DIR &= ~BIT5;                         // Input  P3.5 - URXD0
  P3DIR |= BIT4;                          // Output P3.4 - UTXD0
  U0CTL = CHAR;                           // 8-bit character

  U0TCTL = SSEL1;                         // BRCLK = SMCLK (ca 1 MHz)

  UBR00=0x24; UBR10=0x00; UMCTL0=0x10;    // uart0 4160000Hz 115235bps
  //UBR00=0x09; UBR10=0x00; UMCTL0=0x00;  // uart0 1042000Hz 115777bps

  U0ME |= UTXE0 + URXE0;                  // Enable USART0 TXD,RXD

  U0CTL &= ~SWRST;                        // Start USART0 --> SET SWRST to 0
  IE1 |= URXIE0;                          // Enable RX interrupt
}

/*****************************************************************************************/
/* Send a character.                                                                     */
/*                                                                                       */
/* @DATE      : 09.11.2007                                                               */
/* @AUTHOR    : Ravel                                                                    */
/* @INPUT     : char c > Character (ASCII)                                               */
/* @OUPUT     : -                                                                        */
/* @REVISION  : v1.0                                                                     */
/*****************************************************************************************/
void rs232_putc (char c) {
  while (!(IFG1 & UTXIFG0)) ;
  U0TXBUF = c;
}

/*****************************************************************************************/
/* Print out a message of characters.                                                    */
/*                                                                                       */
/* @DATE      : 09.11.2007                                                               */
/* @AUTHOR    : Ravel                                                                    */
/* @INPUT     : const char * fmt > String composition                                    */
/*              ...              > Parameters                                           */
```

```c
/* @OUPUT     : -                                                                  */
/* @REVISION  : v1.0                                                               */
/*********************************************************************************/
void rs232_printf(const char *fmt, ...) {
  va_list ap;
  va_start(ap, fmt);
  vtxprintf(rs232_putc, fmt, ap);
  va_end(ap);
}

/*********************************************************************************/
/* [static] - Proves the rs232_buffer and calculate the event.                    */
/* (call from rs232 interrupt routine!)                                            */
/* -> 001 : Set parking state                 - TX cfg-001:X;       - RX rsp-001:OK;      */
/* -> 002 : Set parking number                - TX cfg-002:XXX;     - RX rsp-002:OK;      */
/* -> 003 : Get parking number (over RS232)   - TX cfg-003;         - RX rsp-003:XXX;     */
/* -> 004 : Set Src IEEE address              - TX cfg-004:HH...LL; - RX rsp-004:OK;      */
/* -> 005 : Get Src IEEE address (over RS232) - TX cfg-005;         - RX rsp-005:HH...LL; */
/* -> 006 : Set Dst IEEE address              - TX cfg-006:HH...LL; - RX rsp-006:OK;      */
/* -> 007 : Get Dst IEEE address (over RS232) - TX cfg-007;         - RX rsp-007:HH...LL; */
/* -> 008 : Set Dst PAN-ID                    - TX cfg-008:XXXX;    - RX rsp-008:OK;      */
/* -> 009 : Get Dst PAN-ID (over RS232)       - TX cfg-009;         - RX rsp-009:XXXX;    */
/*                                                                                 */
/* @DATE      : 09.11.2007                                                         */
/* @AUTHOR    : Ravel                                                              */
/* @INPUT     : -                                                                  */
/* @OUPUT     : -                                                                  */
/* @REVISION  : v1.0                                                               */
/*********************************************************************************/
static void rs232_getEvent(void)
{
  uint8_t i;
  uint8_t event = 0;

  if(gDebug == GENERAL)
  {
    // print out rs232_buffer
    rs232_printf("rs232_buffer [%u] : ", RS232_BUFFER_SIZE);
    for(i = 0; i < RS232_BUFFER_SIZE; i++)
    {
      rs232_printf("%2x ", rs232_buffer[i]);
    }
    rs232_printf("\r\n");
  }

  if((rs232_buffer[0] == 'c') && (rs232_buffer[1] == 'f') && (rs232_buffer[2] == 'g') && (rs23
  {
    event = (((uint8_t) rs232_buffer[4] - 48) * 100) + (((uint8_t) rs232_buffer[5] - 48) * 10)
  }

  if(gDebug == GENERAL) rs232_printf("Event : %u\r\n", event);

  if(event == 1)
  {
    gState = rs232_buffer[8];
    gStateChanged = 1;
    if(gDebug != VIZGRAPH) rs232_printf("rsp-001:OK;\r\n");
  }
  else if(event == 2)
  {
    gParkingNumber = (((uint8_t) rs232_buffer[8] - 48) * 100) + (((uint8_t) rs232_buffer[9] -
    if(gDebug != VIZGRAPH) rs232_printf("rsp-002:OK;\r\n");
  }
  else if(event == 3)
  {
    if(gDebug != VIZGRAPH) rs232_printf("rsp-003:%3u;\r\n", gParkingNumber);
  }
  else if(event == 4)
  {
    for(i = 0; i < 4; i++)
    {
      gSrcIEEEAddr[i] = ((((uint16_t) setAscii2DecHexValue(rs232_buffer[i+i+i+i+8])) << 12)
                        | (((uint16_t) setAscii2DecHexValue(rs232_buffer[i+i+i+i+9])) << 8)
                          | (((uint16_t) setAscii2DecHexValue(rs232_buffer[i+i+i+i+10])) << 4
                            | ((uint16_t) setAscii2DecHexValue(rs232_buffer[i+i+i+i+11])));
    }
```

```
    cc2420_set_reg_ram(0x60, 4, gSrcIEEEAddr);
    if(gDebug != VIZGRAPH) rs232_printf("rsp-004:OK;\r\n");
  }
  else if(event == 5)
  {
    if(gDebug != VIZGRAPH)
    {
      rs232_printf("rsp-005:");
      for(i = 0; i < 4; i++)
      {
        rs232_printf("%4x", gSrcIEEEAddr[i]);
      }
      rs232_printf(";\r\n");
    }
  }
  else if(event == 6)
  {
    for(i = 0; i < 4; i++)
    {
      gDstIEEEAddr[i] = ((((uint16_t) setAscii2DecHexValue(rs232_buffer[i+i+i+i+8])) << 12)
                         | (((uint16_t) setAscii2DecHexValue(rs232_buffer[i+i+i+i+9])) << 8)
                           | (((uint16_t) setAscii2DecHexValue(rs232_buffer[i+i+i+i+10])) << 4
                             | ((uint16_t) setAscii2DecHexValue(rs232_buffer[i+i+i+i+11])));
    }
    if(gDebug != VIZGRAPH) rs232_printf("rsp-006:OK;\r\n");
  }
  else if(event == 7)
  {
    if(gDebug != VIZGRAPH)
    {
      rs232_printf("rsp-007:");
      for(i = 0; i < 4; i++)
      {
        rs232_printf("%4x", gDstIEEEAddr[i]);
      }
      rs232_printf(";\r\n");
    }
  }
  else if(event == 8)
  {
    gDstPANId = ((((uint16_t) setAscii2DecHexValue(rs232_buffer[8])) << 12)
                 | (((uint16_t) setAscii2DecHexValue(rs232_buffer[9])) << 8)
                   | (((uint16_t) setAscii2DecHexValue(rs232_buffer[10])) << 4)
                     | ((uint16_t) setAscii2DecHexValue(rs232_buffer[11])));
    cc2420_set_reg_ram(0x68, 1, (uint16_t *) gDstPANId);
    if(gDebug != VIZGRAPH) rs232_printf("rsp-008:OK;\r\n");
  }
  else if(event == 9)
  {
    if(gDebug != VIZGRAPH) rs232_printf("rsp-009:%4x;\r\n", gDstPANId);
  }
}

/*****************************************************************************/
/* [INTERRUPT] - ISR by receiving a character over the USART0.               */
/*                                                                            */
/* @DATE      : 09.11.2007                                                    */
/* @AUTHOR    : Ravel                                                         */
/* @INPUT     : -                                                             */
/* @OUPUT     : -                                                             */
/* @REVISION  : v1.0                                                          */
/*****************************************************************************/
interrupt (USART0RX_VECTOR) rs232_isr(void)
{
  if(gRs232_buffer_ptr == RS232_BUFFER_SIZE)
  {
    gRs232_buffer_ptr = 0;
  }

  if(U0RXBUF == ';')
  {
    rs232_buffer[gRs232_buffer_ptr] = U0RXBUF;
    rs232_getEvent();
    gRs232_buffer_ptr = 0;
  }
  else
```

```
229    {
230      rs232_buffer[gRs232_buffer_ptr] = U0RXBUF;
231      gRs232_buffer_ptr++;
232    }
233  }
234
```

```c
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Headerfile sleep.h                                                               */
/*                                                                                   */
/*  Class who administred the lower energy consumation.                              */
/*-----------------------------------------------------------------------------------*/

#ifndef _SLEEP_
#define _SLEEP_


/******** INCLUDES ***************************************************************/
#include <io.h>


/******** FUNCTIONS **************************************************************/
extern void sleep(void);


#endif

```

```c
/*-----------------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                       */
/*  RAVEL - 2007                                                                           */
/*  Classfile sleep.c                                                                      */
/*                                                                                         */
/*  Class who administred the lower energy consumation.                                    */
/*-----------------------------------------------------------------------------------------*/

/******** INCLUDES ***************************************************************************/
#include "sleep.h"


/******** DEFINITIONS ************************************************************************/
//-- IO CC2420
#define FIFO          (1<<5)
#define FIFOP         (1<<6)
#define CCA           (1<<7)
#define SDF           (1<<0)
#define CMDE_NRST     (1<<1)
#define VREG_EN       (1<<2)
#define SPI_CS        (1<<0)
#define SPI_MOSI      (1<<1)
#define SPI_MISO      (1<<2)
#define SPI_CLK       (1<<3)

#define AMR_XY_SET    BIT5                  // <P4.5>
#define AMR_XY_RESET  BIT4                  // <P4.4>
#define AMR_XY_NWP    BIT0                  // <P4.0>
#define AMR_Z_SET     BIT2                  // <P4.2>
#define AMR_Z_RESET   BIT3                  // <P4.3>
#define AMR_Z_NWP     BIT1                  // <P4.1>


/******** FUNCTIONS **************************************************************************/

/*****************************************************************************************/
/* Set the microcontroller in a low consomation mode.                                    */
/*                                                                                       */
/* @DATE      : 27.11.2007                                                               */
/* @AUTHOR    : Ravel                                                                    */
/* @INPUT     : -                                                                        */
/* @OUPUT     : -                                                                        */
/* @REVISION  : v1.0                                                                     */
/*****************************************************************************************/
void sleep(void)
{
  //--- Disable interrupts
  _DINT();

  //--- Reset the ports
  P1SEL = 0;
  P2SEL = 0;
  P3SEL = 0;
  P4SEL = 0;
  P5SEL = 0;
  P6SEL = 0;

  P1DIR  = 0;
  P2DIR  = 0;
  P3DIR  = 0;
  //P4DIR  = 0;
  P5DIR  = 0;
  P6DIR  = 0;

  //--- CC2420 Low Consomation
  P1OUT  &= ~(FIFO | FIFOP | CCA);
  P1DIR  |= (FIFO | FIFOP | CCA);
  P2OUT  &= ~(SDF | CMDE_NRST | VREG_EN);
  P4DIR |= AMR_XY_RESET & AMR_Z_RESET & AMR_XY_SET & AMR_Z_SET & AMR_XY_NWP & AMR_Z_NWP;
  P4OUT |= (AMR_XY_SET & AMR_Z_SET & AMR_XY_RESET & AMR_Z_RESET);
  P4OUT &= ~(AMR_XY_NWP + AMR_Z_NWP);
  P2OUT  &= (CMDE_NRST | VREG_EN);
  P2DIR  |= (CMDE_NRST | VREG_EN);
  P5OUT |= (SPI_CS);
  P5DIR  |= (SPI_CS);
  P5OUT &= ~( SPI_MOSI | SPI_MISO | SPI_CLK);
```

```
77    P5DIR  |= ( SPI_MOSI | SPI_MISO | SPI_CLK);
78
79    //--- Set microcontroller in low power mode and enable the interrupts
80    _BIS_SR(LPM3_bits + GIE);
81  }
82
83
84
```

```c
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Headerfile spi.h                                                                 */
/*                                                                                   */
/*  Class to use the SPI-Bus                                                         */
/*-----------------------------------------------------------------------------------*/

#ifndef _SPI_H_
#define _SPI_H_

/******** INCLUDES ***************************************************************/
#include <stdint.h>
#include <io.h>


/******** CONSTANTS **************************************************************/
#define CS0_ENABLE      P5OUT &= ~BIT0      // Select the slave (low level enabled)  SPI_CS
#define CS0_DISABLE     P5OUT |= BIT0       // Select the slave (high level enabled) SPI_CS


/******** FUNCTIONS **************************************************************/
void spi_init(void);
void spi_send(uint8_t);
char spi_receive(void);

#endif  // endif _SPI_H_

```

```c
/*-----------------------------------------------------------------------------------*/
/*  SPI Library for CZAME parking system                                             */
/*  RAVEL - 2007                                                                     */
/*                                                                                   */
/*  Use the USART1 of the MSP430F169 in mode SPI                                     */
/*-----------------------------------------------------------------------------------*/

/******** INCLUDES ***************************************************************/
#include "definitions.h"
#include "spi.h"
#include <io.h>
#include <stdint.h>


/******** INTERN CONSTANTS *******************************************************/
#define CS        BIT0                      // Chip Select
#define MOSI      BIT1                      // Master Out Slave In
#define MISO      BIT2                      // Master In Slave Out
#define CLK       BIT3                      // Clock


/******** FUNCTIONS **************************************************************

/***********************************************************************************/
/* Initialisation of the USART1 [for mode SPI].                                    */
/*                                                                                 */
/* @DATE      : 09.11.2007                                                         */
/* @AUTHOR    : Ravel                                                              */
/* @INPUT     : -                                                                  */
/* @OUPUT     : -                                                                  */
/* @REVISION  : v1.0                                                               */
/***********************************************************************************/
void spi_init(void)
{
  UCTL1 |= SWRST;                          // Reset the USART so as to configure the SPI

  P5SEL |= (MOSI + MISO + CLK);            // Select peripheral func. options for the SPI
                                           //  (CLK, MOSI and MISO)
  P5OUT |= CS;                             // CS on level 1 to invalidate the slave
  P5OUT &= ~(CLK + MOSI);                  // MOSI and CLK to gound (bus in static cond.)
  P5DIR |= CS + CLK + MOSI;                // CS, MOSI and CLK as output
  P5DIR &= ~MISO;                          // MISO as input

  /*********************************************************************************/
  /*           ATTENTION : THE CC2420 FUNCTIONS ONLY WITH THIS CONFIGURATION!      */
  /*          DON'T PUT ONLY CKPL THAN YET CHRONOGRAM BUT THE STATIC CONDITION     */
  /*        OF THE SPI IS DIFFERENT AND ACCORDINGLY NOT COMPREHEND BY THE CC2420   */
  /*********************************************************************************/

  U1CTL |= CHAR + SYNC + MM;               // 8 Bit + LISTEN (to remove) + SPI + Master
  U1TCTL |= CKPH + SSEL1 + STC;            // Use SMCLK

  UBR01 = 0x02;                            // SMCLK/2 = 500 kHz  (MCLK : Master clock)
  UBR11 = 0x00;                            // 0
  UMCTL1 = 0x00;                           // No modulation (not used by SPI)

  ME2 = USPIE1;                            // Validation the mode SPI of the USART1

  UCTL1 &= ~SWRST;                         // Re-starting of the USART and the SPI
}

/***********************************************************************************/
/* Send a character over the SPI bus.                                              */
/*                                                                                 */
/* @DATE      : 09.11.2007                                                         */
/* @AUTHOR    : Ravel                                                              */
/* @INPUT     : uint8_t data > Character to send                                   */
/* @OUPUT     : -                                                                  */
/* @REVISION  : v1.0                                                               */
/***********************************************************************************/
void spi_send(uint8_t data)
{
  IFG2 &= ~(URXIFG1);                      // Set interrupt flag return to zero
  U1TXBUF = data;                          // Sending a value
  while (!(IFG2&URXIFG1));                 // Wait for the end of the transmission
}
```

```c

/*****************************************************************************/
/* Receive a character over the SPI bus and write him in the RX buffer.      */
/*                                                                           */
/* @DATE      : 09.11.2007                                                   */
/* @AUTHOR    : Ravel                                                        */
/* @INPUT     : -                                                            */
/* @OUPUT     : char < Character                                             */
/* @REVISION  : v1.0                                                         */
/*****************************************************************************/
char spi_receive(void)
{
  uint8_t data;
  IFG2 &= ~(URXIFG1);                       // Set interrupt flag return to zero
                                            //  (becase in listen mode)
  U1TXBUF = 0;                              // Set the buffer return to zero so that no data
                                            //  is transmitted
  while (!(IFG2&URXIFG1));                  // Wait of a reception
  data = U1RXBUF;                           // Reading of the data
  return data;
}

```

```c
/*-----------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                           */
/*  RAVEL - 2007                                                               */
/*  Headerfile variables.h                                                     */
/*                                                                             */
/*  Globale variables                                                          */
/*-----------------------------------------------------------------------------*/

#ifndef _VARIABLES_H_
#define _VARIABLES_H_


/******** INCLUDES ***************************************************************/
#include <stdint.h>
#include <stdbool.h>


/******** CONSTANTS **************************************************************/
#define RF2_BUFFER_SIZE      256
#define RS232_BUFFER_SIZE    32


/******** GLOBAL VARIABLES *******************************************************/

//--- DEBUG VARIABLES ---------------------------------------------------------
typedef enum { NONE, GENERAL, VIZGRAPH } debug_mode_t;

//--- GENERAL VARIABLES -------------------------------------------------------
extern volatile bool gState;                // actually parking state
extern volatile bool gState_s;                // actually parking state
extern const uint16_t gParkingNumber;       // parking number
extern const uint16_t gGlitch;

//--- MAC VARIABLES -----------------------------------------------------------
extern uint8_t gSequenceNbr;                // IEEE 802.15.4 frame sequence number
extern uint16_t gSrcIEEEAddr[4];            // IEEE 802.15.4 extended source address
extern uint16_t gDstIEEEAddr[4];            // IEEE 802.15.4 extended destination address
extern uint16_t gDstPANId;                  // IEEE 802.14.4 destination pan id

//--- RF2 VARIABLES -----------------------------------------------------------
extern uint8_t rf2_buffer[RF2_BUFFER_SIZE]; // RF buffer for IEEE 802.15.4 frames

//--- RS232 VARIABLES ---------------------------------------------------------
extern uint8_t rs232_buffer[RS232_BUFFER_SIZE];   // RS232 buffer
extern volatile uint8_t gRs232_buffer_ptr;        // RS232 buffer pointer

//--- AMR VARIABLES -----------------------------------------------------------
extern uint32_t z_moy;
extern int16_t gX_offset;
extern int16_t gY_offset;
extern int16_t gZ_offset;


#endif //_VARIABLES_H_
```

```c
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Classfile variables.c                                                            */
/*                                                                                   */
/*  Globale variables                                                                */
/*-----------------------------------------------------------------------------------*/

/******** INCLUDES ***************************************************************/
#include "variables.h"


/******** GLOBAL VARIABLES *******************************************************/

//--- GENERAL VARIABLES ------------------------------------------------------------
volatile bool gState_s = false;              // actually state
volatile bool gState = false;                // actually state
uint16_t const gParkingNumber = 1;           // parking number
uint16_t const gGlitch = 2;

//--- MAC VARIABLES ----------------------------------------------------------------
uint8_t gSequenceNbr = 0;                   // sequence number of frame
uint16_t gSrcIEEEAddr[4] = { 0x1234, 0x5678, 0x9012, 0x3456 };
//uint16_t gDstIEEEAddr[4] = { 0x0013, 0xA200, 0x4008, 0xF72D };
uint16_t gDstIEEEAddr[4] = { 0xCCDD, 0xAABB, 0x8899, 0x6677 };
uint16_t gDstPANId = 0x4012;

//--- RF2 VARIABLES ----------------------------------------------------------------
uint8_t rf2_buffer[RF2_BUFFER_SIZE];

//--- RS232 VARIABLES --------------------------------------------------------------
uint8_t rs232_buffer[RS232_BUFFER_SIZE];
volatile uint8_t gRs232_buffer_ptr = 0;

//--- AMR VARIABLES ----------------------------------------------------------------
int16_t gX_offset;
int16_t gY_offset;
int16_t gZ_offset;

```

```c
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Headerfile vtxprintf.h                                                           */
/*                                                                                   */
/*  Reduced printf-like function for embeded softwares                               */
/*-----------------------------------------------------------------------------------*/

#ifndef _VTXPRINTF_H_
#define _VTXPRINTF_H_

/* Reduced printf-like function for embeded softwares */

#include <stdarg.h>

/*
   %%   => print percent
   %c   => print a char
   %7c  => do it 7 times (unlike std. %7c)
   %s   => print a string
   %7s  => do it possibly right padded by spaces
   %u   => print an unsigned number
   %7u  => do it possibly left padded by zeros (like std. %07u)
   %lu  => print an unsigned long number
   %j   => print a signed number always prefixed by a sign
   %b   => print a number as binary
   %x   => print a number as hexadecimal (like std. %X)
*/

/******** FUNCTIONS ***************************************************************/
void vtxprintf (void (*)(char), const char *, va_list);

#endif

```

```c
/*-----------------------------------------------------------------------------------*/
/*  Library for CZAME parking system                                                 */
/*  RAVEL - 2007                                                                     */
/*  Headerfile vtxprintf.h                                                           */
/*                                                                                   */
/*  Reduced printf-like function for embeded softwares                               */
/*-----------------------------------------------------------------------------------*/

/******** INCLUDES ***************************************************************/
#include "vtxprintf.h"


/******** CONSTANTS **************************************************************/
#define SIGNED 1
#define LARGE  2


/******** FUNCTIONS **************************************************************/

/***********************************************************************************/
/* Calculate the output format for the String.                                     */
/*                                                                                 */
/* @DATE      : 09.11.2007                                                         */
/* @AUTHOR    : Ravel                                                              */
/* @INPUT     : void (*tx_char)(char) > void pointer function with character       */
/*              const char *fmt       > constant pointer with data (string)        */
/*              va_list    ap         > attributes                                 */
/* @OUPUT     : -                                                                  */
/* @REVISION  : v1.0                                                               */
/***********************************************************************************/
void vtxprintf (void (*tx_char)(char), const char *fmt, va_list ap) {

  int i;
  int base;
  int width;
  int flags;

  unsigned long x, q;
  unsigned char r;

  const char * str;
  const char digits[] = "0123456789ABCDEF";
  char tmp[36];

  for ( ; *fmt ; fmt++ ) {

    /* Verbatim print all other chars */
    if (*fmt != '%') { tx_char(*fmt); continue; }

    fmt++;      /* Skip percent char */
    flags = 0;  /* Unsigned short number by default */

    /* Get field width if exists */
    width = 0;
    while ('0' <= *fmt && *fmt <= '9')
      width = (((width << 2) + width) << 1) + (*fmt++)-'0';

    /* Get conversion qualifier */
    if (*fmt == 'l') { flags |= LARGE; fmt++; }

    switch (*fmt) {

    case '%':
      tx_char('%');
      continue;

    case 'c':
      /* Repeat print char as width */
      i = va_arg(ap, int);
      if (!width) width = 1;
      while (width--) tx_char(i);
      continue;

    case 's':
      /* Space padding after string if needed */
      str = va_arg(ap, char *);
```

```
      while (*str) { tx_char(*str++); width--; }
      if (0 < width) while (width--) tx_char(' ');
      continue;

    case 'b': base = 2; break;
    case 'j': flags |= SIGNED;
    case 'u': base = 10; break;
    case 'x': base = 16; break;

    default:
      continue;
    }

    /* Get converted number */
    x = (flags & LARGE) ? va_arg(ap, unsigned long) : va_arg(ap, unsigned int);

    i = 0;

    /* Print sign if signed number */
    if (flags & SIGNED) {
      if ((signed long)x < 0)
      { tx_char('-'); x = -(signed long)x; }
      else
      { tx_char('+'); }
    }

    switch (base) {

    /* Binary number (base 2) */
    case 2:
      while (x) {
        q = x >> 1; r = x & 1;
        tmp[i++] = digits[r];
        x = q;
      }
      break;

    /* Decimal number (base 10) */
    case 10:
      while (x) {
        q = ((x >> 1) + x) >> 3;
        q = (q >>  4) + q;
        q = (q >>  8) + q;
        q = (q >> 16) + q;
        q >>= 1;
        r = x - (((q << 2) + q) << 1);
        if (10 <= r) { r -= 10; q++; }
        tmp[i++] = digits[r];
        x = q;
      }
      break;

    /* Hexadecimal number (base 16) */
    case 16:
      while (x) {
        q = x >> 4; r = x & 15;
        tmp[i++] = digits[r];
        x = q;
      }
      break;
    }

    /* Zero padding before number if needed */
    if (i < width)
    { width -= i; while (width--) tx_char('0'); }

    /* Print the number */
    while (i--) tx_char(tmp[i]);

  }  /* main loop */

} /* vtxprintf */
```